NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R S.C. 1970, c C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c C-30, et ses amendements subséquents

Canada

Proposition and Security Evaluation of an Analog Speech Scrambling Device for Secure Speech Communications

Alexandros K. Goniotakis

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering at
Concordia University
Montréal, Québec, Canada

March, 1990

ISBN 0-315-59163-3

ABSTRACT

Proposition and Security Evaluation of a an Analog Speech Scrambling Device for Secure Speech Communications

Alexandros K. Goniotakis

The communication environment is very widely used, and very vulnerable to abuse in the form of intrusion. The solution to this problem is to secure the environment. This is made possible through the use of data cipher systems and speech secure systems, depending on the form of the information to be protected.

One particular category of speech secure systems are analog speech scrambling systems. In general, an analog speech scrambling system consists of an analog speech scrambling algorithm operated in a pseudorandom fashion. One way of obtaining this pseudorandom element is through the use of a keystream generator and an initial key.

An analog speech scrambling algorithm, consisting of a hopping filters algorithm, and an amplitude multiplication scrambling algorithm, is proposed and evaluated as most secure with respect to other analog speech scrambling algorithms, on the basis of certain specified criteria and operational constraints.

A particular DES (Data Encryption Standard)-based keystream generator algorithm is then developed so as to satisfy certain criteria for a secure keystream generator. This keystream generator algorithm in question is not only compatible with the analog speech scrambling algorithm, but the combined operation is declared secure even for a "known original speech" attack.

The two proposed algorithms make up the major component of the proposed analog speech scrambling device. The specification of the device is completed with the introduction of timing and synchronization circuitry required for speech signal recovery, as well as a user interface to facilitate the use of this device.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. A.K. Elhakeem for his guidance and support throughout the course of this work. Not only did he lead me to an area of concentration which I wanted to enter and new nothing about, but was also patient in waiting for me to develop the required background in this area. Furthermore, he allowed me the freedom to build on this background, and to develop my own ideas, while at the same time providing me with ample suggestions, and comments necessary for keeping me on the right track.

I deeply appreciate his constant supply of motivation and encouragement, which allowed me to see this work to the end, and hope that he may look upon this work as a small reward of my appreciation for his dedication in the field of communications.

Last but not least, I would like to acknowledge my family's support and patience for the countless hours which I was unable to spend with them, as a result of this work.

To My Father and Mother
Kostas and Efthalia Goniotakis,
My Sisters
Maria and Catherine Goniotaki,
and My Aunt
Andonia Papadaki.

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

T_c	The keystream bit duration
HF	Hopping Filters
T_F	The time duration between HF
$m(t)$	The stochastic process representing speech
$\hat{m}_1(t)$	The speech process sub-band between 200 Hz and 400 Hz
$\hat{m}_2(t)$	The speech process sub-band between 400 Hz and 600 Hz
$\hat{m}_3(t)$	The speech process sub-band between 600 Hz and 800 Hz
$\hat{m}_4(t)$	The speech process sub-band between 800 Hz and 1100 Hz
$\hat{m}_5(t)$	The speech process sub-band between 1100 Hz and 1400 Hz
$\hat{m}_6(t)$	The speech process sub-band between 1400 Hz and 1800 Hz
$\hat{m}_7(t)$	The speech process sub-band between 1800 Hz and 2400 Hz
$\tilde{m}(t)$	The speech process sub-band between 200 Hz and 2400 Hz
PAM	Pulse Amplitude Modulation
$d_{1F}(t)$, ..., $d_{7F}(t)$	The PAM processes manipulating the speech sub-bands $\hat{m}_1(t)$, ..., $\hat{m}_7(t)$ respectively
$g_F(t-kT_F)$	The gate function of width T_F for PAM processes $d_{1F}(t)$, ..., $d_{7F}(t)$
$D_{1F}(k)$, ..., $D_{7F}(k)$	The discrete-time stochastic processes representing $d_{1F}(t)$, ..., $d_{7F}(t)$ respectively
$y_f(t)$	The stochastic process representing the output of the HF algorithm
AAS	Amplitude Addition Scrambling
AMS	Amplitude Multiplication Scrambling
T_A	The time duration between AAS or AMS level changes

$\mathbf{d_A}(t)$	The PAM process representing the AAS or AMS algorithm
$g_A(t-kT_A)$	The gate function of width T_A for the PAM process $\mathbf{d_A}(t)$
$\mathbf{D_A}(k)$	The discrete-time stochastic process representing the PAM process $\mathbf{d_A}(t)$
$\mathbf{y_a}(t)$	The stochastic process representing the output of the AAS algorithm
$\mathbf{y_m}(t)$	The stochastic process representing the output of the AMS algorithm
HF-AAS	Hopping Filters-Amplitude Addition Scrambling
HF-AMS	Hopping Filters-Amplitude Multiplication Scrambling
$\mathbf{z_a}(t)$	The stochastic process representing the output of the HF-AAS algorithm
$\mathbf{z_m}(t)$	The stochastic process representing the output of the HF-AMS algorithm
$\mathbf{v}(t)$	The output process of an analog speech scrambling algorithm in general
τ	The autocorrelation index
$C_{mv}(\tau)$	The cross-covariance between input $\mathbf{m}(t)$ and $\mathbf{v}(t)$ processes of an analog speech scrambling algorithm in general
E	Expectation operator
$\mu_{1F}, \ldots, \mu_{7F}$	The means of the PAM processes $\mathbf{d_{1F}}(t), \ldots, \mathbf{d_{7F}}(t)$ respectively
μ_F	The mean of the HF PAM processes collectively
μ_A	The mean of the AAS or AMS PAM process
C_{mz_a}	The cross-covariance of the HF-AAS algorithm

C_{mz_m}	The cross-covariance of the HF-AMS algorithm
C_{my_f}	The cross-covariance of the HF algorithm
C_{my_m}	The cross-covariance of the AMS algorithm
C_{my_a}	The cross-covariance of the AAS algorithm
$R_w(\tau)$	The autocorrelation of white noise
$R_{z_a z_a}(\tau)$	The autocorrelation of the output of the HF-AAS algorithm
$R_{m_i}(\tau)$	The autocorrelation of the ith filtered speech process
$R_{m_i m_j}(\tau)$	The cross-correlation of the ith and jth filtered speech processes
$R_{d_{iF}}(\tau)$	The autocorrelation of the ith out of the seven PAM processes of the HF algorithm
$R_{d_A}(\tau)$	The autocorrelation of the PAM process of the AAS or AMS algorithm
$R_{d_F}(\tau)$	The autocorrelation of all seven PAM processes of the HF algorithm collectively
$R_{z_m z_m}(\tau)$	The autocorrelation of the output of the HF-AMS algorithm
$R_m(\tau)$	The autocorrelation of input speech
$R_{y_f y_f}(\tau)$	The autocorrelation of the output of the HF algorithm
$R_{y_m y_m}(\tau)$	The autocorrelation of the output of the AMS algorithm
WSS	Wide Sense Stationary
H(f)	The transfer function of a time-invariant linear network
$S_{m_j}(f)$	The power spectral density of the jth filterd speech process
$S_m(f)$	The power spectral density of input speech

$X(f)$	The unity gain bandpass filter with bandwidth 200 Hz - 2400 Hz
F	The Fourier transform operator
F^{-1}	The inverse Fourier transform operator
$R_m[0]$	The average power of speech
$C_p(\tau)$	The time-averaged autocorrelation of speech normalized by the average power of speech
s	Discrete time units
c	The duty cycle factor of a PAM process
T	The duration of a PAM process level in general
$g(\tau-sT_F)$	The gate function of width T_F used as a window on $R_{d_F}(\tau)$
$R_{d_F}[s]$	The discrete-time autocorrelation of the PAM processes of the HF algorithm
$g(\tau-sT_A)$	The gate function of width T_A used as a window on $R_{d_A}(\tau)$
$R_{d_A}[s]$	The discrete-time autocorrelation of the PAM process of the AMS algorithm
α	Collective reference to the PAM levels of the PAM processes of the HF algorithm
k_F	The number of PAM levels of each PAM process of the HF algorithm
$\pm p_l$	Two additive inverse levels of a PAM process of the HF algorithm
D_{F_0}	The initial level of a PAM process of the HF algorithm
D_{F_s}	The level of a PAM process of the HF algorithm at time s
β	Collective reference to the PAM levels of the PAM process of the AMS algorithm

k_A	The number of PAM levels of the PAM process of the AMS algorithm
$\pm q_l$	Two additive inverse levels of the PAM process of the AMS algorithm
D_{A_0}	The initial level of the PAM process of the AMS algorithm
D_{A_s}	The level of the PAM process of the AMS algorithm at time s
$R_{d_F}[0]$	The average power of each PAM process of the HF algorithm
$R_{d_A}[0]$	The average power of the PAM process of the AMS algorithm
NBS	National Bureau of Standards
DES	The Data Encryption Standard algorithm
ϕ	Euler's function
SK1	Secret key number 1
SK2	Secret key number 2
KS1	Keystream resulting from simulation of generator scheme without DES key scheduling algorithm, or alternate output complementation using SK1
KS1'	Keystream resulting from simulation of generator scheme with DES key scheduling algorithm, but without alternate output complementation using SK1
KS1''	Keystream resulting from simulation of proposed generator scheme using SK1
KS2''	Keystream resulting from simulation of proposed generator scheme using SK2
$R_{KS1}(\tau)$	Autocorelation of KS1

$R_{KS1'}(\tau)$	Autocorrelation of KS1'
$R_{KS1''}(\tau)$	Autocorrelation of KS1''
$R_{KS1''\,KS2''}(\tau)$	Cross-correlation of KS1'' and KS2''

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 THE COMMUNICATION ENVIRONMENT

Communication is the link between individual entities. It is the transfer of information from a source to a destination through a medium. The transmitter (source), the receiver (destination), the information conveyed between them, and the medium to convey this information are all embodied within a communication environment. The participants of the communication environment, which can at any one time act as sources or destinations of the information being communicated, may be people, or computers acting on behalf of people. The form of the information communicated between people, and the form of the information communicated between computers is very different, but can be made to be similar. People communicate with speech, which is naturally produced while computers communicate with data. The major component of the communication environment is the telecommunication system. It is what allows the information to be transferred, from one participant within this environment to another such participant. It mainly consists of a network made up of a hierarchy of switching offices, and appropriate transmission media, which allow analog and/or digital transmission of information.

1.2 INTRUSION TO THE ENVIRONMENT

Due to the great dependency of the general public on the telecommunication system it is not only impossible to do without this system, and hence the communication environment embodying it, but it can prove very disastrous, if the normal or expected operation of this environment is intruded upon with malicious intent.

1.2.1 Two Forms of Intrusion

Intrusion can take one of two forms. One form can be referred to as active intrusion. This consists of a third party attempting to obstruct the communication path between two users, so that the information does not reach the intended receivers. The effect of this on the users can range from frustrating to disastrous, depending on the gravity of the information being communicated. The important thing though is that this form of intrusion does not pass unnoticed. The users will become aware of it, and thus informed can take appropriate measures to overcome it, or find other means of communication.

The other major form of intrusion, and the concern of this work, can be referred to as passive intrusion. This consists of a third party receiving information, not intended for him/her, from the communication path set up between two unsuspecting users. The effect of this on the users also ranges from frustrating to disastrous, depending on the gravity of the information being communicated, but on a greater scale than before. This is because in this case the communicating parties have a false sense of security, in thinking that everything being communicated is only between them.

1.2.2 Location of Intrusion

Having identified the particular form of intrusion it is appropriate to show where this intrusion can take place, and to what extent is the telecommunication system vulnerable to this form of intrusion. Theoretically, the intrusion can take place anywhere within the telecommunication system along the path between two communicating parties. Practically though, it will most probably take place within the local loop of any one of the communicators, and more specifically, within a junction box closer to the near-end of the local loop relative to a particular communicator. The intrusion, being within the local loop, ensures that only the desired conversation is traveling through this

loop. Being closer to the near-end of the local loop, relative to either communicator respectively, physically identifies the particular local line from other such lines. Finally, it is practically easier to tap into a line at a junction, rather than along the body of this line.

1.2.3 Assessment of Intrusion

Before attempting to secure the communication environment from intrusion, the designer has to assess the possible intrusion by asking certain questions such as: Who is the intruder, and what is the quantity and quality of his/her available resources, which he/she can use to successfully carry out the intrusion? The latter question is directly related to the former, since the more powerful the intruder, the more quantity and/or quality of resources he/she will have at his/her disposal. The power and capabilities of a particular intruder are themselves directly related, to the importance level of the information seeked, and the importance of the involved participants or unsuspecting users. These can range from heads of nations, to diplomats of various levels, to military personnel, to various government and law enforcement agencies, to banks, and other commercial users, or to simply private users [1]. Based on the answers to the above questions, the designer can then decide as to what is the required security level for the particular application, or communication environment. The required security level will be discussed later in more qualitative detail.

1.3 SECURING THE ENVIRONMENT

1.3.1 Theoretical vs. Practical Security

Although investigation of the theoretical security can be interesting, evaluating the practical security of a system is of greater importance. Practical security or insecurity, does not necessarily follow from theoretical security or insecurity, respectively. In

theory, it is assumed that a cryptanalyst has unlimited time, facilities, and funds to compromise a system. If a system proves to be secure even under these conditions, it is then theoretically secure. Although theoretical security can imply practical security this is not always the case. An example of this is a system claiming to have perfect secrecy. Perfect secrecy refers to an ideal level of security with well known properties. With perfect secrecy every element of the message space, through a particular transformation, can map into each and every element of the cryptogram space. The cryptogram space consists of all possible encrypted messages that can result from corresponding messages in the message space. Furthermore, given that a particular cryptogram was received does not give any information in helping a cryptanalyst decide which message was sent. In other words, all cryptograms are statistically independent of all messages [1]. Theoretically, perfect secrecy is achievable. Its definition though inherently implies that for it to be achieved the number of keys should be at least as large as the total number of possible messages. In this context, a key identifies a particular transformation rule from the message space to the cryptogram space. Having at least as many possible transformations as the number of messages will ensure that all messages will map into all cryptograms with equal probabilities, hence ensuring statistical independence between the message space and the cryptogram space. In fact Shannon showed [2] that if a system has the same number of messages, cryptograms, and keys then it has perfect secrecy if, and only if

a) for any given message m and any cryptogram c there is exactly one key transforming m into c (i.e. there is a unique transformation t with $c = t(m)$), and

b) all keys are equally likely [1].

An example of perfect secrecy with five message elements, five cryptogram elements, and five keys is shown in figure 1.1. For a few number of possible messages perfect secrecy can also be achieved practically. On the other hand, for transmitting a reasonable

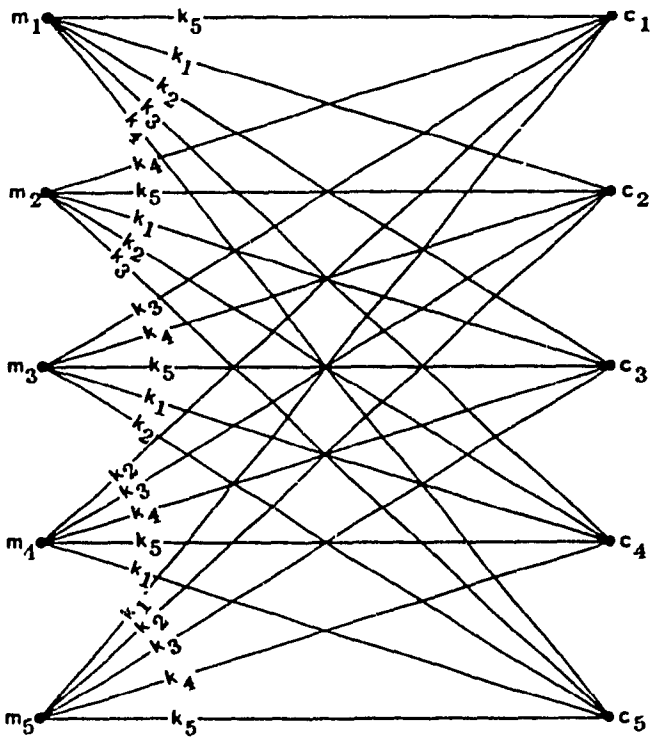

Figure 1.1 An example of perfect secrecy [3].

amount of information a large message space is required, hence requiring an impractically large key space. If such a key space cannot be handled then the particular system in question cannot achieve perfect secrecy in practice.

On the other hand, if a system proves to be insecure under the conditions of unlimited time, facilities, and funds as pertaining to the cryptanalyst, it is then theoretically insecure. Although theoretical insecurity can imply practical insecurity this is not always the case, since the cryptanalyst may, and will be limited in at least one, if not all three factors of time, facilities, and funds.

1.3.2 Practical Required Security Level

Every element of information has some level of importance or value associated with it, such that this value can be translated to cost. The cost will be the cost incurred by an unsuspecting user of the communication environment, whose conversation with another user is intruded upon by a third party, or inversely, the cost gained by the third party, upon successful completion of the intrusion. Related to cost is the cover time of the information element. The cover time is the time within which the above mentioned cost applies to the information, and beyond which the cost of this information reduces to zero. In practice, the required security level can be thought of as having an upper bound and a lower bound. The upper bound on the required security level to protect certain information is governed by the cost, and cover time of this information. In this case knowing the funds and/or facilities at the intruder's disposal is irrelevant This is because it is not economically feasible for the intruder no matter how powerful to pay a higher cost in funds and facilities than the actual cost of the information being secured, or spend more time in compromising the system than the allotted cover time. On the other hand, when considering the lower bound on the required security level, knowing the funds and/or facilities at the intruder's disposal is very relevant If these are known with certainty then the least required security level is that which would require the intruder to either spend more funds, make use of more powerful facilities than those available, or using his/her available resources spend more time than the allotted cover time. If, on the other hand, the funds and/or facilities of the possible intruder cannot be identified with certainty then the worst is assumed. Practically speaking, the intruder in this case will not be thought of as someone who has unlimited funds and facilities, but someone who has large quantities of the above factors with practical limits The practical limits on the available funds and facilities can be the present funds of the most powerful intruder, and the most powerful computers and sophisticated analytical equipment known at present, respectively. In this situation the least required security level is

that which would require the intruder to, either spend more funds, make use of more powerful facilities than the above mentioned practical limits, or using the above practically limited resources spend more time than the allotted cover time. If the lower bound happens to exceed the upper bound then the upper bound prevails. This would be the case of a very powerful intruder attempting to break a very insignificant system. Although the system in this case can be practically broken, there will be a loss incurred by the intruder as a result of the information being worthless. This is not likely to happen.

1.4 THESIS CONTRIBUTION

Having identified the communication environment, and having identified the problem resulting from intrusion to this environment, the expected course of action is to propose a possible solution for practically securing this environment. The general solution is to design a secure system, which can be incorporated in the existing communication environment, and to guarantee that this system can provide the practical security level required. There have been many instances of the solution to this general problem in the past. This has resulted in the design of many possible systems of variable security. At the very least, all this previous work has led to the clarification of the problem, the specification of the various factors contributing to this problem, as well as the general specification constituting a solution to this problem.

The object of this work is to draw on all the positive aspects of previously proposed secure systems, in order to propose yet another secure system for the protection of the communication environment. The proposed secure system is an analog speech scrambling device operated by a digital keystream generator. It consists of a transmitting and a receiving unit, which in turn include a new analog speech scrambling and its corresponding descrambling algorithms, two instances of a keystream generator algorithm, and the appropriate timing and synchronization circuitry respectively. Primarily,

the analog speech scrambling algorithm, and the keystream generator algorithm are each proposed and evaluated independently, each with its own respective security criteria. Following this, the two algorithms are united, forming the major part of the analog speech scrambling device, and the security of their combined operation is also evaluated. Finally, the required timing and synchronization circuitry, as well as a general description of a user interface for this device are put forth for the sake of completeness. It should be noted that this work focuses on the security of the proposed analog speech scrambling device, as opposed to the level of degradation that this device imparts on the speech signal upon recovery.

1.5 THESIS OUTLINE

This work is split into six chapters; four main chapters besides the introductory and concluding chapters. Chapter II consists of an extensive survey of previous related works, which forms a basis for what is to follow. Chapter III evaluates the security of several possible analog speech scrambling algorithms, and as a result proposes a new such algorithm as most secure. Chapter IV puts forth a new DES-based keystream generator algorithm, showing why it was put together as such, and evaluates its security. Finally, chapter V combines the two algorithms as part of one device, and evaluates the security of their combined operation. It then puts forth the timing and synchronization circuitry, as well as a general description of the user interface for this device.

CHAPTER II

SECURE SYSTEMS : SURVEY OF PREVIOUS RELATED WORKS

2.1 DATA CIPHER SYSTEMS

To secure the environment against passive intrusion it is necessary to introduce cipher systems, which will encipher or disguise confidential information entering the telecommunication system, and decipher or reveal disguised information coming from within the telecommunication system. When the information involved is data, these systems are referred to as data cipher systems. The art of designing such systems is known as cryptography, while the process of breaking data cipher systems is referred to as cryptanalysis. Data cipher systems can be classified into various categories based at least on the secrecy of their keys (private or public), the type of transformation which these keys perform (substitution, transposition, or algebraic), and the structure of the entire algorithm (block or stream). These classifications are not depicted in any order of importance (eg. broader to finer classifications and vice versa). Furthermore, not all will apply to all systems.

Private-key and public-key systems as their name implies are distinguished by the secrecy of their enciphering and deciphering keys, involved in the enciphering and deciphering transformations respectively. In the case of a private-key or conventional system, the enciphering and deciphering keys are either identical, or if different, are such that each key can be computed from the other. They are known only by the enciphering and deciphering participants. One of the most famous such systems is the data encryption standard (DES), which will be explained in detail later.

In a public-key system the enciphering and deciphering keys are different. The enciphering key is made public so that any participant can send an enciphered message

to any other participant. The deciphering key on the other hand is secret so that only the intended receiver of the enciphered message can decipher it. An example of this is the RSA cipher system named after its creators Ron Rivest, Adi Shamir, and Len Adleman. This system consists of separating the plaintext message into blocks, and encoding each i-th block into an integer m_i. This integer is then raised to the h-th power modulo n, to produce the ciphertext equivalent $c_i = m_i^h \pmod{n}$. The two integers h and n make up the enciphering key. The deciphering procedure is similarly $m_i = c_i^d \pmod{n}$ where d is secret. The two integers d and n make up the deciphering key. The number n is the product of two large primes p and q, h is chosen to satisfy $(h, (p-1), (q-1)) = 1$, and d is computed so that $dh \equiv 1 \pmod{(p-1)(q-1)}$. The notation $(\alpha, \beta, \gamma) = 1$ implies that α, β, and γ have a greatest common divisor of 1 (i.e. they are relatively prime). Knowing p and q allows for the calculation of n, h, and d in that order. Knowing n and h though, as will be the case with a cryptanalyst, it is very difficult to factor n into its two primes p and q, so as to use p, q, and h to find d [1].

Substitution and transposition are two major methods of transformation of plaintext to ciphertext, and vice versa. Substitution consists of replacing all the plaintext characters by characters from another alphabet, as governed by a key, without changing the position of these characters. There are many variations of such ciphers based on the number of substitution alphabets used, the way in which these alphabets are generated, and the ratio of the number of plaintext characters, to the number of ciphertext characters per substitution. A simple example of a substitution cipher is shown in figure 2.1. In this case the key is the letter D, which implies a one-to-one correspondence between the characters of the English alphabet, and the characters of the English alphabet shifted, so as to start from the letter D.

Transposition on the other hand consists of rearranging the plaintext characters as governed by a key, without changing the identity of these characters. Again, there are various methods of transposition, a simple example of which is shown in figure 2.2. In

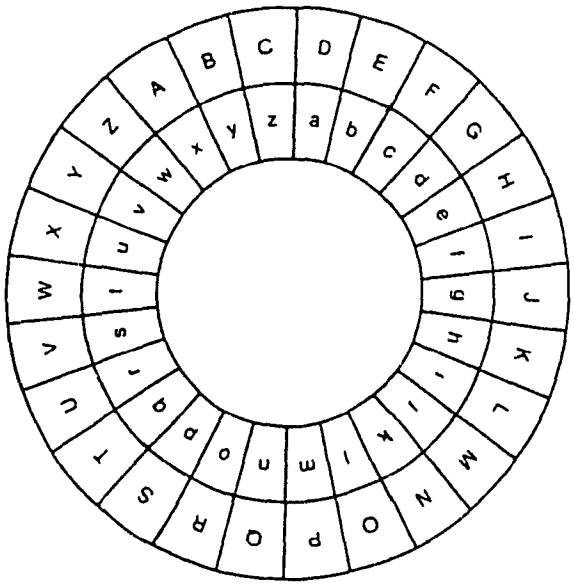

Figure 2.1 Simple substitution cipher [3].

A R I S T O T L E
1 6 3 7 8 5 9 4 2
M A J O R P R O D
U C T A N N O U N
C E M E N T F R I
D A Y A M

Figure 2.2 Keyed-columnar transposition cipher [4].

this case, the characters in the key are used to produce a mixed number pattern. This is done by numbering the characters, in accordance with their relative order of appearance in the standard alphabet, numbering repeating letters in sequence from left to right. The message is written as it is, row after row horizontally under the key, and then transcribed column by column according to the order of the mixed number pattern.

The above substitution and transposition examples involved letters of the English alphabet. It should be pointed out that the above concepts of substitution and

transposition can also be extended to include numeric equivalents of plaintext characters. These can be sequences of binary digits, or integers representing bit positions within such sequences. The above extension of substitution or transposition ciphers to include numbers is simply symbolic. This is still considered as substitution or transposition accordingly, and should not be confused with algebraic ciphers.

Algebraic ciphers involve the numeric equivalents of plaintext characters as mentioned above, manipulating these through algebraic operations. An example of this is the RSA cipher system already explained. An another example of this is the Vernam cipher. This cipher makes use of the exclusive-OR operation, to operate on the key and plaintext (represented as binary sequences) producing another binary sequence representing the ciphertext [4]. The idea of the exclusive-OR operation as an algebraic manipulation is used extensively in DES.

The final classification of data cipher systems, as block or stream ciphers, will follow in the next two sections.

2.1.1 Block Ciphers

A block cipher transforms a string of s plaintext characters (an input block), into a string of s ciphertext characters (an output block), as governed by a fixed-size key as shown in figure 2.3.

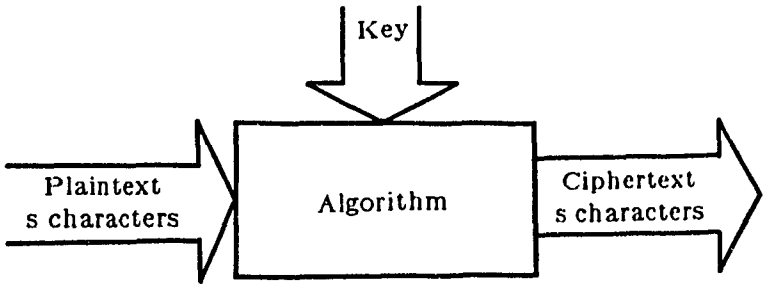

Figure 2.3 Block cipher [1].

The characteristics of the block cipher are such, that the key remains fixed relative to

the rate of incoming plaintext blocks. Furthermore, the enciphering transformation is such, that every character of the ciphertext block is jointly dependent on every character in the plaintext block, and every character in the key. This ensures diffusion and confusion respectively.

Confusion makes it difficult for any statistical analysis to indicate properties of the key. Furthermore, it forces the cryptanalyst to attempt to find the whole key simultaneously, as opposed to piece by piece, thus requiring to solve more complex equations [1].

Diffusion, on the other hand, attempts to spread the statistics of the plaintext over long portions of the ciphertext. This requires the cryptanalyst to intercept longer portions of ciphertext, before any attempts of statistical decipherment are made [1].

Confusion and diffusion ensure that the slightest error in the ciphertext obtained by the cryptanalyst, or in the possible deciphering key used by the cryptanalyst, will result in a totally different block of recovered plaintext, not at all close to the original plaintext block. This intense error propagation is another characteristic of a block cipher.

Finally, the cryptographic strength of a block cipher is directly proportional to the size of its plaintext block s, and the size of its key. Besides the obvious reason of increasing diffusion and confusion, the larger the size of the plaintext block, and key respectively, the more discouraged the cryptanalyst will be in constructing and maintaining a dictionary of plaintext-to-ciphertext block pairs. This can be constructed by using the enciphering key to map different plaintext blocks into ciphertext blocks, and then working backwards to match ciphertext blocks, obtaining the corresponding plaintext blocks. Obviously, this is possible only when the enciphering key is already known. This is the case with the public-key cipher systems described previously.

The most famous example of a block cipher is the Data Encryption Standard or DES algorithm. This was developed by IBM, and published first in 1975. After thorough examination it was considered secure enough, and was declared satisfactory as

a standard in 1977 by the United States National Bureau of Standards, at least for the next 10-15 years [1]. This is a conventional algebraic block cipher which takes a plaintext sequence of 64 bits, and a 64-bit key, and produces a 64-bit ciphertext sequence. The 64-bit key is made up of 8-bit words or bytes, with the last bit in each word used to maintain odd parity. These 8 bits numbered 8, 16, 24, 32, 40, 48, 56, and 64 are not used in the calculation, leaving the algorithm with a 56-bit key. The DES algorithm begins with an initial permutation of the plaintext block, and ends with the inverse of this initial permutation before producing the ciphertext block. These two permutations or transpositions, besides being inverses of one another, are key independent or fixed. Between these two inverse permutations is the major part of the DES algorithm, a product cipher, as shown in figure 2.4. By definition, this cipher is a transformation consisting of both transposition and substitution ciphers. This product cipher is accomplished through 16 identical rounds of computations. The input to the first round of computation is the plaintext block after being passed through the initial permutation. The input to every successive round is the output of the previous round. Each round makes use of one out of 16 48-bit keys, all of which are generated by first permuting the 56 non-parity key bits of the original key, using a permutation referred to as permuting choice 1, breaking the resulting 56-bit sequence into two blocks of 28 bits each, left shifting each block by a prescribed number of bits, putting the two blocks together, and permuting 48 of the 56 resulting bits using another permutation referred to as permuting choice 2. The shifting and permuting operations using permuting choice 2 are repeated 16 times, resulting in the 16 48-bit subkeys.

The computations performed in a particular round consist of splitting the input 64-bit word into two 32-bit left and right halves. The right half block is then passed through a selection operation which selects the bits in this block in a semi-ordered key independent fashion, in which 16 bits are used twice, expanding the 32-bit input block into a 48-bit output block. This is then modulo-2 added with the corresponding 48-bit

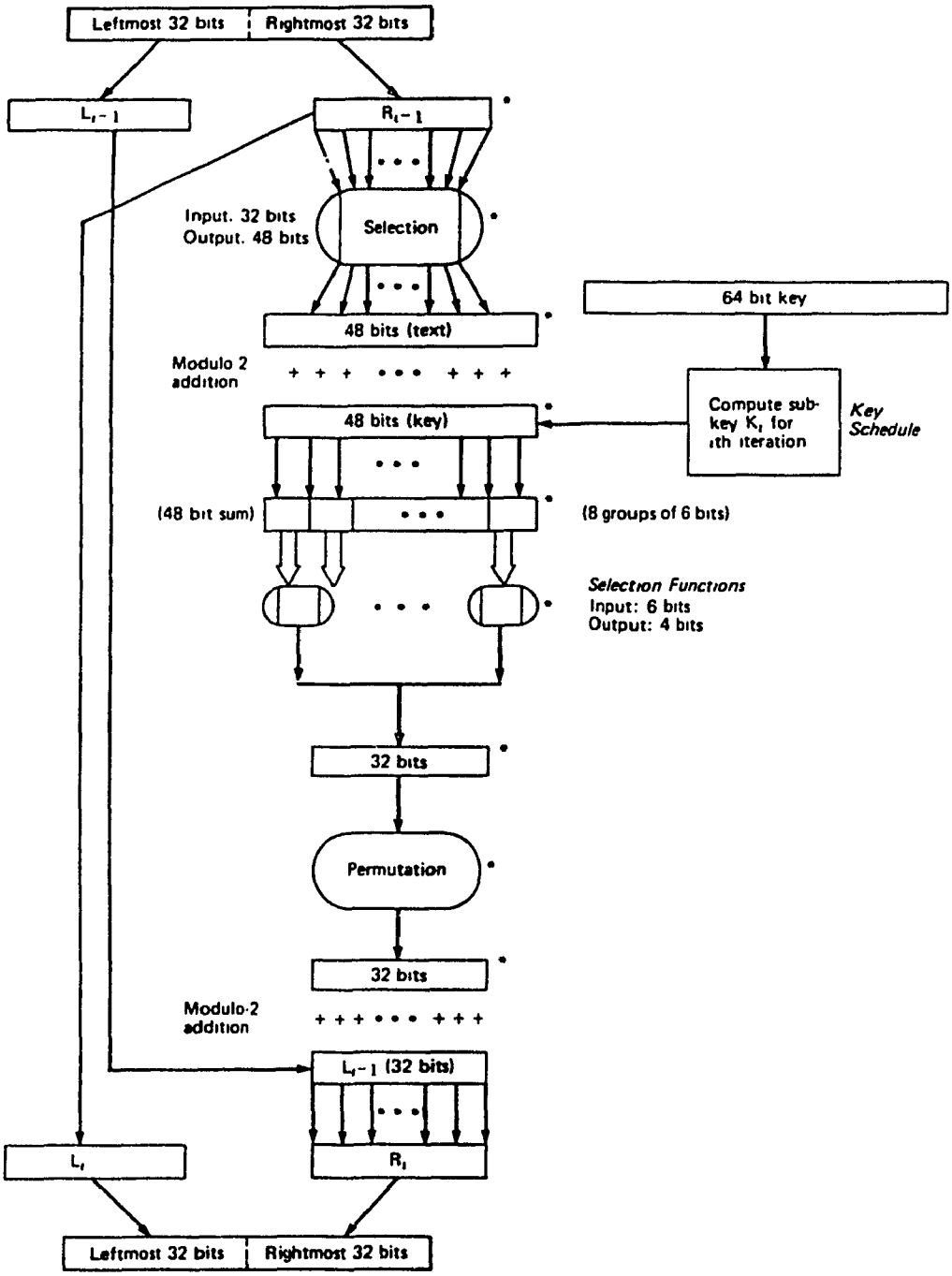

Figure 2.4 Product cipher within the DES algorithm [4].

subkey of that particular round. The result of the modulo-2 addition is a 48-bit block, which is then split into 8 6-bit blocks. Each respective 6-bit block is then used to select

an integer of 1 to 15, from one of eight corresponding selection functions or matrices. The first and last bits of the 6-bit block give reference to the rows of a particular matrix, while the middle four bits give reference to the columns. The resulting integer is then converted into a 4-bit string. The result of all eight parallel selection functions are then concatenated to produce a 32-bit block. This block is then permuted through another key independent permutation to produce another 32-bit block, which is modulo-2 added with the left half 32-bit block of the input 64-bit block of the particular round in question. The result is a 32-bit block, making up the right half of the output 64-bit bock of this round. The corresponding 32-bit left half of this output 64-bit block, is the original right half 32-bit block of the input 64-bit block of this round. After the final or sixteenth round, the left and right 32-bit halves of the final 64-bit output block are interchanged, before applying the inverse initial permutation [4,5,6].

Since its acceptance as a standard by the NBS, the DES has been the target of ample controversy. Some people have attempted to point out the major strengths and weaknesses of this algorithm, others have attempted to pursue its weaknesses in an attempt to break the algorithm in the shortest time possible, while still others accept the DES as it is, and attempt to find ways to use it as the basic building block in larger secure systems.

In the paper entitled "Long Key Variants of DES" by Thomas A. Berson [7], the existing key scheduling procedure of the DES algorithm is examined, and modifications are proposed for increased security. Berson first makes the point that many commercial implementations of DES precompute all 16 48-bit keys equal to 768 bits, in order to reduce the block processing time. This is because the key usually remains fixed for a certain amount of plaintext blocks. In this case, by precomputing all 48-bit keys in advance and storing them, all plaintext inputs after the first will be processed much faster. Otherwise, all the subkeys would have to be calculated, one per round, for every incoming plaintext block. Justifying the storing of all 16 48-bit subkeys (thus requiring

768 memory elements), Berson goes on to suggest bypassing the existing key scheduling algorithm, and directly applying a 768-bit key to operate the remaining DES algorithm. It is suggested that the 768-bit key be generated by a new key scheduling algorithm, which uses a longer key than 56 bits to produce a larger key space than 2^{56} keys, thus more strength against an exhaustive key search attack. Since a particular 56-bit key pattern produces a particular 768-bit key, the larger key space would result in more 768-bit keys, the upper bound obviously being 2^{768}. This upper bound implies the absence of a key scheduling algorithm, and the application of a user-chosen 768-bit key directly to the algorithm. The existing key scheduling algorithm, which uses a 56-bit key to produce a 768-bit key, would then be a special case of the proposed key scheduling algorithm. Another method would be to select a sub-sequence of 768 bits from a real or pseudo-random number generator.

Going back to the existing key scheduling algorithm, it is pointed out that among the 2^{56} possible keys, there are "weak" and "semi-weak" keys, which lack strength due to the fact that they expose a symmetry or other regularity, such as resulting in subkeys which take on only one o. two distinct values instead of 16. The author wants to point out that the smaller the initial key, which is to generate the 768-bit key, the more rounds will have to be performed within the key scheduling algorithm, increasing the chances of symmetry between subkeys. On the other hand, the larger the initial key the fewer key scheduling rounds will be performed, decreasing the chances of symmetry. Even with a larger key and fewer rounds of key scheduling, there can still be some "weak" and "semi-weak" keys, though these will diminish as the key gets longer. In fact, if a 768-bit key is applied directly with no key scheduling algorithm the person supplying the key can make sure that no symmetries exist within this long key. Of course, the author also realizes that the longer the key that will have to be supplied by the user, the harder it will be for this user to choose every bit of this key, making sure to avoid symmetries within the key, and repetitions relative to other such long keys. To remedy

this, the author is suggesting, at least conceptually, the introduction of a "weak key function" over the 768-bit keys, no matter how these are generated. When implemented, this function would decide on the relative strength or weakness of a particular key, which can then be compared with a predetermined threshold. If the key would be found adequately strong it would be used, otherwise, it would be deterministically modified until it would be declared adequately strong by the "weak key function." The ideas proposed by Berson will increase security against an exhaustive key search attack.

The second paper surveyed on the topic of DES is entitled "Drainage And The DES Summary," by Martin E. Hellman, and Justin M. Reyneri [8]. The object of this paper is to first study the behavior of drainage, for a random function in general, and then compare this with the behavior of drainage for the DES algorithm. A random function referred to here is a mapping from a finite set into itself, such that there is no obvious relationship between input and output. The output of this function appears to be chosen uniformly, independent from the input. This function can be represented by a directed graph, where each vertex is an element in the domain and range (since they are the same), and each edge represents the mapping between two particular elements. The random functions referred to here are such that their graphs consist of component graphs, each component of which has one cycle. The number of vertices making up respective component graphs, representing a random function, are referred to as the respective drainages into these components. Before calculating the drainage into the component graphs of a random function, the authors point out that this is not the only information these graphs can give about the particular functions. If the random function in question is the mapping from key to ciphertext of a block cipher with fixed plaintext, whose input key element is the previous ciphertext output, or similarly the mapping from one element in the key space to another element in the key space of this block cipher, then the exact structure of the component graphs representing this function is important. This is because the structure dictates the behavior or performance of a gen-

eralized cryptanalytical attack against the particular block cipher described above, as proposed by Hellman [9]. Although the exact structure of all the component graphs is not known, the general structure is known to consist of exactly one cycle with possibly a certain amount of "rivers" draining into it.

It was shown by Hellman that the proposed cryptanalytical attack for block ciphers, of the above described configuration, was able to break such ciphers if their component graphs consisted only of cycles, with no "rivers" draining into them. In this case the cipher would be broken in $\sqrt{N}$ operations, using $\sqrt{N}$ words of memory, where N is the number of elements in the key space. On the other hand, other structures of the component graphs would not allow the particular cipher to be compromised at all by Hellman's cryptanalytical method [9].

In this paper, the drainage of the largest component graph of a random function is examined, by first considering the case where more than 50% of the points of the entire graph drain into the same component. Supposing this case exists, the particular component graph would undoubtedly be the largest, since no other component graph could be larger with less than 50% of the points of the entire graph. The authors begin by giving an expression for the probability density of this drainage to be

$$P_N(i) = \left(\frac{1}{N}\right)^N \binom{N}{i} \left(\frac{i}{N}\right)^1 \left(\frac{N-i}{N}\right)^{(N-1)} C(i), \qquad (2.1)$$

where - N^N is the number of possible directed graphs of N points,

- $\binom{N}{i}$ is the number of possible component graphs of drainage i from a specific graph of N points,

- $\left(\frac{i}{N}\right)^1$ represents the mapping of the i points in the largest component graph into the same set of points,

- $\left(\frac{N-i}{N}\right)^{N-1}$ represents the mapping of all the remaining points into themselves,

and -C(i) is the probability that the component graph of drainage i is the only component in the entire graph of a random function.

Using Stirling's formula to approximately calculate large factorials and taking the asymptotic expression of C(i) to be $\sqrt{\frac{\pi}{i}}$ results in

$$P_N(i) = \frac{k}{i}\sqrt{\frac{N}{N-i}}, \tag{2.2}$$

where k is a constant dependent on the number of points in the entire graph of the random function. From this it is seen that the probability density remains roughly constant, as the drainage of the largest component graph of a random function goes from 50% to 90%. Finding the probability density of the drainage of the largest component graph, for the case where this accounts for less than 50% of the total points on the graph, and combining with the previous results, the expected value of the drainage of the largest component graph of a random function is 80%. This implies that theoretically 80% of all the points in the entire graph of a random function are within the largest component graph of this random function.

Having theoretical results as to the drainage of a random function, the authors chose the supposedly random function relating key to ciphertext for the DES algorithm, with fixed plaintext, to obtain corresponding experimental results. The drainage of the largest component graph of this function was calculated, to see how close it would be to that of a truly random function. In this case, the ciphertext output of each iteration was fed back as the next key element of the cipher, thus mapping the key space to itself as required in the specification of the random function previously. The points or vertices of the graph of this function thus consisted of the entire key space of the DES algorithm.

According to the DES algorithm, a particular vertex or key element cannot appear in more than one component graph if the plaintext is fixed. Hence, in this case the vertices of each component graph are mutually exclusive, from those of other component

graphs. When considering component graphs with different but fixed plaintext input, a particular vertex can appear in more than one component graph, but a particular mapping, or in other words a combination of two adjacent vertices cannot appear in more than one component graph.

The above rules particular to the DES algorithm, as well as certain general rules concerning the structure of a directed graph, allowed the authors to estimate the size of the largest component graph of the function, relating key to ciphertext of the DES algorithm, with fixed plaintext, in a relatively short amount of time, using a relatively few number of starting points. More specifically, the authors fixed the plaintext to the all zero value and then chose 100 different starting points, calculating the lengths of the cycles into which they drained and noting when different points drained into the same component graph. The resulting estimate was that 99% of all points or key elements within the entire key space of DES will drain into the same, hence the largest component graph. In other words, 99% of all keys in the DES key space will be connected as part of one component graph. This implies that each and every element out 99% of such key elements in the DES key space can possibly map at random into any other key element within the same component graph. To make the experiment more complete, the authors examined the drainage of the largest component graph of the same random function for different values of fixed plaintext. This was accomplished by taking a pair of starting points for every fixed plaintext, and following the same procedure as before. In total, 70 different plaintext inputs were examined. By counting how many times a pair of starting points drained into the same component graph, the authors were able to estimate the average drainage into the largest component graph over all plaintext inputs to be 80%. This implies that each and every element out of 80% of all such key elements in the DES key space will possibly map at random into any other key element within the same group, irrespective of the input plaintext. Furthermore, this percentage is equal to that of a truly random function, implying that no matter the actual fixed plaintext, the func-

tion from key to ciphertext of the DES algorithm in the specified configuration represents a truly random function.

The final paper to be examined on the topic of the DES algorithm is entitled "The Importance of 'Good' Key Scheduling Schemes (How to Make a Secure DES Scheme With $\leq$ 48-Bit Keys ?)," by Jean Jacques Quisquater, Yvo Desmedt, and Marc Davio [10]. Just like the first paper discussed, part of this paper examines an alternative key scheduling scheme for the DES algorithm. This scheme would render this algorithm either more secure against an exhaustive key search attack, without increasing the key size, or similarly maintain the same security against an exhaustive key search attack as the existing DES algorithm, using a shorter key. The paper then goes on to suggest other key scheduling schemes external to the DES algorithm, which can generate main keys, as opposed to subkeys, for this algorithm.

In an exhaustive key search attack the largest enemy of the cryptanalyst is the time needed to exhaust all possible keys (at worst case), or as many keys as necessary until the particular desired key is found. Thus, any factor to increase this time would increase the security against an exhaustive key search attack. For the DES algorithm, the time taken to find the desired key is at worst case proportional to

$$\frac{\max(t_{ks}, t_{ea}) \text{ x (number of keys)}}{\text{(number of used processors)}}, \tag{2.3}$$

where -t_{ks} is the time taken to execute the key scheduling scheme,

and -t_{ea} is the time taken to execute the encryption algorithm without any key scheduling.

In general $t_{ks} \ll t_{ea}$. Of the above factors, a designer of the DES algorithm can increase security against an exhaustive key search attack by increasing either t_{ks}, t_{ea}, or the size of the key space. Increasing the size of the key space for increased security against an exhaustive key search attack on the DES algorithm is a well known option, and the subject of many papers. Having already discussed this option in the first paper

presented here, it is time to examine the other options. Increasing t_{ea} is possible by increasing the complexity of the existing encryption algorithm, or replacing it altogether with another more complex such algorithm. The disadvantage of this is that it will also increase the processing time of DES, or in other words slow down the operation of the algorithm for the user. The option left is to increase the key scheduling time t_{ks}.

In general, the operation of the DES algorithm is such that many plaintext (ciphertext) blocks are encrypted (decrypted), using one particular key, or in other words, the input plaintext (ciphertext) rate to the DES (DES^{-1}) algorithm is much faster than the rate of change of the key. Operating the DES algorithm under these conditions requires the same subkeys for the entire time the key remains the same, hence making it more feasible to calculate these once when a particular key is introduced, and then storing them for subsequent use. In this respect, increasing t_{ks} will effectively not slow down the algorithm for the user, but will slow down the cryptanalyst considerably in his/her attempt of an exhaustive key search attack. In fact the authors suggest that t_{ks} is increased such that $t_{ks} >> t_{ea}$. This is made possible by using a key scheduling scheme involving 16 DES modules, all of which use a common key, with the output of one fed sequentially into the output of another as shown in figure 2.5.

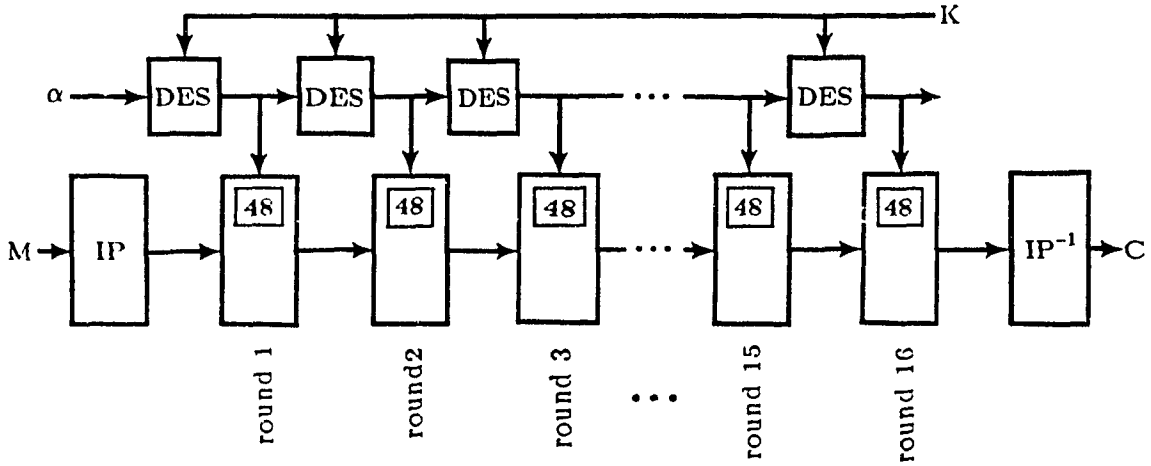

Figure 2.5 The DES algorithm 16 times stronger [10].

The input α to the first module is a publicly known 64-bit sequence. Furthermore, the output of each DES module constitutes a particular subkey. Using this key scheduling scheme, t_{ks} went from being negligible ($t_{ks} << t_{ea}$) to being approximately $16t_{ea}$. Considering the 16 48-bit output subkeys of this key scheduling scheme as one long 768-bit key, the cryptanalyst will at worst case still have to exhaust 2^{56} 768-bit keys, but the generation time of each of these long keys will now be 16 times longer than before. This is similar to keeping the generation time of each 768-bit key the same as before, and increasing the key space a factor of 16, from 2^{56} to 2^{60}. Either way the new scheme makes the existing algorithm 16 times stronger against an exhaustive key search attack. Furthermore, the new scheme can be extended by iterating a particular DES module 16 times before producing the next subkey. This will increase the security of the algorithm against an exhaustive key search attack by a factor of 256. Again, this can be looked upon as increasing the generation time of each of the 2^{56} 768-bit long keys, by a factor of 256, or similarly, keeping the generation of each of these keys the same, increasing the key space from 2^{56} to 2^{64}. Finally, the extended new scheme proposed can provide the same security as the existing DES algorithm, against an exhaustive key search attack, if the key length of the former is decreased by 8 bits relative to the key length of the latter. It should be pointed out that there are practical limits on the increase of t_{ks}, and the decrease of the key space. The increase of t_{ks} should be inversely proportional to how often the user will change keys. The more this happens the less t_{ks} is required to be, so as not to slow down the overall operation of the algorithm for the user. Furthermore, the key space cannot be decreased beyond a point which would allow the cryptanalyst to practically precalculate and store all subkeys for all possible keys

After proposing a key scheduling scheme to create the 16 48-bit subkeys, or one 768-bit long key for the DES algorithm, the authors examine key scheduling external to the DES. This would be a key scheduling scheme which would constantly provide main keys for the 64-bit key port of the DES algorithm. In this case the DES will not be

operated as usual, where the rate of the plaintext input is much greater than the rate of change of the key. Instead the plaintext input and key will change at the same rate. Two methods of continuously providing keys to the algorithm is as shown in figure 2.6.

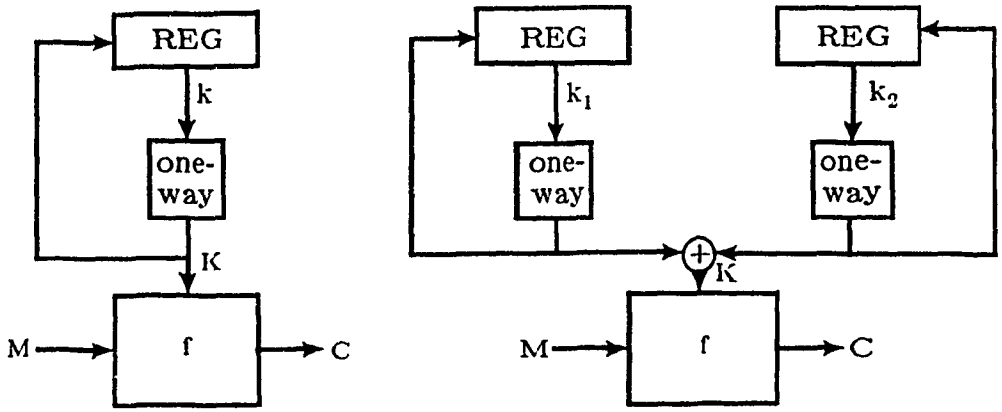

Figure 2.6 One-way key scheduling schemes in feedback [10].

In the first case (figure 2.6a), starting with a particular input key, the output of a one-way function is used as the generated key, as well as the input to the next iteration of this one-way function. The one-way function as its name implies, allows the derivation of the output from the input but not vice versa. The constantly changing key makes it hard for the cryptanalyst to obtain an instance of this key. Assuming this happens though, the cryptanalyst will be able to generate the future keys to be used, in order, but not the previous keys already used. Thus, having one key allows one to decrypt the present and subsequent plaintext, but not any previous plaintext. This can lead to serious problems, more so the sooner the cryptanalyst obtains an instance of the key K.

In the second case (figure 2.6b) two one-way functions in the same configuration as before are used to operate in parallel. Starting with two distinct input keys the respective output of each is fed as input to itself. The generated key is then the exclusive-OR of the output pair at each iteration. The key is constantly changing as before, making it

hard for the cryptanalyst to obtain an instance of this key. Assuming this happens though, the cryptanalyst will not be able to generate the previous keys already used, as before, but also particular in this case he/she will not be able to generate the future keys to be used. Thus, having one instance of the key K, allows one to decrypt only the plaintext encrypted by this key.

One possible scheme for the one-way function used in the above configuration is the DES algorithm, with fixed publicly known plaintext, and ciphertext feedback into the key port. By studying the first paper presented here, it is known that no matter the actual fixed plaintext, the function from key to ciphertext of the DES algorithm in this configuration represents a random function, which maps 80% of its key space into the same space. This implies that on average 80% of the entire key space can possibly be generated through iterative applications of DES. Although the exact structure is not known, the general structure of the graph representing this mapping will consist of one cycle, and possibly one or more "rivers" draining into it. It is pointed out here that the scheme of figure 2.6a is only secure while the generated keys are part of a "river", within the structure of the graph, and not secure while within the cycle. This is because by allowing a cryptanalyst to generate future keys from one instance of a key K derived, the cyclic structure will allow this cryptanalyst to even generate previous keys by moving forward along this cycle, thus overriding this "one-way" principle. On the other hand, the security of the scheme of figure 2.6b does not depend on the structure of the graph representing the random mapping of 80% of the key space into itself. This advantage is provided by the exclusive-OR in this scheme. Only if an instance of a key K at the output of the exclusive-OR is derived, and broken into its component parts (entering into the exclusive-OR), can the security of the scheme of figure 2.6b be reduced to that of figure 2.6a. The latter is next to impossible, since for every derived key K of length n at the output of the exclusive-OR there are 2^n possible pairs of component keys, which can lead to this key.

The schemes of figure 2.6 are block ciphers which incorporate certain principles characteristic of a stream cipher. In particular, this refers to the key generation scheme. As in stream ciphers, the key is constantly changing. In the schemes of figure 2.6 a new block of key is applied to every incoming block of plaintext. The main difference between this and an actual stream cipher is that in the stream cipher the key will not be generated continuously in blocks, but rather continuously character by character, and applied as such to the incoming plaintext.

2.1.2 Stream Ciphers

A stream cipher, as shown in figure 2.7, consists of encrypting a stream of plaintext characters, by combining or mixing these on a one-to-one basis with a relatively long or infinite sequence of similar characters.

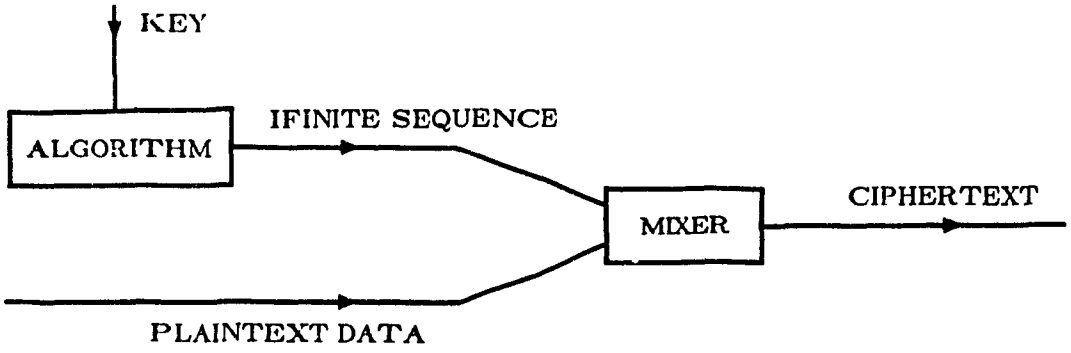

Figure 2.7 Stream cipher [3].

These characters are usually bits, and the mixing operation is usually an exclusive-OR. The long generated sequence or keystream is the result of an algorithm, referred to as the keystream generator, whose input is a relatively short key referred to as the seed key.

The difference between stream ciphers and block ciphers lies mainly in the manipulation of the key. In block ciphers the key interacts directly with the mixing operation, and remains relatively constant with changing plaintext. Assuming a plaintext attack,

all emphasis is placed on finding this key. On the other hand, in the case of stream ciphers the keystream is constantly changing, as is the incoming plaintext. Assuming a known plaintext attack, the mixing operation is readily reversed, and a portion of the keystream equivalent to the length of the known plaintext is revealed. If this portion is less than one period of the keystream (which is usually the case), then information from this portion can be used in an attempt to either predict statistically the remaining portion of this keystream (completing one period), or to somehow find the exact or equivalent generator configuration, and seed key, that would generate this keystream. Thus, from a cryptographer's point of view a keystream generator should satisfy the three necessary requirements, without which it cannot be considered secure against cryptanalysis. The sequence produced from the keystream generator

1) must have a guaranteed minimum length for its period,

2) must appear random,

3) must have a large linear equivalence or complexity [1].

The most commonly used type of keystream generator, or at least one that is commonly used as a basic building block of other keystream generators, is the shift register. An n-stage shift register consists of n binary storage elements connected in series, and identified as $S_0, S_1, \ldots, S_{n-1}$ such that $S_{i+1}(t) = S_i(t+1)$. The contents of the register at any one time identify a state. A register will change states until a particular previous state is encountered, after which the states will enter a cycle. Correspondingly, for every state there will be an output from the last stage of the register, which after some time will also cycle, thus producing a cyclic or periodic sequence of bits.

The number of states through which a shift register will cycle, or equivalently the period of the sequence it produces all depend on the actual feedback function, as well as the initial state of the register. Keeping a constant feedback function, a particular initial state will result in a particular cycle. Another initial state from within the same cycle will result in a translated version of this cycle. On the other hand, another initial

state outside this cycle will result in another cycle none of whose states can be found in the former cycle. In all, there are 2^n possible states which are distributed among mutually exclusive and collectively exhaustive cycles of various lengths. The smallest possible cycle length is 1 and the largest possible cycle length is $2^n - 1$. The number of cycles, and corresponding lengths of these cycles will depend on the particular function employed. If this is nonlinear (ie. makes use of the multiplication operation) then this dependence is unknown. On the other hand, if the feedback function employed is linear (ie. makes use of the exclusive-OR operation) then this function will convey a lot of information about the sequences generated by the particular shift register. The feedback function in this case is such that

$$f(s_0(t), s_1(t), \ldots, s_{n-1}(t)) = \sum_{i=0}^{n-1} c_i s_i(t) \pmod 2, \tag{2.4}$$

with each c_i, a feedback coefficient, being equal to 1 or 0. The feedback coefficients c_0, c_1, ..., c_{n-1} will identify the particular configuration of an n-stage linear feedback shift register. Furthermore, they will act as the coefficients of the characteristic polynomial

$$f(x) = 1 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1} + x^n \tag{2.5}$$

representing this register. If the characteristic polynomial f(x) of a particular n-stage linear feedback shift register is such that f(x) divides $x^e + 1$, but does not divide $x^r + 1$ for any $r < e$, then f(x) is said to have exponent e where $e \leq 2^n - 1$. If for a particular characteristic polynomial the exponent e is $2^n - 1$, then this is a primitive polynomial. Such a polynomial is known to produce one cycle of all $2^n - 1$ possible states (excluding the all-zero state). Obviously, this is the largest possible cycle of states that can be generated by a particular n-stage shift register. Furthermore, the sequence produced from such a shift register configuration is thus also of length $2^n - 1$, and is referred to as a maximal length sequence. The statistical properties of such sequences are very well known. Furthermore, all primitive polynomials leading to maximal length sequences are known and tabulated. Thus, to satisfy the first of the three necessary requirements as

to the length of a keystream, it is appropriate to use a keystream generator containing a linear feedback shift register, represented by a primitive polynomial. In this case, not only is the period guaranteed to be 2^n -1, but this value is also the largest possible, a favorable factor for the impediment of cryptanalysis.

The second requirement for the security of a keystream generator, concerning the randomness of the keystream, is important in fighting against statistical attacks on this keystream. A statistical attack can consist of obtaining a large amount of keystream bits, finding the distribution that best fits these bits, and then using this distribution to predict probabilistically the keystream bits to follow. Randomness in the keystream would not allow for such a distribution to be obtained, hence obstructing a statistical attack. Furthermore, randomness can be built into the keystream as it is being generated. An example of this is the generation of keystreams with noiselike characteristics, as will be shown later in the next paper surveyed. In this case an equivalence is implied between the randomness of the generated keystream, and the randomness of noise. Three necessary requirements as proposed by Golomb for declaring randomness within one period p of a keystream are

1) If p is even then the generated keystream of length p shall contain an equal number of zeros and ones. If p is odd then the number of zeros shall be one more or one less than the number of ones [3]

2) In the generated keystream of length p, half the runs have length 1, a quarter have length 2, an eighth have length 3, and in general, for each i for which there are at least 2^{i+1} runs, $\frac{1}{2^i}$ of the runs have length i. Moreover, for each of these lengths which are greater than one, there are equally many gaps (runs of zeros) and blocks (runs of ones) [3].

3) The out-of-phase autocorrelation is constant. [3]

The final requirement for the security of a keystream generator is that the sequence it produces has a large linear equivalence. In general, a particular sequence can be generated by either a linear or a nonlinear generator. Linear equivalence refers to the smallest size shift register, or combined shift registers, required to generate a particular sequence using a linear generator. This is definitely greater than the size of the shift register required to generate the same sequence using a nonlinear generator.

Having (a complete period, or a portion of a period of) a sequence of linear equivalence α, the objective of the cryptanalyst is to obtain the shift register configuration and initial load required, to generate this sequence. Despite the exact generator (linear or nonlinear) used by the cryptographer to generate this sequence, the cryptanalyst is only capable of systematically finding an equivalent linear generator which can generate this sequence.

No matter the linear equivalence of a sequence, if a complete period of this sequence is known, then this sequence can be generated by an ordinary recirculating shift register having an initial load equal to some phase of the sequence. This particular sequence can be represented by the ratio of a polynomial representing the initial load of the recirculating shift register, to the characteristic polynomial of this generator. More specifically, if there is no greatest common divisor (gcd) between numerator and denominator polynomials, then the linear equivalence of this sequence is maximum. This implies that no shift register, or combined shift registers, of shorter length than the recirculating shift register can be linearly configured to generate the same sequence. On the other hand, if there exist any gcd's between numerator and denominator, and these are cancelled, then the resulting numerator and denominator polynomials represent the initial load and configuration respectively, of a shorter linear feedback shift register than the recirculating shift register, which can generate the exact sequence in question. In this case, the length of the resulting shift registers, or combined shift registers, will depict the linear

equivalence of this sequence.

If a complete period of a sequence with linear equivalence α is not known, then an equivalent linear generator which can generate this sequence can only be found, if at least 2α - 1 consecutive elements of this sequence are known. The 2α - 1 elements define α independent vectors (IV) of length α, representing α sequential states of the shift register. These can be used to obtain all other states or vectors contained in the sequence. This can be done either by solving α simultaneous equations of α unknowns, or equivalently inverting an (α x α) matrix.

Related to linear equivalence is the complexity of a sequence. Complexity is the ratio of linear equivalence to sequence length, or in other words, the number of independent vectors to the total number of vectors contained in a sequence.

The following paper to be surveyed is entitled "Generation of Binary Sequences with Controllable Complexity" by Edward J. Groth [11]. The first part of this paper shows how a linearly generated maximal length sequence of minimum complexity can be made more complex by the introduction of nonlinear logic. Keeping the length of the sequence constant, the particular configuration of the nonlinear logic will control the linear equivalence of this sequence precisely, allowing it to go from minimum complexity to maximum complexity. The second part of this paper shows how further constraints on the configuration of the nonlinear logic will impose noiselike characteristics on the generated sequence, besides its maximal length and maximal complexity.

The first point made by Groth is that using a shift register in a linear configuration, and using the same shift register in a nonlinear configuration, the latter will result in a sequence of higher linear equivalence or IV count. Furthermore, if the length of both sequences is the same, then the one generated by the nonlinear configuration will have a higher complexity. Two possible locations for the nonlinear logic in a shift register sequence generator are the feedback path, and the feedforward path. Although introducing nonlinear logic in the feedback path does increase the linear

equivalence of a sequence, there is no known relationship between the actual nonlinear configuration, and the precise linear equivalence. Furthermore, nonlinear logic in the feedback path of a shift register affects the length of the sequence in an unknown way. Thus, a particular pre-determined sequence length cannot be obtained with a particular nonlinear logic configuration, thus violating the first requirement for a secure keystream generator. As a result of the linear equivalence, and sequence length of a sequence being generated by a generator with nonlinear logic in the feedback path not being controllable, the complexity of such a sequence is also not controllable.

It is thus suggested by Groth that the logic in the feedback path of a shift register remain linear, and that nonlinear logic be introduced in the feed-forward path. Furthermore, the logic in the feedback path will be such, so as to guarantee a maximal length sequence feeding through the shift register. This will ensure that the shift register will go through the maximum number of states, which will in turn guarantee a sequence of maximum length at the output of the generator. An example of this is shown in figure 2.8. Here the shift register consists of 6 stages. The feedback configuration is such, so as to produce a maximal length sequence of 2^6 -1 or 63 at the output of this register, and hence at the final output of this generator. Furthermore, the IV count of such a linear feedback shift register is 6. By incorporating the nonlinear logic in the feed-forward path, the IV count at the output of the generator is raised to 21. This is directly related to the 21 stages of the combined shift registers in the linear generator, needed to produce the same sequence. Thus, although the sequence length remains the same at 63, the IV count is raised from 6 to 21, thus raising the complexity from 0.095 to 0.333, by the use of the nonlinear logic in the 6-stage shift register.

Groth showed that in general (of which figure 2.8 is a special case), a nonlinear feed-forward generator with shift register of length r, and one nonlinear stage, under the following constraints, increases its IV count from r to $\frac{r(r+1)}{2}$.

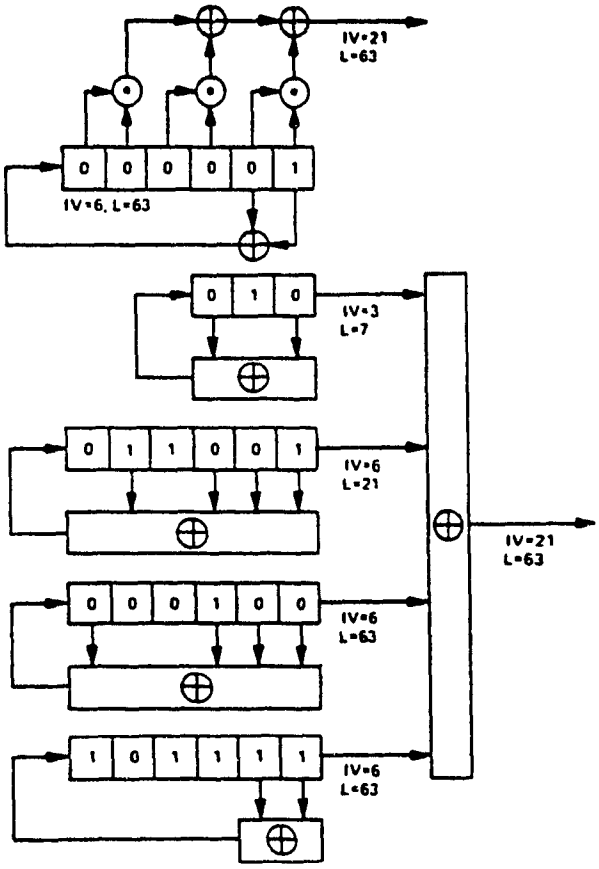

Figure 2.8 Linear generator with feed-forward nonlinear logic and its linear equivalent [11].

Constraints:

1) The feedback generator will produce a maximum length sequence.

2) The multipliers fed from the shift register stages of the feedback generator will have only two inputs each.

3) The span of any multiplier's input connections will not exceed the length of the shift register of the feedback generator.

4) Without loss of generality, the initial load of the feedback generator will be 000 ... 001.

It is suggested that these results can be further extended. Cascading additional non-

linear stages to the original feedback shift register, will raise the IV count of the generated sequence to a maximum value, this being the length of the generated sequence. In turn, the corresponding complexity will be 1 (or maximum). Groth showed that this can be achieved with r-1 layers of nonlinear logic, where r is the length of the shift register, if the following constraint is respected.

Constraint:

All multipliers in all layers use one particular span type.

In this case, the IV count will rise from r in the 0-th layer (consisting only of the linear feedback shift register) to $2^r - 1$ in the (r-1)-th layer. For every shift register of length r, the IV count of a particular layer q is related to the index of this layer by

$$IV(r,q) = \sum_{i=1}^{q+1} p_{r,i} \tag{2.6}$$

where $p_{r,i}$ is an element in the r-th row and i-th column of Pascal's triangle (figure 2.9), in which these rows and columns are indexed respectively, beginning with 0.

1										
1	1									
1	2	1								
1	3	3	1							
1	4	6	4	1						
1	5	10	10	5	1					
1	6	15	20	15	6	1				
1	7	21	35	35	21	7	1			
1	8	28	56	70	56	28	8	1		
1	9	36	84	126	126	84	36	9	1	
1	10	45	120	210	252	210	120	45	10	1
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.

Figure 2.9 Pascal's triangle [11].

Groth found that the IV count can be made to grow most rapidly by replacing the last stated constraint with the following constraint.

Constraint:

Place a series of multipiers having spans 1, 2, 4, 8, etc. in the separate layers irrespective of the order, and irrespective of the presence of other multipliers.

In this case, the IV count will rise from r in the 0-th layer (consisting only of the linear feedback shift register) to $2^r - 1$ in the $\left(\left\lceil \log_2 r \right\rceil\right)$-th layer. For every shift register of length r, the IV count of a particular layer q is related to the index of this layer by

$$IV(r,q) = \sum_{i=1}^{2^q} p_{r,i} \tag{2.7}$$

where $p_{r,i}$ is the element in the r-th row and i-th column of Pascal's triangle as mentioned before.

Thus far it has been shown that a shift register configured in a particular way can act as an excellent keystream generator. This configuration consists of linear feedback leading to a sequence of maximal length, and nonlinear feed-forward logic leading to a sequence of maximum IV count and complexity, using a minimum number of nonlinear logic layers. Besides increasing the complexity of the generated sequence, the nonlinear logic in the feed-forward path can be configured so as to additionally impose noiselike characteristics on the generated sequence. Noiselike characteristics may be taken as the running rms unbalance over a certain window length of the sequence, and the distribution of lengths of runs of 1 and 0 in a running window. It has been found that noiselike characteristics can be achieved if, in addition to the first four constraints given earlier, the following constraints are also imposed.

Constraint:

1) All the multipliers in each layer must have different input spans.

2) There should be as many multipliers in a given layer as an r-stage shift register will support with one tap per stage.

Groth points out that arranging the multiplier connections to satisfy the above constraints is equivalent to a generalization of the Langford problem, which is to arrange the numbers 1, 1, 2, 2, 3, 3, ..., g, g, in a sequence (without gaps), in such a way that for h = 1, 2, 3, ..., g, the two h are separated by exactly h places; for example 41312432 (g=4) or 6751814657342832 (g=8). Groth suggests that g in this case is the number of multipliers and 2g the register length. Furthermore, the numbers in the Langford arrangement are assigned sequentially to register stages, so that two inputs of a multiplier are connected to the two stages with the same number. An algorithm is provided for finding all Langford arrangements of a certain length. Applying this algorithm, the number of possible Langford arrangements representing multiplier configurations, within one nonlinear logic layer of register length r, leading to noiselike characteristics are as shown in table 2.1.

Table 2.1 Number of Langford arrangements according to Groth.

Register Length r	Multipliers g	Langford Arrangements per Layer
2	1	0
4	2	0
6	3	1
8	4	1
10	5	0
12	6	0
14	7	26
16	8	150
18	9	0
20	10	0
22	11	17 792
24	12	108 144

The above Langford arrangements are available for each register, in each nonlinear logic layer of a particular multilayer nonlinear feed-forward generator.

Groth suggests that the number of maximum-length sequences of maximum complexity that can be generated by a particular generator of length r, is equal to at least the product of the number of all maximal length sequences of length $2^r - 1$, times the number of layers, times the number of Langford arrangements available for each layer.

Here Groth should be more specific. The product of the last two factors does not represent all nonlinear logic configurations available from a specific generator, as it is supposed to. This value is better quantified by the number corresponding to the product of all Langford arrangements available for each layer. Besides the above miscalculation, two other problems were encountered in the ideas put forth by Groth in his attempt to generate sequences with noiselike characteristics. The first problem is a discrepancy as to what is the exact definition of the span of a multiplier In the previous analysis dealing with the relation of the complexity to the configuration of a particular generator, the span of a particular multiplier referred to the distance between the two stages in which the multiplier was connected, as one counts from one stage to the other. The span following this convention comes out one number more, as compared to the convention adopted when introducing the Langford problem. Under the latter convention, the span of a multiplier is the actual distance between the two stages on which the multiplier is connected, excluding these stages.

The second problem encountered in Groth's work is in the exact comparison of a Langford arrangement to the configuration of the multipiers in a nonlinear logic layer of a generator. In the Langford problem there are two instances of g distinct but consecutive numbers starting from 1, to be arranged accordingly. The last two constraints outlined for obtaining noiselike characteristics do not dictate that the spans of the multiplier, in a particular layer, have to be consecutive from 1 to g, where g is the number of multipliers, and one half the length of the register in that layer. In fact this is only a subset of the spans that can be used. In general the spans of the multipliers can be any $\frac{r}{2}$ numbers within the range 1 ... r-1. These can be arranged as in the Langford problem. This will result in many more arrangements than before, of the multipliers in a particular layer, without violating the constraints for noiselike characteristics Since not all types of multiplier spans are required to appear in a particular nonlinear logic layer, using the scheme just described, caution has to be maintained in making sure that the

constraint leading to a generated sequence of maximum complexity within a minimum number of layers is satisfied. This is carried out by making sure that at least one layer contains a multiplier of span 1, at least one other layer contains a multiplier of span 2, at least one other layer contains a multiplier of span 4, etc. The possible arrangements for a particular layer will thus depend on the existing arrangements of the multipliers in the other layers of the same generator. The number of arrangements, representing multiplier configurations for the separate layers of a particular multilayer nonlinear feedforward generator of length r, are experimentally found to be as shown in table 2.2.

Table 2.2 Actual number of Langford arrangements.

r	Guaranteed one Multiplier of Span	Arrangements per Layer	Total Nonlinear Logic Configurations
2	1	1	1
4	1	1	
	2	0	0
6	1	2	
	2	2	
	4	2	8
8	1	13	
	2	12	
	4	11	1 716
10	1	64	
	2	58	
	4	52	
	8	23	4 439 552
12	1	409	
	2	374	
	4	354	
	8	240	1.2996×10^{10}
14	1	3 428	
	2	3 374	
	4	3 256	
	8	2 531	9.5315×10^{13}

Besides the few problems arising in this paper the basic ideas and results behind Groth's work are very important. They allow for the generation of secure keystreams, having maximum length, minimum complexity, and noiselike characteristics.

2.2 SPEECH SECURE SYSTEMS IN GENERAL

The basic ideas of cryptography can be applied directly or indirectly to speech security. Speech security is made possible either through analog secure systems or digital secure systems. Digital speech encryptors as well as analog speech encryptors operate on digitized speech, which without the inherent properties of speech is equivalent to data. As a result, the techniques underlying these systems are greatly influenced by data cipher systems. Furthermore, all speech secure systems will require some element of randomness in their operations. Generating pseudorandom numbers is a major operation behind stream cipher systems. Hence, these can offer insight into the development of all speech secure systems.

The distinction between digital and analog speech secure systems is based on the form of the signal being transmitted. In digital speech secure systems the incoming signal is both encrypted and transmitted in digital form, as shown in figure 2.10.

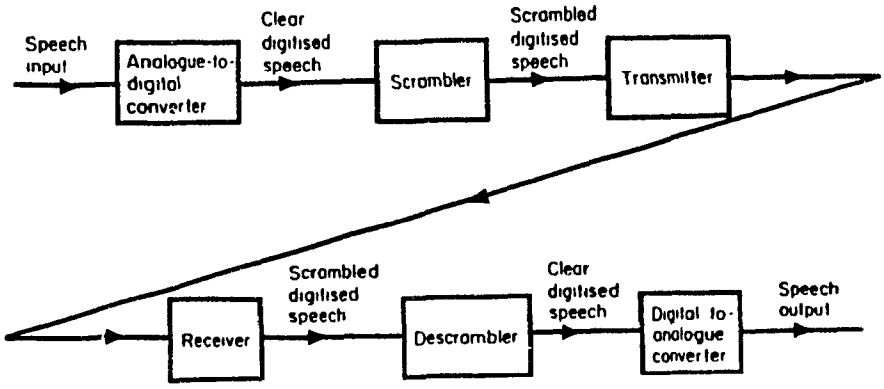

Figure 2.10 Digital secure system [1].

On the other hand, analog speech secure systems can be classified further into two categories; analog speech encryption systems and analog speech scrambling systems. In analog speech encryption systems the incoming signal is encrypted in digital form but transmitted in analog form as shown in figure 2.11. In analog speech scrambling systems

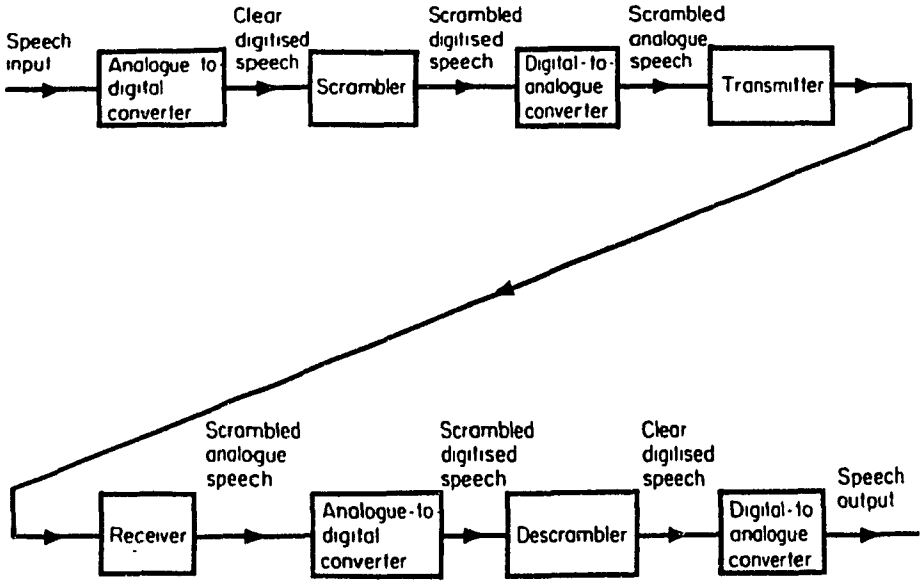

Figure 2.11 Analog speech encryption system [1].

on the other hand, the incoming signal is both scrambled and transmitted in analog form, as shown in figure 2.12.

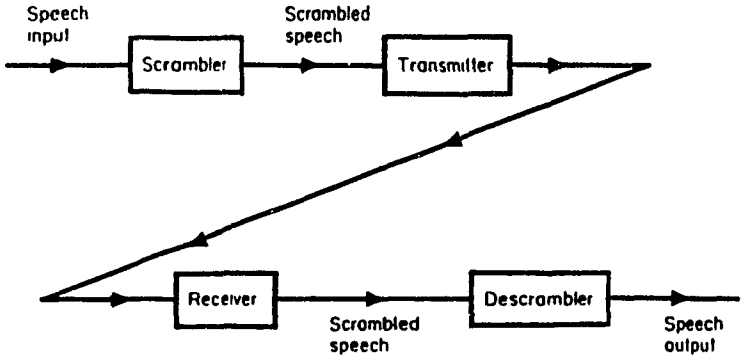

Figure 2.12 Analog speech scrambling system [1].

2.2.1 Digital vs. Analog Secure Systems

Technically speaking, digital speech encryption techniques have been known to achieve the highest degrees of security, over all other speech encryption techniques. The implication then is that digital secure systems should be used over analog secure systems. This is not the case at present. The reason for this is that digital transmission is still not quite compatible with today's technical environment, the major part of which is geared towards analog transmission. This is evident in the major classes of analog channels used for speech communication, including telephone and radio channels that are currently standard. Consequently, there is justification, at least for now, in examining secure systems which make use of analog transmission, concentrating on the improvement of their security level.

2.2.2 Analog Speech Encryption Systems

As a result of the reasons mentioned in the previous section, there will be no further mention of digital secure systems. The topic of discussion from this point on will be analog speech secure systems. As mentioned previously, there are two classifications of analog speech secure systems; those performing analog speech scrambling, and those performing analog speech encryption, of which, both employ analog transmission. An analog speech encryption system thus allows for compatibility with today's technical environment, while at the same time taking advantage of the high security level inherent in digital speech encryption. This section will examine the degradation that analog speech encryption systems impose on speech, as compared to analog speech scrambling systems in general.

Analog speech encryption systems, and analog speech scrambling systems in general, can be compared on the basis of security level upon encryption or scrambling, as well as voice degradation upon decryption or descrambling, respectively. In general, the

two factors of security and voice quality are always in conflict, or in other words, inversely related. Maintaining a constant level of security for both types of systems at the transmitter, these can then simply be compared solely on the basis of speech degradation at the receiver. This procedure is carried out in the paper entitled "Perfect Secrecy Encryption of Analog Signals" by Allen Gersho [12]. The objective of this paper is to show that irrespective of the analog encryption or scrambling scheme used, if perfect secrecy is attained, using a finite-size digital key, then the quality of the recovered message must inevitably be degraded. Furthermore, that this degradation is lower bounded by the degradation achieved by analog speech encryption.

According to Gersho, analog speech encryption is simply a special subclass of analog speech scrambling in general, since considering its overall operation it also manipulates an analog signal into a secure analog signal ready for transmission, irrespective of what this manipulation may be. He points out that for both these systems, the tradeoff between security level and voice quality is controlled by the key size of the system.

A general analog speech secure system can be modeled as shown in figure 2.13.

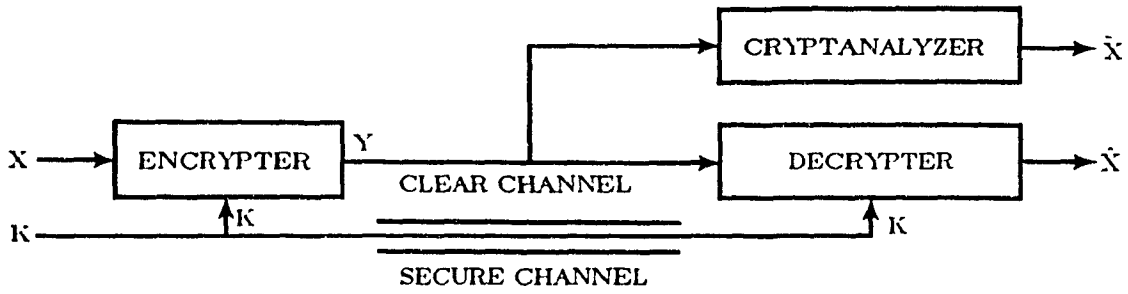

Figure 2.13 Model of an analog speech secure system [12].

Here $\mathbf{X}$ is a random vector denoting a finite segment of a discrete-time analog amplitude random process. There are M components to this vector having a joint probability density function denoted by $p_{\mathbf{X}}(x_1, x_2, ..., x_M)$. The encrypter transforms $\mathbf{X}$ into $\mathbf{Y}$, a similar random vector with a possibly different number of components. The transformation from $\mathbf{X}$ to $\mathbf{Y}$ is one of many, as specified by the key $\mathbf{K}$, where $\mathbf{K}$ is itself a random vari-

able that is uniformly distributed on the set 1, 2, 3, ..., N, where N is the key size. To maintain generality, the encryption in figure 2.13 is assumed to be random, thus represented by

$$\mathbf{Y} = f(\mathbf{X}, \mathbf{K}, w), \tag{2.8}$$

where w indicates a particular point in the sample space to account for the possibly random aspect of the encryption. This implies that given $\mathbf{X}$ and $\mathbf{K}$ there exist various values of $\mathbf{Y}$ as specified by a conditional distribution. Similarly, the decryption operation by both the intended receiver and the cryptanalyst can be represented by

$$\hat{\mathbf{X}} = g(\mathbf{Y}, \mathbf{K}), \tag{2.9}$$

and

$$\tilde{\mathbf{X}} = c(\mathbf{Y}) \tag{2.10}$$

respectively. Both $\hat{\mathbf{X}}$ and $\tilde{\mathbf{X}}$ will be degraded versions of $\mathbf{X}$, but these degradations will be minimal and maximal respectively. Degradation is defined by the function ρ where $\rho(\mathbf{x}, \mathbf{z})$ is the degradation between an original message $\mathbf{x}$, and its degraded version $\mathbf{z}$. The value of degradation ρ is a nonnegative number, which increases from 0 (i.e. $\mathbf{z} = \mathbf{x}$) with increased degradation. In the case of random vectors $\mathbf{X}$ and $\mathbf{Z}$ the degradation D is the average distance between $\mathbf{X}$ and $\mathbf{Z}$, such that

$$D = E\rho(\mathbf{X}, \mathbf{Z}) \tag{2.11}$$

An analog speech encryption system employing perfect secrecy can be modeled as shown in figure 2.14. In figure 2.14, $\mathbf{I}$ is a digital message which takes on integer values from the set $\{1, 2, 3, ..., N\}$. Using a key size N, as described previously, perfect secrecy is achieved by encrypting $\mathbf{I}$ using

$$\mathbf{J} = \mathbf{I} + \mathbf{K} \pmod{N} \tag{2.12}$$

to obtain $\mathbf{J}$, a uniformly distributed random variable over the set $\{1, 2, 3, ..., N\}$. Since $\mathbf{K}$ is by definition statistically independent of $\mathbf{I}$, according to the above encryption rule,

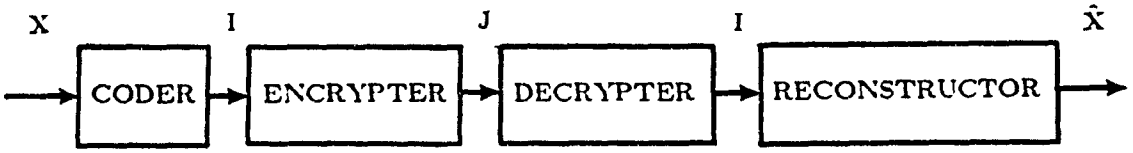

Figure 2.14 Model of an analog speech encryption system with perfect secrecy.

J is also statistically independent of **I**. Assuming an ideal channel, thus disregarding channel impairments and noise, **J** is transmitted to the receiver to be decrypted by the inverse transformation of the above, namely

$$\mathbf{I} = \mathbf{J} - \mathbf{K} \pmod{N}. \tag{2.13}$$

This is accomplished by the intended receiver, who, knowing **K** obtains **I**. It should be pointed out that the size of **I**, **J**, and **K** are all the same, namely N, put forth by Shannon as a requirement for achieving perfect secrecy. Shannon in his development of rate-distortion theory [13] also put forth a general method of describing the digitization of a message vector **X**, known as block source coding, and which is now known as vector quantization.

The coder (C) and reconstructor (R) of figure 2.14 represent the respective coder and reconstructor of an N-point quantizer. The operation of these is defined by

$$\mathbf{I} = C(\mathbf{X}) \tag{2.14}$$

and

$$\hat{\mathbf{X}} = R(\mathbf{I}) \tag{2.15}$$

respectively.

The performance of the general analog speech encryption scheme of figure 2.14 is thus dependent on the performance of the above quantization scheme. The latter is measured by the degradation between **X** and $\hat{\mathbf{X}}$, which is $E\rho(\mathbf{X},\hat{\mathbf{X}})$. Assuming an optimum quantizer, the degradation between **X** and $\hat{\mathbf{X}}$ for all possible N-point coder

reconstructor pairs $\{C, R\}$ is minimum and denoted by $D'_N(p_X)$.

Coming back to the generalized analog speech secure system of figure 2.13, Gersho points out that this can, without any modification, be rearranged as shown in figure 2.15

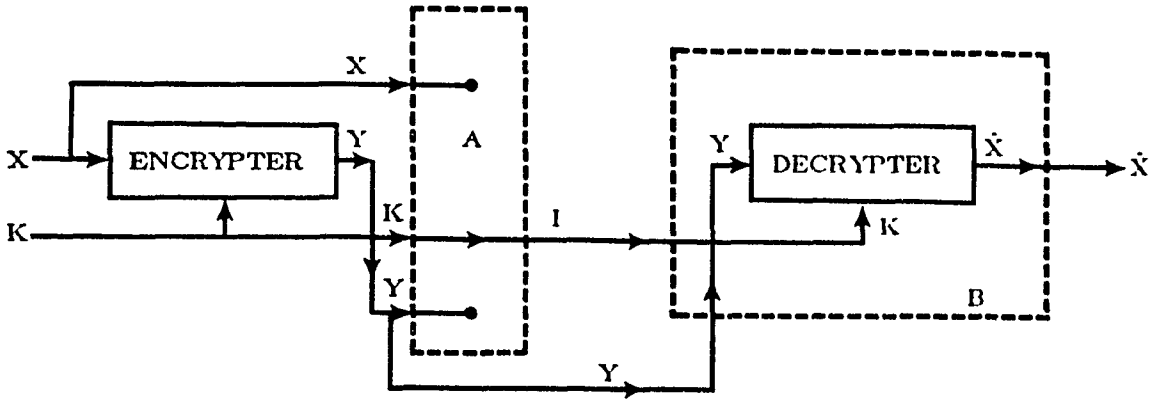

Figure 2.15 Rearranged analog speech secure system model [12].

to fit the model of figure 2.16.

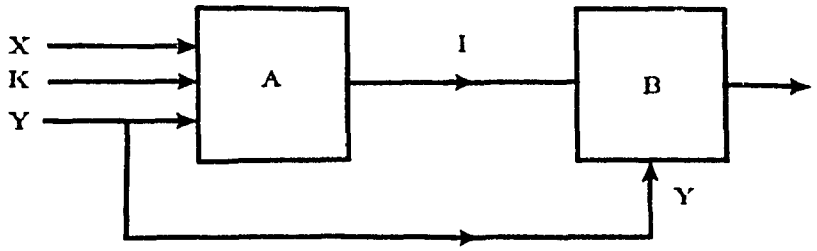

Figure 2.16 Equivalent model of analog speech secure system [12].

I in this case over a clear channel is equivalent to **K** over a secure channel. The model of figure 2.16 is similar to the model of figure 2.14 except for the additional side information between the coder and reconstructor. This implies that any analog speech secure system can be modeled as an analog speech encryption system with side information.

Maintaining perfect secrecy, **X** and **Y** are statistically independent. Hence, the side information **Y**, in this case, cannot provide any information for the reconstruction of **X**. As a result, the side information cannot improve the degradation beyond $D'_N(p_X)$, as provided by the analog speech encryption scheme of figure 2.14. $D'_N(p_X)$ is thus declared the minimum degradation. On the other hand, if the constraint of perfect

secrecy would have been violated, such that $\mathbf{Y}$ and $\mathbf{X}$ would be statistically dependent, then the side information would contribute to the reconstruction of $\mathbf{X}$, resulting in less degradation than $D'_N(p_X)$. Thus, under the constraints of perfect secrecy and the use of a finite-size digital key, $D'_N(p_X)$ is the minimum degradation possible, equal to the degradation obtained from equivalently digitizing the analog message before encryption, as is the case with analog speech encryption schemes.

2.2.3 Analog Speech Scrambling Systems

Analog speech scrambling methods are those operating directly on speech in analog form, without digitization, to produce a transformed waveform that is (ideally) unintelligible, but from which the original speech signal can be recovered by a suitable inverse transformation [14]. Analog speech scramblers can be further categorized as those which operate on a continuous-time continuous-amplitude speech signal, and those which operate on a sampled version of this signal, or a discrete-time continuous-amplitude speech signal. Analog scramblers of the former case are more specifically referred to as jig-saw puzzle scramblers, as a result of their separating the incoming analog waveform into components or pieces, much like in a jig-saw puzzle, and permuting these pieces to disguise the message content. The central factor here is that each piece preserves the integrity of some essential features of the original waveform [14]. Analog scramblers of the latter case are more specifically referred to as sample-based scramblers. These consist of sampling the input speech signal at the Nyquist rate, and then manipulating the amplitude and/or time ordering of these samples. Sample-based scrambling techniques usually lead to bandwidth expansion. Furthermore, they are extremely sensitive to channel degradations, and will usually require an adaptive equalization technique if the descrambled speech is to be of acceptable quality [14].

In general, all analog speech scrambling techniques which have a time-varying key require synchronization. They need to ensure that the part of the key operating on a

portion of the original speech signal in the transmitter, is the same part of the key operating on the corresponding portion of the scrambled speech signal in the receiver. This includes most scrambling techniques which have any cryptanalytical value. In addition, generally all sample-based speech scrambling techniques require synchronization, since in such cases a particular transformation is actually carried out on a group of speech samples as a whole. Thus, a slight synchronization error will lead to different groups of samples being descrambled, as compared to the corresponding groups of samples being scrambled. This makes transmission very sensitive to channel conditions. Besides attempting to improve the existing synchronization techniques, or attempting to provide new ones, the only other viable solution to the synchronization problem is attempting to eliminate the requirement of synchronization altogether. Two such techniques which have accomplished this are described in [15]-[18]. The general theory behind such techniques, of which the above are special cases, is outlined in detail in the paper entitled " A General Theory for Asynchronous Speech Encryption Techniques" by Lin-Shan Lee and Ger-Chih Chou [19]. In this paper, the general operation of an analog sample-based speech scrambler is first outlined, as shown in figure 2.17. It consists of sampling the incoming analog signal x(t) to obtain discrete samples x(n), which are then encrypted into discrete samples y(n), from which the encrypted analog signal y(t) is recovered. Every N discrete samples of x(n) denoted as {x(rN-N+1), x(rN-N+2), .., x(rN-1), x(rN)} form a frame, where N is the frame length, and r is the frame index. The N samples in a particular frame are identified by an N-sample window w(n), which is nonzero only for n=1, 2, 3, ..., N-1, and zero otherwise. The resulting windowed speech is then represented by an N-component vector $\bar{x}_r$, as

$$\bar{x}_r = \begin{bmatrix} x(rN-N+1)\, w(N-1) \\ x(rN-N+2)\, w(N-2) \\ \vdots \\ x(rN-1)\, w(1) \\ x(rN)\, w(0) \end{bmatrix} \triangleq \begin{bmatrix} x_r(0) \\ x_r(1) \\ \vdots \\ x_r(N-2) \\ x_r(N-1) \end{bmatrix}. \qquad (2.16)$$

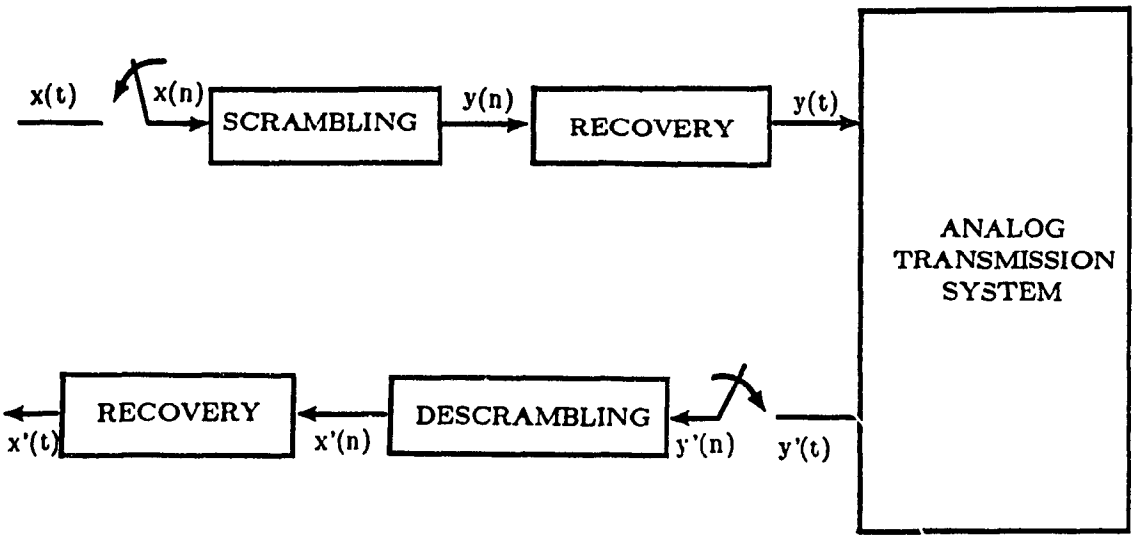

Figure 2.17 Sample-based speech scrambler [19].

Every frame of N samples is then scrambled with respect to a particular NxN transformation matrix T, to produce the corresponding frame of scrambled signals $\overline{y}_r$, where

$$\overline{y}_r = T\overline{x}_r = \begin{bmatrix} y_r(0) \\ y_r(1) \\ \vdots \\ y_r(N-2) \\ y_r(N-1) \end{bmatrix}. \tag{2.17}$$

From frame $\overline{y}_r$, the N samples of y(n) and hence the continuous-time scrambled signal y(t) are readily obtained. The descrambling is similar, except T is replaced by T^{-1}.

It is obvious that since the reciprocal transformations T and T^{-1} operate in frames of samples, the slightest synchronization problem will result in the input frame of T^{-1} being a slightly shifted version (i.e. different from) the corresponding output frame of T, leading to unsuccessful signal recovery.

The authors propose that the requirement of synchronization be eliminated from the system by removing the concept of scrambling on a frame to frame basis. This is

accomplished by increasing the size of a frame from N to N', where N' is much larger than N. The N' samples of a particular frame are identified by an N'-sample window w(n), similarly as before. The resulting windowed speech is then represented by an N'-component vector $\overline{x}_r$ as

$$\overline{x}_r = \begin{bmatrix} x(rN-N'+1)\, w(N'-1) \\ x(rN-N'+2)\, w(N'-2) \\ \vdots \\ x(rN-1)\, w(1) \\ x(rN)\, w(0) \end{bmatrix} \triangleq \begin{bmatrix} x_r(0) \\ x_r(1) \\ \vdots \\ x_r(N'-2) \\ x_r(N'-1) \end{bmatrix}. \tag{2.18}$$

Obviously there will be significant overlapping, or redundancy, between samples of adjacent frames. This redundancy will eliminate the requirement of correct frame timing, and will allow arbitrary frame positioning for recovering the correct signal.

Due to difficulties in finding the appropriate scrambling transformation of the N'-component vector $\overline{x}_r$, this vector is preprocessed such that it is compressed back to size N. This is accomplished by a particular transformation function F producing

$$\overline{u}_r = F(\overline{x}_r), \tag{2.19}$$

where

$$\overline{u}_r = \begin{bmatrix} u_r(0) \\ u_r(1) \\ \vdots \\ u_r(N-2) \\ u_r(N-1) \end{bmatrix}. \tag{2.20}$$

The scrambling transformation T is then carried out in the compressed N-component vector $\overline{u}_r$ resulting as

$$\overline{v}_r = T\overline{u}_r = \begin{bmatrix} v_r(0) \\ v_r(1) \\ \vdots \\ v_r(N-2) \\ v_r(N-1) \end{bmatrix}. \tag{2.21}$$

The vector $\bar{v}_r$ is then used to obtain the N samples of y(n) and hence the continuous-time scrambled signal y(t).

It remains to identify specifically N', the N'-sample window w(n) and the function F. To do this the authors propose, and later on show, that the asynchronous analog speech sample-based scrambling system put forth in this paper is simply a short-time Fourier analysis of x(n) (a time-sampled discrete signal representing original speech), cascaded with a short-time Fourier synthesis resulting in y(n) (the corresponding time-sampled discrete signal representing sampled speech). N' and w(n) are chosen to be

$$N' = 2LN \qquad \text{where L is integer, } L>>1 \tag{2.22}$$

and

$$w(n) = \begin{cases} \sin\left[\dfrac{(n-LN)\pi}{2LN}\right] & n = 0, 1, 2, \ldots, 2LN-1 \\ 0 & \text{otherwise} \end{cases} \tag{2.23}$$

respectively Furthermore, the transformation F can be defined as

$$u_r(k) = \sum_{l=0}^{2L-1} x_r(k+lN) \qquad k = 0, 1, 2, \ldots, N-1 \tag{2.24}$$

A small drawback of this system is that besides the processing delay, between input and output of the scrambler, there will be an additional delay equivalent to N' samples. This results from the time taken initially to collect the first N'-sample frame.

Another possible drawback is whether the degree of security in the proposed asynchronous analog speech sample-based scrambling system is degraded, as a result of the redundancy in the scrambled signal. The authors point out that preliminary studies on the degree of security have been conducted, with the results indicating a very good degree of practical security. They do not go into any details though, thus raising doubts as to the degree of validity of this statement.

An analog speech signal is fully described by the variation of its amplitude, with

respect to frequency and time. There are thus three characteristics of a signal which can be manipulated for the purpose of scrambling. The combined analog speech scrambling techniques of jig-saw puzzle scramblers, and sample-based scramblers can be classified according to the particular characteristic(s) which they manipulate. Furthermore, the collective advantages or disadvantages of each classification can be identified. This is accomplished in the paper entitled "Analog Scramblers for Speech Privacy" by Nuggehally S. Jayant [20]. According to Jayant, all analog scrambling techniques or algorithms can be classified as one-dimensional or two-dimensional. One-dimensional algorithms manipulate the time, frequency, or amplitude characteristics of an analog speech signal, while two-dimensional algorithms manipulate a combination of the above characteristics.

One-dimensional techniques which manipulate the frequency characteristic of an analog speech signal include frequency inversion, band-shift inversion, and band splitting. Frequency inversion consists of multiplying the speech signal by a carrier of 3300Hz, filtering out the higher side-band, and retaining the lower side-band, which contains a set of difference-frequency components, which completely specify the speech process. The important characteristic of this algorithm is that the carrier frequency of 3300Hz maintains the inverted signal in the same band as the original signal. Obviously, this algorithm lacks cryptanalytical value since its operation is purely deterministic On the other hand, if the carrier frequency is time-varying to a limited degree, the inverted signal will not remain in the same band as the original signal. This is referred to as frequency-hopping inversion, where the time-varying carrier offers some cryptanalytical value. Finally, if the input speech is sampled, frequency inversion of this speech signal will consist of inverting the sign of every other Nyquist sample [21]. Again, this has no cryptanalytical value.

Band-shift inversion is based on frequency inversion controlled with a key. The key allows the user to select the carrier frequency above 3300Hz, such that the extra carrier

frequency (above 3300Hz) forces the inverted signal outside the band (beyond 3000Hz). The part of the signal then extending beyond 3000Hz is put at the low frequency end starting at 300Hz. The introduction of the key puts some randomness in the frequency inversion operation. This can be seen as combining the principles of frequency inversion and frequency-hopping inversion.

The final technique to be discussed is bandsplitting. This consists of dividing the spectrum into f sub-bands and permuting these sub-bands in one of f! possible ways, based on a key. The key allows the user to select the particular carrier frequencies for the respective bands. This algorithm is carried out by multiplying each band by the specific carrier chosen, and filtering. A particular carrier puts a particular band in a new particular position in the spectrum, relative to the other bands. A variation of bandsplitting consists of using a time-varying key. This is referred to as a rolling-code bandsplitter.

All the above mentioned frequency manipulation techniques are examples of jig-saw puzzle scramblers. Being frequency manipulation techniques, they produce scrambled speech signals which allow for good transmission, since they are not sensitive to real-channel imperfections. The drawback of these techniques though is that the scrambled signals they produce allow a speech-like rhythm to pass at the output of the transmitter, which leads to high values of residual intelligibility.

One-dimensional techniques, which manipulate the time characteristic of an analog speech signal, include time inversion, time segment permutation (with hopping window or sliding window), and reverberation. Time inversion consists of sampling the speech signal within a block at the Nyquist rate, and inverting the order of speech samples in time within each block. This algorithm can lead to a substantial loss of intelligibility at the transmitter, if the block length is chosen to be in the order of 128 or 256 ms [22]. On the other hand, it has the same problem as basic frequency inversion, in that it is deterministic, and hence has no cryptanalytical value.

Time segment permutation with hopping window (or block TSP), consists of dividing the speech signal into blocks, and dividing each block into b segments. A special case of time segment permutation is time sample permutation, where the segment length is equal to the duration of one Nyquist sample. The smaller the segment length, the more the number of nonspeech-like discontinuities between segments, leading to bandwidth expansion for the scrambled speech signal. The choice of segment duration should reflect a compromise between the conflicting requirements of bandwidth expansion (a decreasing function of segment duration), and total communication delay (an increasing function of segment duration). As shown in [23], a segment length of 16 to 32 ms long is a good compromise. The segments within each block are chosen and transmitted in random order. The window remains on the current block until all its segments are transmitted, before moving to the next block. With b=8 only 0.1% of all 8! permutations allowed by this technique are considered good from the point of view of residual intelligibility at the transmitter.

Time segment permutation with sliding window consists of dividing the speech signal into segments, and taking an initial block of b segments as above, which corresponds to the initial window. Once one segment within this block is chosen at random and transmitted, all segments on one side of the transmitted segment are moved over by one stage in memory, and the vacant memory location is filled with a new incoming speech segment. The procedure is then repeated One precaution that this algorithm has to take, is that a segment entering the block has to be transmitted within a time duration of t time segments, overriding the random selection process if necessary. The optimal value of t is found to be 2b from the point of view of residual intelligibility at the transmitter, as shown in [24], as well as from the point of view of total number of permutation keys as shown in [25].

Finally, reverberation consists of purposely creating echos of past speech samples, and outputting these with each current speech sample. This is termed forward rever-

beration. Reverse reverberation on the other hand, consists of creating echos of future speech samples, and outputting these with each current speech sample. Reverse reverberation where individual speech sounds gradually build up, instead of decaying, allows for lower residual intelligibility at the transmitter. This is because the human ear is less accustomed to hearing evolving sound patterns, than it is to hearing decaying sound patterns

From all the above mentioned time manipulation techniques, time segment permutation with (hopping window or sliding window) is a jig-saw puzzle scrambling technique, which preserves signal bandwidth. The remaining techniques of time sample permutation, time inversion, and reverberation are sample-based scrambling techniques. Of these, the time sample permutation technique leads to bandwidth expansion, as would be expected, while time inversion, and reverberation are special cases leading to bandwidth preservation. All these time manipulation techniques mentioned above provide scrambled speech signals without any speech-like rhythm, but they are sensitive to real-channel imperfections. Furthermore, they require extremely large values of communication delay, beyond the expected processing delay, if they are to maintain acceptable levels of residual intelligibility.

The only technique known which manipulates the amplitude characteristic of an analog speech signal is masking. Masking of speech signals, also referred to as amplitude scrambling, can come in various forms. One form consists of the linear addition or multiplication of pseudorandom noise amplitudes to the speech signal. This allows for the possibility of a transmitter output that can sound like white noise, but at the cost of reducing the speech-to-channel noise ratio after descrambling at the receiver. Another method of masking consists of the nonlinear-modulo arithmetic of pseudorandom noise amplitudes to the speech signal. Masking of speech signals by linear addition is a jig-saw puzzle scrambling technique preserving bandwidth. On the other hand, masking of speech signals by linear multiplication is a special case of a jig-saw puzzle scrambling

technique which does not preserve bandwidth. Furthermore, masking of speech signals by nonlinear modulo-arithmetic is a sample-based scrambling technique, which leads to bandwidth expansion. Masking techniques are transmission sensitive but produce a noiselike output at the transmitter. Thus, they are similar to time manipulation techniques without the communication delay.

Two-dimensional algorithms result from cascading two one-dimensional algorithms, such that a speech signal is scrambled with respect to two of its characteristics, as mentioned previously. The respective strengths and weaknesses of the two one-dimensional component algorithms combine, and complement each other. This results in scrambled speech signals with lower residual intelligibility, and more cryptanalytical strength than can be provided than any one-dimensional algorithm alone. The speech-like rhythm characteristic resulting from frequency manipulations of speech is destroyed by time manipulations, while the high transmission sensitivity of time manipulations of speech is complemented, by the transmission robustness of frequency manipulations. Furthermore, the time delay resulting from any time manipulations of speech within a two-dimensional algorithm can be reduced, depending on the scrambling ability of the other accompanying one-dimensional algorithm.

CHAPTER III

SECURITY EVALUATION OF A NEW ANALOG SPEECH SCRAMBLING ALGORITHM USING HOPPING FILTERS

3.1 INTRODUCTION

Upon surveying previous related works on speech secure systems in general, it was realized that although the foundations of this field have been set, its full development is far from over. There is still room for development of new speech scrambling algorithms, as well as of ways in which these, and others previously developed, can be combined to enhance each others strengths, by obscuring each others weaknesses.

In the previous chapter it was made clear that digital speech encryption, although powerful, cannot be taken advantage of at present. The reason for this is that the telecommunication system started out with fully analog carriers, of which most remain until present. Some have been replaced with digital carriers, and although this process is ongoing, analog carriers still dominate digital carriers today. Thus, digital encryption still remains as an alternative for the future.

The next best thing to digital speech encryption is analog speech encryption. This constitutes a family of speech encryption techniques, which although employ analog transmission, use analog to digital and digital to analog conversions to obtain an intermediary digital signal, on which digital speech encryption techniques can be imposed. Although very strong, this area of speech security has been fully exhausted, especially with the constant developments of digital signal processing.

The only area of speech security which at present does not offer the highest levels of security, but which still allows room for further development is the area of analog speech scrambling.

The purpose of this chapter is thus to build on previous related works, as outlined in the previous chapter, in order to develop the area of analog speech scrambling one step further. It was seen from [12] that maintaining the two constraints of, perfect secrecy and use of a finite-size digital key, the degradation of an analog speech scrambling system cannot be less than $D'_N(p_X)$, where this is the degradation offered by an analog speech encryption system respecting the same constraints. Although this provides a lower bound for the degradation of all analog speech scrambling systems respecting the above constraints, the actual degradation is system dependent. Furthermore, it was put forth in [12] that if any of the stated constraints would have been violated, then a degradation below the minimum of $D'_N(p_X)$ could be achieved. Although perfect secrecy is very desirable it is an ideal security level which cannot be obtained and made use of in any practical sense. Therefore, an analog speech scrambling system which can provide a security level close to, but nonetheless inferior to perfect secrecy, will be ensured to have a degradation upper bounded by $D'_N(p_X)$. Such a security level is thus a desirable requirement for an analog speech scrambling system.

As mentioned previously, there are various jig-saw puzzle analog scrambling algorithms, as well as sample-based analog scrambling algorithms, which can be classified according to the characteristics (i.e. frequency, time or amplitude) of speech which they manipulate. From [20] it was seen that frequency manipulation algorithms in general produce scrambled speech signals which are not sensitive to real-channel imperfections, thus allowing for good transmission, but at the price of allowing a speech-like rhythm to pass at the output of the transmitter. A speech-like rhythm at the output of the transmitter results from frequency manipulation techniques, since these techniques are nothing more than a mapping of one finite set of phonemes to another such set. This could be avoided by changing the particular frequency manipulation periodically, or in other words changing the particular mapping of phonemes. Time manipulation algorithms on the other hand eliminate speech-like rhythm, but are sensitive to real-channel

imperfections and require some communication delay. Time manipulation algorithms can be made less sensitive to real-channel imperfections but at the price of larger communication delays. The transmission sensitivity of time manipulation algorithms results from the problem of bandwidth expansion, as well as that of preserving sample integrity. Bandwidth expansion results from the nonspeech-like discontinuities in the scrambled speech waveform, as a result of the re-ordering of speech samples. Preserving sample integrity between speech samples in the descrambler input, and corresponding speech samples of the scrambler output is difficult because of the discontinuities in scrambled speech. These discontinuities render a particular level of distortion (as a result of real-channel imperfections) perceptually much more objectionable, than the same level of distortion as applied to unpermuted speech, in which case successive sample distortions are continuous or slowly varying. Finally, masking, or amplitude scrambling algorithms allow for the output of the transmitter to sound like white noise, but are transmission sensitive. They are thus similar to time manipulation algorithms, without any communication delay. The additional drawback here is that amplitude scrambling algorithms involving linear addition or multiplication tend to decrease the speech-to-channel noise ratio.

From the above it seems appropriate to study one-dimensional frequency manipulation algorithms (in which the frequency manipulation is changed periodically), as well as amplitude scrambling algorithms, with the possibility of combining two algorithms (one from each category) to produce two-dimensional algorithms. This chapter examines a new one-dimensional frequency manipulation algorithm, which can be referred to as the hopping filters algorithm. Of all the frequency manipulation algorithms already known, the one, if any, to which it resembles the most is bandsplitting with a rolling code. This new algorithm also divides the spectrum into f sub-bands, but instead of permuting these sub-bands as governed by the key, the key enhances or attenuates the amplitude of each sub-band, which is equivalent to filtering the incoming speech with various hop-

ping filters, hence the name of the algorithm.

The other one-dimensional algorithms examined in this chapter involve amplitude scrambling of speech, using pseudorandom noise amplitudes, via linear addition as well as multiplication. These are referred to as amplitude addition scrambling and amplitude multiplication scrambling algorithms, respectively. Finally, this chapter examines the two-dimensional algorithms consisting of the hopping filters algorithm, with either of the two amplitude scrambling algorithms.

The general operation of all five scrambling algorithms as outlined above are described in section 3.2. Section 3.3 then evaluates these algorithms on the basis of perfect secrecy, residual intelligibility at the output of the transmitter, and cryptanalytical strength. By crossing these three factors, one of the five algorithms is chosen and optimally configured so that its particular operation

1) approaches perfect secrecy very closely,

2) provides least residual intelligibility at the output of the transmitter by resembling white noise as much as possible, and

3) offers maximum cryptanalytical strength as a result of offering the largest number of possible transformations for the incoming speech signal.

3.2 Proposed Analog Speech Scrambling Algorithms : One-Dimensional and Two-Dimensional

The first of three one-dimensional analog speech scrambling algorithms to be described is a new frequency manipulation algorithm [26] This algorithm consists of choosing a particular hopped filter from a group of n hopped filters, and passing the incoming source analog speech signal through this filter for a certain period of time, before hopping to another filter at a rate to be discussed later. Each hopped filter is a piece-wise combination of seven unity-gain bandpass filters, which are mutually exclusive

and collectively exhaust the frequency band 200Hz-2400Hz. Following each bandpass filter is a gain control device, which enhances or attenuates the incoming signal by one of several built-in factors or levels, as chosen from by a short binary codeword. The outputs of all seven gain control devices are then added to produce the final output of this algorithm as shown in figure 3.1.

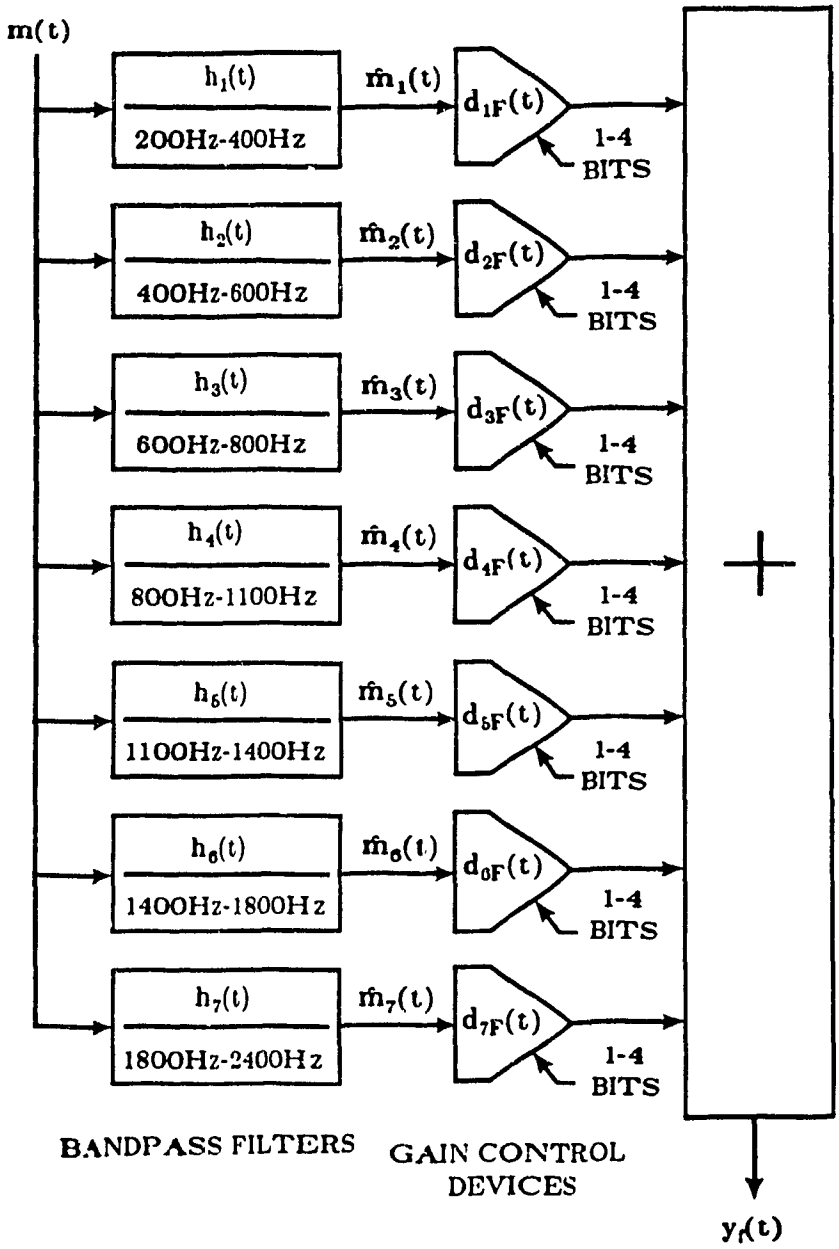

Figure 3.1 Details of the hopping filters algorithm.

The seven short binary codewords are respectively parts of seven pseudorandomly generated secret codes, which are produced independently from, but simultaneously with each other, and are the topic of the next chapter.

The time duration for which a particular hopped filter will be used is practically upper-bounded by $T_F = 1.25$ ms for reasons to be explained later, and lower-bounded by the time needed to generate seven 1-bit codewords in parallel (the smallest codewords that can be generated). The lower bound thus depends on the clock frequency of each code generator, which is selected as 3200Hz for reasons to be explained later As a result, the fastest available timing signal has a period of $T_c = 0.3125$ ms. This provides the lower bound on the time duration for which a particular hopped filter will be used. These bounds restrict the number of bits in a particular short codeword (which is the same for all seven such short codewords), from 1 to 4. As a result, the number of gain factors or levels, by which each unity-gain bandpass filter output can be enhanced or attenuated, as chosen by the seven short codewords, range from 2^1 (= 2) to 2^4 (= 16). This leads to the number of possible piece-wise hopped filters n to range from 2^7 to 16^7 (= 2^{28}), as well as the hopping rate ($\frac{1}{T_F}$) between filters to range from $\frac{1}{T_c}$ to $\frac{1}{4T_c}$ respectively. At this point, it might seem that the more the hopped filters to choose from (i.e. 2^{28}) the higher the security of the algorithm. This is true if the cryptanalytical attack consists of trying to find the particular hopped filter used, through exhaustively applying all possible inverse filters to the scrambled signal. Since this number of hopping filters is inversely proportional to the hopping rate, the residual intelligibility will be increased. On the other hand, if the cryptanalytical attack consists of simply listening to the scrambled signal then a lower residual intelligibility is required, which implies a faster hopping rate. Since the hopping rate is inversely proportional to the number of hopping filters, then fewer hopping filters will be necessary Since the cryptanalyst will follow the attack which best serves his/her purposes, it is up to the designer to provide a compromise between the two conflicting factors of maximum possible number of hopping

filters, and maximum possible hopping rate, thus providing overall security. This compromise will be resolved later with a design choice as to the actual value of T_F.

Let the source input analog speech signal to the hopping filters (HF) algorithm be a band-limited speech signal between 200Hz and 3200Hz, represented by the stochastic process $\mathbf{m}(t)$. At any particular time t the probability density of $\mathbf{m}(t)$ "is characterized in general by a very high probability of zero and near-zero amplitudes (related to pauses and low-energy segments of the speech waveform), by a significant probability of very high amplitudes, and by a monotonically decreasing function of amplitudes in between those extremes" [27]. Furthermore, let the output signals of the seven bandpass filters in this algorithm, which are filtered versions of $\mathbf{m}(t)$, be represented by the stochastic processes $\hat{\mathbf{m}}_1(t)$ ... $\hat{\mathbf{m}}_7(t)$ respectively. Finally, let the control inputs of the HF algorithm be seven identical but independent PAM processes. The levels of each of these processes are the gain factors by which a particular frequency band, represented by a particular stochastic process $\hat{\mathbf{m}}_1(t)$ ... $\hat{\mathbf{m}}_7(t)$, can be enhanced or attenuated, as chosen by a particular pseudo-randomly generated codeword. The seven PAM processes are represented as $\mathbf{d}_{1F}(t)$, ..., $\mathbf{d}_{JF}(t)$, ..., $\mathbf{d}_{7F}(t)$ such that

$$\mathbf{d}_{JF}(t) = \sum_{k=-\infty}^{+\infty} g_F(t-kT_F)\mathbf{D}_{JF}(k) \,. \tag{3.1}$$

$\mathbf{D}_{1F}(k)$, ..., $\mathbf{D}_{7F}(k)$ are seven identical, but independent uniformly distributed discrete-time stochastic processes, and $g_F(t-kT_F)$ is a gate function defined as

$$g_F(t-kT_F) = \begin{cases} 1 & \text{if } kT_F \leq t \leq (k+1)T_F \\ 0 & \text{if otherwise} \end{cases} , \tag{3.2}$$

where 0.3125 ms $\leq T_F \leq$ 1.25 ms depending on the design choice to resolve the compromise outlined above. Refer to figure 3.1 for the details of the HF algorithm as described above. The final output of this algorithm is denoted by the stochastic process $\mathbf{y}_f(t)$, which based on the above description is

$$\mathbf{y}_f(t) = \sum_{l=1}^{7} \hat{\mathbf{m}}_l(t)\mathbf{d}_{lF}(t) \,. \tag{3.3}$$

The second and third one-dimensional algorithms to be described are known algorithms of amplitude scrambling. One of them consists of linearly adding an analog signal to the input source analog speech signal (figure 3.2), while the other consists of multiplying the input source analog speech signal by an analog signal (figure 3.3).

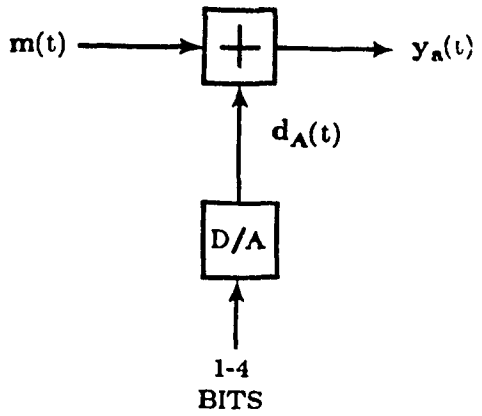

Figure 3.2 Details of the amplitude addition scrambling algorithm.

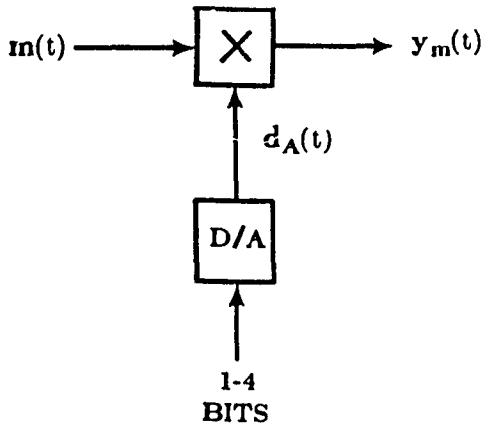

Figure 3.3 Details of the amplitude multiplication scrambling algorithm.

In both these cases the analog signal is made up of various dc levels, which are the outputs of a D/A converter to which the inputs are short codewords produced from the

code generator Using the same reasoning as in the timing of the hopping filters algorithm explained previously, the time duration for which a particular dc level will multiply, or be added to the input source analog speech signal ranges from $T_A = 0.3125$ ms to $T_A = 1.25$ ms Furthermore, the length of the codeword choosing a particular level will range from 1 to 4 bits, and as a result the number of possible codewords at the input of the D/A, or number of possible levels at the output of the D/A will range from 2^1 (= 2) to 2^4 (=16) Similarly, as before a compromise will have to be reached between the maximum possible number of levels to multiply, or be added to the incoming speech signal, and the maximum possible rate of change between these levels These are two conflicting factors both of which contribute in different ways to the security of the algorithm. This compromise will also be resolved later with a design choice as to the actual value of T_A

As before, let the source input analog speech signal to these two amplitude scrambling algorithms be a band-limited speech signal between 200Hz-3200Hz, represented by the stochastic process $\mathbf{m}(t)$ Furthermore, let the control input to these algorithms be a PAM process represented by $\mathbf{d_A}(t)$, such that

$$\mathbf{d_A}(t) = \sum_{k=-\infty}^{+\infty} g_A(t-kT_A)\mathbf{D_A}(k) \; . \tag{3.4}$$

$\mathbf{D_A}(k)$ is a uniformly distributed discrete-time stochastic process, and $g_A(t-kT_A)$ is a gate function defined as

$$g_A(t-kT_A) = \begin{cases} 1 & \text{if } kT_A \leq t \leq (k+1)T_A \\ 0 & \text{if otherwise} \end{cases} \; , \tag{3.5}$$

where 0 3125 ms $\leq T_A \leq$ 1.25 ms depending on the design choice to resolve the compromise outlined above. The levels of the PAM process $\mathbf{d_A}(t)$ are the levels which are added to, or multiply the input speech signal $\mathbf{m}(t)$, as shown in figure 3.2 or figure 3.3 respectively.

The input to both of these candidate algorithms is $\mathbf{m}(t)$, while their outputs are

$$\mathbf{y_a}(t) = \mathbf{m}(t) + \mathbf{d_A}(t) \, . \tag{3.6}$$

and

$$\mathbf{y_m}(t) = \mathbf{m}(t)\mathbf{d_A}(t) \tag{3.7}$$

for the amplitude addition scrambling (AAS) algorithm, and the amplitude multiplication scrambling (AMS) algorithm respectively.

The first of the two-dimensional analog speech scrambling algorithms consists of the HF algorithm described above, followed by the AAS algorithm in that order, with the combination referred to as HF-AAS. The output of the HF algorithm (3.3) is used as input to the AAS algorithm (3.6). Maintaining the input of this overall two-dimensional analog speech scrambling algorithm as $\mathbf{m}(t)$, and referring to the output as $\mathbf{z_a}(t)$ then

$$\mathbf{z_a}(t) = \left[\sum_{l=1}^{7} \hat{\mathbf{m}}_l(t)\mathbf{d_{lF}}(t) \right] + \mathbf{d_A}(t) \tag{3.8}$$

The second of the two-dimensional analog speech scrambling algorithms consists of the HF algorithm described above, followed by the AMS algorithm in that order, with the combination referred to as HF-AMS. The output of the HF algorithm (3.3) is used as input to the AMS algorithm (3.7). Maintaining the input of this overall two-dimensional analog speech scrambling algorithm as $\mathbf{m}(t)$ and referring to the output as $\mathbf{z_m}(t)$ then

$$\mathbf{z_m}(t) = \left[\sum_{l=1}^{7} \hat{\mathbf{m}}_l(t)\mathbf{d_{lF}}(t) \right] \mathbf{d_A}(t) \, . \tag{3.9}$$

3.3 Security Evaluation of Proposed Analog Speech Scrambling Algorithms

The first objective of this analysis is to find the cross-covariance

$$\mathbf{C_{mv}}(\tau) = \mathbf{E}\left[\mathbf{m}(t)\mathbf{v}(t+\tau)\right] - \mathbf{E}\left[\mathbf{m}(t)\right]\mathbf{E}\left[\mathbf{v}(t+\tau)\right] \tag{3.10}$$

[28], between the input process $\mathbf{m}(t)$ of each analog speech scrambling algorithm, and the output process $\mathbf{v}(t)$ of the same algorithm shifted by an amount τ, thus $\mathbf{v}(t+\tau)$ with respect to this shift. This is carried out so as to examine the degree of correlation between the input and output of a particular algorithm, with respect to a time difference of τ. The smaller this correlation the lesser the amount of information resulting at the output of the algorithm, with regard to the input of the algorithm. Thus, the ideal value of cross-covariance for a scrambling algorithm is zero. This would imply that the input and output are statistically uncorrelated, and hence the statistics of the output signal would convey no information as to the statistics of the input signal. This is a slightly weaker condition than, but nevertheless, resulting from the idea of perfect secrecy, which implies that the output of a scrambling algorithm is statistically independent from its input [12].

The first algorithm to be examined in terms of cross-covariance is the two-dimensional HF-AAS algorithm. The output of this algorithm is as shown in (3.8), and the shifted version of this output shifted by τ is

$$\mathbf{z}_a(t+\tau) = \left(\sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau) \right) + \mathbf{d}_A(t+\tau) . \tag{3.11}$$

Substituting (3.11) for $\mathbf{v}(t+\tau)$ in the definition of cross-covariance (3.10),

$$\begin{aligned} E\left[\mathbf{m}(t)\mathbf{z}_a(t+\tau)\right] &= E\left[\mathbf{m}(t)\left(\left(\sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau) \right) + \mathbf{d}_A(t+\tau) \right)\right] \\ &= E\left[\sum_{l=1}^{7} \mathbf{m}(t)\hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau) \right] + E\left[\mathbf{m}(t)\mathbf{d}_A(t+\tau)\right] \\ &= \sum_{l=1}^{7} E\left[\mathbf{m}(t)\hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau)\right] + E\left[\mathbf{m}(t)\mathbf{d}_A(t+\tau)\right] . \end{aligned} \tag{3.12}$$

The PAM processes $\mathbf{d}_{1F}(t+\tau)$, ..., $\mathbf{d}_{7F}(t+\tau)$ and $\mathbf{d}_A(t+\tau)$ are independent of the speech process $\mathbf{m}(t)$ and its filtered versions $\hat{\mathbf{m}}_1(t+\tau)$, ..., $\hat{\mathbf{m}}_7(t+\tau)$, hence

$$E\left[\mathbf{m}(t)\mathbf{z}_a(t+\tau)\right] = \left(\sum_{l=1}^{7} E\left[\mathbf{m}(t)\hat{\mathbf{m}}_l(t+\tau)\right] E\left[\mathbf{d}_{lF}(t+\tau)\right]\right) + E\left[\mathbf{m}(t)\right] E\left[\mathbf{d}_A(t+\tau)\right] . \tag{3.13}$$

At this point a constraint is put on the PAM processes $\mathbf{d}_{1F}(t+\tau)$, ..., $\mathbf{d}_{7F}(t+\tau)$ such that their levels have a constant mean μ_{1F}, ..., μ_{7F} respectively. Since the levels of these PAM processes are to be the same for each, then $\mu_{1F} = \ldots = \mu_{7F} = \mu_F$. Furthermore, the same constraint is applied to the PAM process $\mathbf{d}_A(t+\tau)$ and its mean is μ_A. As a result, $E[\mathbf{d}_{1F}(t+\tau)] = \ldots = E[\mathbf{d}_{7F}(t+\tau)] = \mu_F$ and $E[\mathbf{d}_A(t+\tau)] = \mu_A$ for all t

$$\begin{aligned} E\left[\mathbf{m}(t)\mathbf{z}_a(t+\tau)\right] &= \mu_F \sum_{l=1}^{7} E\left[\mathbf{m}(t)\hat{\mathbf{m}}_l(t+\tau)\right] + \mu_A E\left[\mathbf{m}(t)\right] \\ &= \mu_F E\left[\mathbf{m}(t)\sum_{l=1}^{7}\hat{\mathbf{m}}_l(t+\tau)\right] + \mu_A E\left[\mathbf{m}(t)\right] \end{aligned} \tag{3.14}$$

Also

$$\begin{aligned} E\left[\mathbf{m}(t)\right] E\left[\mathbf{z}_a(t+\tau)\right] &= E\left[\mathbf{m}(t)\right] E\left[\sum_{l=1}^{7}\hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau) + \mathbf{d}_A(t+\tau)\right] \\ &= E\left[\mathbf{m}(t)\right]\left(\sum_{l=1}^{7} E\left[\hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau)\right] + E\left[\mathbf{d}_A(t+\tau)\right]\right) . \end{aligned} \tag{3.15}$$

The PAM processes $\mathbf{d}_{1F}(t+\tau)$, ..., $\mathbf{d}_{7F}(t+\tau)$ are independent of the filtered speech processes $\hat{\mathbf{m}}_1(t+\tau)$, ..., $\hat{\mathbf{m}}_7(t+\tau)$, hence

$$E\left[\mathbf{m}(t)\right] E\left[\mathbf{z}_a(t+\tau)\right] = E\left[\mathbf{m}(t)\right]\left(\sum_{l=1}^{7} E\left[\hat{\mathbf{m}}_l(t+\tau)\right] E\left[\mathbf{d}_{lF}(t+\tau)\right] + E\left[\mathbf{d}_A(t+\tau)\right]\right) . \tag{3.16}$$

Putting the same constraints on the PAM processes as before $E[\mathbf{d}_{1F}(t+\tau)] = \ldots = E[\mathbf{d}_{7F}(t+\tau)] = \mu_F$ and $E[\mathbf{d}_A(t+\tau)] = \mu_A$ for all t.

$$\begin{aligned} E\left[\mathbf{m}(t)\right] E\left[\mathbf{z}_a(t+\tau)\right] &= \mu_F E\left[\mathbf{m}(t)\right] \sum_{l=1}^{7} E\left[\hat{\mathbf{m}}_l(t+\tau)\right] + \mu_A E\left[\mathbf{m}(t)\right] \\ &= \mu_F E\left[\mathbf{m}(t)\right] E\left[\sum_{l=1}^{7}\hat{\mathbf{m}}_l(t+\tau)\right] + \mu_A E\left[\mathbf{m}(t)\right] \end{aligned} \tag{3.17}$$

The cross-covariance then becomes

$$C_{mz_A}(\tau) = \mu_F \left(E\left[\mathbf{m}(t) \sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau) \right] - E\left[\mathbf{m}(t) \right] E\left[\sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau) \right] \right) . \tag{3.18}$$

Since the bandpass filters are of unity gain, mutually exclusive, and collectively exhaust the band 200Hz-2400Hz then,

$$\sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau) = \bar{\mathbf{m}}(t+\tau) , \tag{3.19}$$

where $\bar{\mathbf{m}}(t+\tau)$ is $\mathbf{m}(t+\tau)$ filtered by a bandpass filter of unity gain and bandwidth 200Hz-2400Hz Since speech $\mathbf{m}(t+\tau)$ ranges from 200Hz to 3200Hz, for all intents and purposes $\mathbf{m}(t+\tau) \approx \bar{\mathbf{m}}(t+\tau)$ and hence,

$$\sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau) \approx \mathbf{m}(t+\tau) . \tag{3.20}$$

The cross-covariance then becomes

$$C_{mz_A}(\tau) = \mu_F \left(E\left[\mathbf{m}(t)\mathbf{m}(t+\tau) \right] - E\left[\mathbf{m}(t) \right] E\left[\mathbf{m}(t+\tau) \right] \right) , \tag{3.21}$$

which is the product of the mean of the seven identical PAM processes representing the HF algorithm, and the autocovariance of the input process $\mathbf{m}(t)$.

The second algorithm to be examined in terms of cross-covariance is the two-dimensional HF-AMS algorithm. The output of this algorithm is as shown in (3.9), and the shifted version of this output shifted by τ is

$$\mathbf{z}_m(t+\tau) = \left(\sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau) \right) \mathbf{d}_A(t+\tau) . \tag{3.22}$$

Substituting (3.22) for $\mathbf{v}(t+\tau)$ in the definition of cross-covariance (3.10),

$$\begin{aligned} E\left[\mathbf{m}(t)\mathbf{z}_m(t+\tau) \right] &= E\left[\mathbf{m}(t) \left(\sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau) \right) \mathbf{d}_A(t+\tau) \right] \\ &= \sum_{l=1}^{7} E\left[\mathbf{m}(t)\hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau)\mathbf{d}_A(t+\tau) \right] . \end{aligned} \tag{3.23}$$

The PAM processes $\mathbf{d}_{1F}(t+\tau)$, ..., $\mathbf{d}_{7F}(t+\tau)$ and $\mathbf{d}_{A}(t+\tau)$ are independent of each other as well as independent of the speech processes $\hat{\mathbf{m}}_1(t+\tau)$, ..., $\hat{\mathbf{m}}_7(t+\tau)$ and $\mathbf{m}(t)$ hence,

$$E\left[\mathbf{m}(t)\mathbf{z}_m(t+\tau)\right] = \sum_{l=1}^{7} E\left[\mathbf{m}(t)\hat{\mathbf{m}}_l(t+\tau)\right] E\left[\mathbf{d}_{lF}(t+\tau)\right] E\left[\mathbf{d}_{A}(t+\tau)\right] . \qquad (3.24)$$

Again, the constraint of constant mean μ_{1F}, ..., μ_{7F} is put on the PAM processes $\mathbf{d}_{1F}(t+\tau)$, ..., $\mathbf{d}_{7F}(t+\tau)$ respectively. Since they are identical processes (i.e. same levels) their mean is $\mu_{1F} = \ldots = \mu_{7F} = \mu_F$. Furthermore, the same constraint is applied to the PAM process $\mathbf{d}_{A}(t+\tau)$ and its mean is μ_A As a result, $E[\mathbf{d}_{1F}(t+\tau)] = \ldots = E[\mathbf{d}_{7F}(t+\tau)] = \mu_F$ and $E[\mathbf{d}_{A}(t+\tau)] = \mu_A$ for all t Then

$$\begin{aligned} E\left[\mathbf{m}(t)\mathbf{z}_m(t+\tau)\right] &= \mu_A\mu_F \sum_{l=1}^{7} E\left[\mathbf{m}(t)\hat{\mathbf{m}}_l(t+\tau)\right] \\ &= \mu_A\mu_F E\left[\mathbf{m}(t)\sum_{l=1}^{7}\hat{\mathbf{m}}_l(t+\tau)\right] . \end{aligned} \qquad (3.25)$$

Also

$$\begin{aligned} E\left[\mathbf{m}(t)\right]E\left[\mathbf{z}_m(t+\tau)\right] &= E\left[\mathbf{m}(t)\right]E\left[\left(\sum_{l=1}^{7}\hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau)\right)\mathbf{d}_{A}(t+\tau)\right] \\ &= E\left[\mathbf{m}(t)\right]\left\{\sum_{l=1}^{7} E\left[\hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau)\mathbf{d}_{A}(t+\tau)\right]\right\} \end{aligned} \qquad (3.26)$$

The PAM processes $\mathbf{d}_{1F}(t+\tau)$, ..., $\mathbf{d}_{7F}(t+\tau)$ and $\mathbf{d}_{A}(t+\tau)$ are independent of each other as well as of the speech processes $\hat{\mathbf{m}}_1(t+\tau)$, ..., $\hat{\mathbf{m}}_7(t+\tau)$, and hence

$$E\left[\mathbf{m}(t)\right]E\left[\mathbf{z}_m(t+\tau)\right] = E\left[\mathbf{m}(t)\right]\left\{\sum_{l=1}^{7} E\left[\hat{\mathbf{m}}_l(t+\tau)\right]E\left[\mathbf{d}_{lF}(t+\tau)\right]E\left[\mathbf{d}_{A}(t+\tau)\right]\right\} . \qquad (3.27)$$

Putting the same constraints on the PAM processes as before, $E[\mathbf{d}_{1F}(t+\tau)] = \ldots = E[\mathbf{d}_{7F}(t+\tau)] = \mu_F$, and $E[\mathbf{d}_{A}(t+\tau)] = \mu_A$ for all t. Then

$$\begin{aligned} E\left[\mathbf{m}(t)\right]E\left[\mathbf{z}_m(t+\tau)\right] &= \mu_A\mu_F E\left[\mathbf{m}(t)\right]\sum_{l=1}^{7} E\left[\hat{\mathbf{m}}_l(t+\tau)\right] \\ &= \mu_A\mu_F E\left[\mathbf{m}(t)\right]E\left[\sum_{l=1}^{7}\hat{\mathbf{m}}_l(t+\tau)\right] . \end{aligned} \qquad (3.28)$$

The cross-covariance $C_{mz_m}(\tau)$ then becomes

$$C_{mz_m}(\tau) = \mu_A \mu_F \left(E\left[\mathbf{m}(t) \sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau) \right] - E\left[\mathbf{m}(t) \right] E\left[\sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau) \right] \right) . \tag{3.29}$$

Using the same reasoning as before, (3.20) applies, and is substituted in (3.29).

Hence

$$C_{mz_m}(\tau) = \mu_A \mu_F \left(E\left[\mathbf{m}(t)\mathbf{m}(t+\tau) \right] - E\left[\mathbf{m}(t) \right] E\left[\mathbf{m}(t+\tau) \right] \right) , \tag{3.30}$$

which is the product of the mean of the seven identical PAM processes representing the HF algorithm, the mean of the process representing the AMS algorithm, and the autocovariance of the input speech process $\mathbf{m}(t)$.

The last three algorithms to be examined in terms of cross-covariance are the one-dimensional HF algorithm, and the one-dimensional AAS and AMS algorithms. The outputs of these algorithms are as shown in (3.3), (3.6), and (3.7), and the shifted versions of these outputs shifted by τ are

$$\mathbf{y}_f(t+\tau) = \sum_{l=1}^{7} \hat{\mathbf{m}}_l(t+\tau)\mathbf{d}_{lF}(t+\tau) , \tag{3.31}$$

$$\mathbf{y}_a(t+\tau) = \mathbf{m}(t+\tau) + \mathbf{d}_A(t+\tau) , \tag{3.32}$$

and

$$\mathbf{y}_m(t+\tau) = \mathbf{m}(t+\tau)\mathbf{d}_A(t+\tau) \tag{3.33}$$

respectively. Substituting (3.31), (3.32), and (3.33) for $\mathbf{v}(t+\tau)$ in the definition of cross-covariance (3.10), and following similar steps as in the previous calculations of cross-covariance results in the cross-covariances of these algorithms being

$$C_{my_f}(\tau) = \mu_F \left(E\left[\mathbf{m}(t)\mathbf{m}(t+\tau) \right] - E\left[\mathbf{m}(t) \right] E\left[\mathbf{m}(t+\tau) \right] \right) , \tag{3.34}$$

$$C_{my_a}(\tau) = E\left[\mathbf{m}(t)\mathbf{m}(t+\tau) \right] - E\left[\mathbf{m}(t) \right] E\left[\mathbf{m}(t+\tau) \right] , \tag{3.35}$$

and

$$C_{my_m}(\tau) = \mu_A \left(E\left[m(t)m(t+\tau)\right] - E\left[m(t)\right]E\left[m(t+\tau)\right] \right) \qquad (3\ 36)$$

respectively. The code generators choosing between the various levels of the PAM processes discussed so far are not biased in any way As a result, the levels of each PAM process at any particular time have a uniform distribution Given this fact, and tightening the constraint of constant mean to a mean of 0 for all the PAM processes implies that the sum of the levels should be 0 The single PAM process representing the AAS or AMS algorithm requires the same levels for the scrambling, as well as the descrambling procedures, with a one-to-one correspondence. The seven identical PAM processes representing the HF algorithm require that their levels in the scrambling procedure are multiplicative inverses of their levels in the descrambling procedure, and vice versa, again with a one-to-one correspondence. Choosing the levels of all the PAM processes accordingly results in $\mu_F = \mu_A = 0$, and making use of the fact that the speech process has zero mean amplitudes for any particular time t ($E[\mathbf{m}(t)] = E[\mathbf{m}(t+\tau)] = 0$) [27], then the cross-covariances derived so far all become 0, with the exception of that of the AAS algorithm. The latter cross-covariance is $R_m(\tau)$, which is equal to $E[\mathbf{m}(t)\mathbf{m}(t+\tau)]$, and is the autocorrelation of the speech input process $\mathbf{m}(t)$. Thus, for all the algorithms except that of AAS there is no correlation between input and output. As a result, when dealing with any of these algorithms a cryptanalyst cannot obtain information as to the statistics of the input signal to the algorithm, by observing the statistics of the output signal from the algorithm. In the case of the one-dimensional AAS algorithm the above does not hold since the autocorrelation of the speech process is nonzero, as will be shown later. Thus, this particular algorithm does not measure up to the others for analog speech scrambling, and is thus not examined further.

The second objective of this analysis is to find the autocorrelation between the output process $\mathbf{z}(t)$ of each algorithm, and the shifted version of the same process $\mathbf{z}(t+\tau)$, with respect to the amount of shift τ. This is carried out so as to examine how close the

autocorrelation of each algorithm is to the autocorrelation of white noise, which is

$$R_w(\tau) = \frac{N_0}{2}\delta(\tau) \tag{3.37}$$

[29]. This autocorrelation implies that no matter how close in time two samples are taken, they are uncorrelated. This is the ideal autocorrelation desired for the output of a scrambling algorithm. It would imply that the information contained in one sample of the output signal, would never appear again in any future output signals, no matter how close they are taken in time to the original signal. Thus, the cryptanalyst would not be able to confirm any information obtained, by relating to each other a few close samples in time, of the output scrambled signal Furthermore, the closer the autocorrelation of the output signal of an analog speech scrambling algorithm is to a delta function, the more the output will sound like white noise. Of course, the delta autocorrelation function of white noise can never be attained, as will be made clearer later when reference is made to the corresponding power spectral density.

The first of the two-dimensional analog speech scrambling algorithms to be examined in terms of autocorrelation is the HF-AAS algorithm. The output of this algorithm is as shown in (3.8), and the shifted version of the output shifted by τ is as shown in (3.11). Autocorrelation is defined as

$$R_{vv}(\tau) = E\left[v(t)v(t+\tau)\right] \tag{3.38}$$

[28]. Using j as a dummy variable in (3.11), and substituting (3.8) and (3.11) for $v(t)$ and $v(t+\tau)$ respectively in (3.38) results in

$$\begin{aligned} R_{z_1z_2}(\tau) &= E\left[\left(\left(\sum_{i=1}^{7}\hat{m}_i(t)d_{iF}(t)\right) + d_A(t)\right)\left(\left(\sum_{j=1}^{7}\hat{m}_j(t+\tau)d_{jF}(t+\tau)\right) + d_A(t+\tau)\right)\right] \\ &= \sum_{i=1}^{7}\sum_{j=1}^{7}E\left[\hat{m}_i(t)\hat{m}_j(t+\tau)d_{iF}(t)d_{jF}(t+\tau)\right] + \sum_{j=1}^{7}E\left[\hat{m}_j(t+\tau)d_{jF}(t+\tau)d_A(t)\right] \\ &\quad + \sum_{i=1}^{7}E\left[\hat{m}_i(t)d_{iF}(t)d_A(t+\tau)\right] + E\left[d_A(t)d_A(t+\tau)\right]. \end{aligned} \tag{3.39}$$

As before, all the PAM processes are independent of each other, as well as independent of the speech process and its filtered versions, hence

$$R_{z_a z_a}(\tau) = \sum_{\substack{i=1 j=1 \\ i=j}}^{7}\sum^{7} \left(E\left[\hat{m}_i(t)\hat{m}_j(t+\tau)\right] E\left[d_{iF}(t)d_{jF}(t+\tau)\right] \right)$$

$$+ \sum_{\substack{i=1 j=1 \\ i\neq j}}^{7}\sum^{7} \left(E\left[\hat{m}_i(t)\hat{m}_j(t+\tau)\right] E\left[d_{iF}(t)\right] E\left[d_{jF}(t+\tau)\right] \right)$$

$$+ \sum_{j=1}^{7} \left(E\left[\hat{m}_j(t+\tau)\right] E\left[d_{jF}(t+\tau)\right] E\left[d_A(t)\right] \right)$$

$$+ \sum_{i=1}^{7} \left(E\left[\hat{m}_i(t)\right] E\left[d_{iF}(t)\right] E\left[d_A(t+\tau)\right] \right) + E\left[d_A(t)d_A(t+\tau)\right] . \quad (3.40)$$

Applying the constraint of constant mean to the PAM processes, and following the same reasoning as before, $E[d_{1F}(t)] = E[d_{1F}(t+\tau)] = \ldots = E[d_{7F}(t)] = E[d_{7F}(t+\tau)] = \mu_F$, and $E[d_A(t)] = E[d_A(t+\tau)] = \mu_A$. Then

$$R_{z_a z_a}(\tau) = \sum_{i=1}^{7} R_{\hat{m}_i}(\tau) R_{d_{iF}}(\tau) + \mu_F^2 \sum_{\substack{i=1 j=1 \\ i\neq j}}^{7}\sum^{7} R_{\hat{m}_i\hat{m}_j}(\tau)$$

$$+ \mu_F\mu_A \sum_{j=1}^{7} E\left[\hat{m}_j(t+\tau)\right] + \mu_F\mu_A \sum_{i=1}^{7} E\left[\hat{m}_i(t)\right] + R_{d_A}(\tau) , \quad (3.41)$$

where $R_{\hat{m}_i}(\tau)$ is the autocorrelation of the i-th filtered speech process, $R_{\substack{\hat{m}_i\hat{m}_j \\ i\neq j}}(\tau)$ is the cross-correlation of the i-th and j-th filtered speech processes, $R_{d_{iF}}(\tau)$ is the autocorrelation of the i-th out of the seven identical PAM processes representing the HF algorithm, and $R_{d_A}(\tau)$ is the autocorrelation of the PAM process $d_A(t)$ representing the AAS algorithm. Since the seven PAM processes representing the HF algorithm are identical, their autocorrelations are equal and hence, $R_{d_{1F}}(\tau) = \ldots = R_{d_{7F}}(\tau) = R_{d_F}(\tau)$. Thus,

$$R_{z_a z_a}(\tau) = R_{d_F} \sum_{i=1}^{7} R_{\hat{m}_i}(\tau) + \mu_F^2 \sum_{\substack{i=1 j=1 \\ i\neq j}}^{7}\sum^{7} R_{\hat{m}_i\hat{m}_j}(\tau)$$

$$+ \mu_F\mu_A E\left[\sum_{j=1}^{7} \hat{m}_j(t+\tau)\right] + \mu_F\mu_A E\left[\sum_{i=1}^{7} \hat{m}_i(t)\right] + R_{d_A}(\tau) . \tag{3.42}$$

Using the same reasoning as before, and replacing dummy variable i for j in (3.20), as well as noting that a variation of (3.20)

$$\sum_{i=1}^{7} \hat{m}_i(t) \approx m(t) \tag{3.43}$$

applies, results in

$$R_{z_a z_a}(\tau) = R_{d_F}(\tau)\sum_{i=1}^{7} R_{\hat{m}_i}(\tau) + \mu_F{}^2 \sum_{\substack{i=1 \\ i\neq j}}^{7}\sum_{j=1}^{7} R_{\hat{m}_i\hat{m}_j}(\tau)$$

$$+ \mu_F\mu_A\left(E\left[m(t+\tau) + m(t)\right]\right) + R_{d_A}(\tau) . \tag{3.44}$$

The second of the two-dimensional analog speech scrambling algorithms to be examined in terms of autocorrelation is the HF-AMS algorithm. The output of this algorithm is as shown in (3.9), and the shifted version of this output shifted by τ is as shown in (3.22). Using j as a dummy variable in (3.22), and substituting (3.9) and (3.22) for $v(t)$ and $v(t+\tau)$ respectively in (3.38) results in

$$R_{z_m z_m}(\tau) = E\left[\left(\sum_{i=1}^{7} \hat{m}_i(t) d_{iF}(t)\right)\left(d_A(t)\right)\left(\sum_{j=1}^{7} \hat{m}_j(t+\tau) d_{jF}(t+\tau)\right)\left(d_A(t+\tau)\right)\right]$$

$$= \sum_{i=1}^{7}\sum_{j=1}^{7} E\left[\hat{m}_i(t)\hat{m}_j(t+\tau) d_{iF}(t) d_{jF}(t+\tau) d_A(t) d_A(t+\tau)\right] . \tag{3.45}$$

As before, all the PAM processes are independent of each other, as well as independent of the speech process and its filtered versions, hence

$$R_{z_m z_m}(\tau) = \sum_{\substack{i=1 \\ i=j}}^{7}\sum_{j=1}^{7}\left(E\left[\hat{m}_i(t)\hat{m}_j(t+\tau)\right]E\left[d_{iF}(t) d_{jF}(t+\tau)\right]E\left[d_A(t) d_A(t+\tau)\right]\right)$$

$$+ \sum_{\substack{i=1 \\ i \neq j}}^{7} \sum_{j=1}^{7} \left\{ E\left[\hat{m}_i(t)\hat{m}_j(t+\tau)\right] E\left[d_{iF}(t)\right] E\left[d_{jF}(t+\tau)\right] E\left[d_A(t)d_A(t+\tau)\right] \right\} . \tag{3.46}$$

Applying the constraint of constant mean to the PAM processes, and following the same reasoning as before, $E[d_{1F}(t)] = E[d_{1F}(t+\tau)] = \ldots = E[d_{7F}(t)] = Ed_{7F}(t+\tau)] = \mu_F$, and $E[d_A(t)] = E[d_A(t+\tau)] = \mu_A$. Then

$$R_{z_m z_m}(\tau) = R_{d_A}(\tau) \sum_{i=1}^{7} R_{\hat{m}_i}(\tau) R_{d_{iF}}(\tau) + \mu_F^2 R_{d_A}(\tau) \sum_{\substack{i=1 \\ i \neq j}}^{7} \sum_{j=1}^{7} R_{\hat{m}_i \hat{m}_j}(\tau) , \tag{3.47}$$

where $R_{\hat{m}_i}(\tau)$, $R_{\substack{\hat{m}_i \hat{m}_j \\ i \neq j}}(\tau)$, $R_{d_{iF}}(\tau)$, and $R_{d_A}(\tau)$ are as defined previously. As before, $R_{d_{1F}}(\tau) = \ldots = R_{d_{7F}}(\tau) = R_{d_F}(\tau)$, thus

$$R_{z_m z_m}(\tau) = R_{d_A}(\tau) R_{d_F}(\tau) \sum_{i=1}^{7} R_{\hat{m}_i}(\tau) + \mu_F^2 R_{d_A}(\tau) \sum_{\substack{i=1 \\ i \neq j}}^{7} \sum_{j=1}^{7} R_{\hat{m}_i \hat{m}_j}(\tau) . \tag{3.48}$$

The two one-dimensional analog speech scrambling algorithms remaining for discussion (after eliminating the AAS algorithm, because of its poor cross-covariance in relation to the other algorithms) are the HF algorithm, and the AMS algorithm. Following similar steps as in the cases of the two two-dimensional algorithms, the autocorrelations of these one-dimensional algorithms become

$$R_{y_p y_f}(\tau) = R_{d_F}(\tau) \sum_{i=1}^{7} R_{\hat{m}_i}(\tau) + \mu_F^2 \sum_{\substack{i=1 \\ i \neq j}}^{7} \sum_{j=1}^{7} R_{\hat{m}_i \hat{m}_j}(\tau) , \tag{3.49}$$

and

$$R_{y_m y_m}(\tau) = R_m(\tau) R_{d_A}(\tau) \tag{3.50}$$

respectively. Respecting the constraint of zero mean for all the PAM processes as discussed before, $\mu_F = 0$ and $\mu_A = 0$. Since speech amplitudes have zero mean, and since the correlations that exist among these amplitudes depend on the time difference

between them, then according to [28] the process representing speech, in this case $\mathbf{m}(t)$ is Wide Sense Stationary (WSS). According to [30] if a WSS process is applied to the input of a time-invariant linear network with transfer function H(f), then the output power spectral density is the product of $|H(f)|^2$ and the input power spectral density. In this case each bandpass filter is a linear time-invariant network. The autocorrelation $R_{m_j}(\tau)$, and the power spectral density $S_{m_j}(f)$ at the output of the j-th bandpass filter are related as follows:

$$F\left[R_{m_j}(\tau)\right] = S_{m_j}(f) \tag{3.51}$$

For all bandpass filters

$$\sum_{j=1}^{7} F\left[R_{m_j}(\tau)\right] = \sum_{j=1}^{7} S_{m_j}(f) \tag{3.52}$$

$$F\left[\sum_{j=1}^{7} R_{m_j}(\tau)\right] = \sum_{j=1}^{7} |H_j(f)|^2 S_m(f)$$

$$= S_m(f) \sum_{j=1}^{7} |H_j(f)|^2 . \tag{3.53}$$

Since the bandpass filters are of unity-gain, mutually exclusive, and collectively exhaust the band 200Hz-2400Hz then

$$F\left[\sum_{j=1}^{7} R_{m_j}(\tau)\right] = S_m(f)\,|X(f)|^2 , \tag{3.54}$$

where X(f) is a unity gain bandpass filter with bandwidth 200Hz-2400Hz. Since the voice bandwidth is 200Hz-3200Hz, if speech signals above 2400Hz are disregarded, as also approximated previously, then $S_m(f)\,|X(f)|^2 \approx S_m(f)$. Thus,

$$F\left[\sum_{j=1}^{7} R_{m_j}(\tau)\right] \approx S_m(f) \tag{3.55}$$

$$\sum_{j=1}^{7} R_{m_j}(\tau) \approx F^{-1}\left[S_m(f)\right]$$

$$\approx R_m(\tau) . \tag{3.56}$$

The autocorrelations of the four algorithms examined above can then be summarized as shown below.

Case 1. HF-AAS

$$R_{z_a z_a}(\tau) = R_{d_F}(\tau)R_m(\tau) + R_{d_A}(\tau) \tag{3.57}$$

Case 2. HF-AMS

$$R_{z_m z_m}(\tau) = R_{d_A}(\tau)R_{d_F}(\tau)R_m(\tau) \tag{3.58}$$

Case 3. HF

$$R_{y_f y_f}(\tau) = R_{d_F}(\tau)R_m(\tau) \tag{3.59}$$

Case 4. AMS

$$R_{y_m y_m}(\tau) = R_m(\tau)R_{d_A}(\tau) \tag{3.60}$$

Refer to figure 3.4 for the long time (55 s) averaged autocorrelation of speech denoted C(n), in terms of Nyquist samples n (125 μs), as a result of speech from two males and two females.

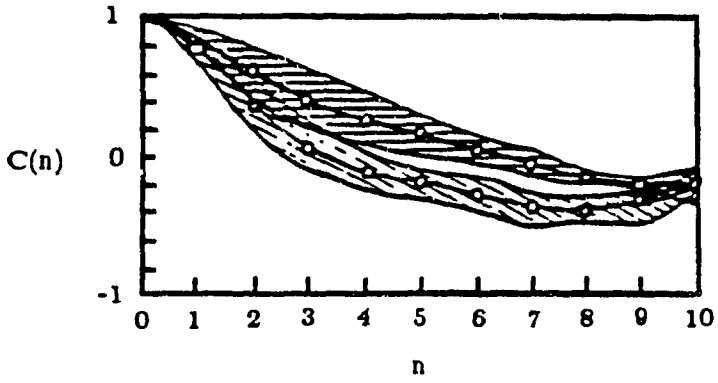

Figure 3.4 Long time autocorrelation function of speech [27].

The top curve represents low pass-filtered (0Hz-3400Hz) speech, and the bottom curve represents bandpass-filtered (200Hz-3400Hz) speech. Both curves are normalized by the expected (time-averaged) value of squared amplitudes, or the average power of speech. For purposes of analysis, points from the bottom curve were used to obtain a polynomial representing this curve. This polynomial denoted by $C_p(\tau)$ is of degree 10, and represents

the autocorrelation of speech (200Hz-3400Hz) in the interval $|\tau| \leq 1.25$ ms, since autocorrelation is an even function. The product of this polynomial, and the average power of speech $R_m[0](= E[m^2(t)])$ is the autocorrelation of speech $R_m(\tau)$ used in the analysis of the various analog speech scrambling algorithms. Hence,

$$R_m(\tau) = R_m[0]C_p(\tau) \qquad \text{for } |\tau| \leq 1.25 \text{ ms}. \tag{3.61}$$

An upper bound had to be chosen for the hopping rate in the HF algorithm, or for the time taken to change levels in the AMS algorithm. This is because the longer it takes for the changes to occur the easier the human ear may adapt to the current situation, and begin to understand what is being said. Since the autocorrelation curve available ranged up to 1.25 ms, for analysis purposes the upper bound was chosen to be 1.25 ms.

In general, a PAM process changes levels every T units of time but maintains a certain level for c units of time (the duty cycle factor), such that $c \leq T$. The autocorrelation of such a process consists of triangles each of base 2c, and centered at multiples of T with height $cR[s]/T$, where s is the time in discrete time units between the PAM process and its shifted version (figure 3.5).

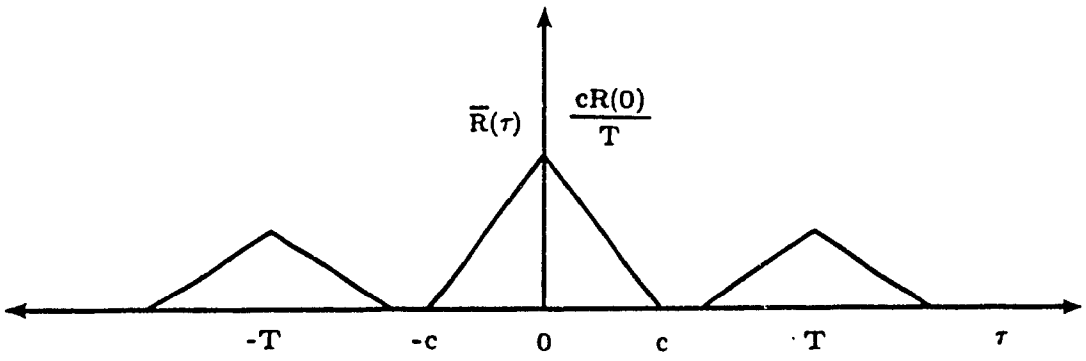

Figure 3.5 Autocorrelation of a PAM process [28].

Applying this general form of the autocorrelation to the PAM processes at hand, where c=T, the autocorrelation of each of these processes consists of triangles each of base 2T

and centered at multiples of T with height R[s]. Thus, for the seven identical PAM processes $\mathbf{d}_{1F}(t)$, ..., $\mathbf{d}_{7F}(t)$ representing the HF algorithm, and the one PAM process $\mathbf{d}_A(t)$ representing the AMS algorithm, the autocorrelation functions are

$$R_{d_F}(\tau) = \sum_{s=-\infty}^{+\infty} g(\tau - sT_F)\left[(s+1)R_{d_F}[s] - sR_{d_F}[s+1] + \frac{\tau}{T_F}\left(R_{d_F}[s+1] - R_{d_F}[s]\right)\right], \quad (3.62)$$

and

$$R_{d_A}(\tau) = \sum_{s=-\infty}^{+\infty} g(\tau - sT_A)\left[(s+1)R_{d_A}[s] - sR_{d_A}[s+1] + \frac{\tau}{T_A}\left(R_{d_A}[s+1] - R_{d_A}[s]\right)\right] \quad (3.63)$$

respectively, where $R_{d_F}[s]$, $R_{d_F}[s+1]$, $R_{d_A}[s]$, and $R_{d_A}[s+1]$ depend on the levels of the PAM processes, and

$$g(\tau - sT_F) = \begin{cases} 1 & \text{if } sT_F \le \tau \le (s+1)T_F \\ 0 & \text{if otherwise} \end{cases} \quad (3.64)$$

and

$$g(\tau - sT_A) = \begin{cases} 1 & \text{if } sT_A \le \tau \le (s+1)T_A \\ 0 & \text{if otherwise} \end{cases}, \quad (3.65)$$

where T_F and T_A are between 0.3125 ms and 1.25 ms, as discussed previously.

From the conditions and constraints imposed on the levels of the PAM processes so far, the levels of the seven identical PAM processes $\mathbf{d}_{1F}(t)$, ..., $\mathbf{d}_{7F}(t)$ representing the HF algorithm can be defined as $\alpha = (\pm p_1, \pm p_2, \pm p_3, \ldots, \pm p_{\frac{k_F}{2}})$, where k_F is the number of levels. The mean is then defined as

$$\mu_F = \frac{1}{k_F}\sum_{l=1}^{k_F}\alpha_l = 0 . \quad (3.66)$$

The autocorrelation in discrete time units s (multiples of T_F) was found to be

$$R_{d_F}[s] = E\left[D_{F_0}D_{F_s}\right] = \begin{cases} \frac{1}{k_F}\sum_{l=1}^{k_F}\alpha_l^2 & \text{if } s = 0 \\ \frac{1}{k_F^2}\left(\sum_{l=1}^{k_F}\alpha_l\right)^2 & \text{if } s \neq 0 \end{cases} . \tag{3.67}$$

Similarly, the levels of the PAM process representing the AMS algorithm can be defined as $\beta = (\pm q_1, \pm q_2, \pm q_3, \ldots, \pm q_{\frac{k_A}{2}})$, where k_A is the number of levels. The mean is then

$$\mu_A = \frac{1}{k_A}\sum_{l=1}^{k_A}\beta_l = 0 . \tag{3.68}$$

The autocorrelation in discrete time units s (multiples of T_A) was found to be

$$R_{d_A}[s] = E\left[D_{A_0}D_{A_s}\right] = \begin{cases} \frac{1}{k_A}\sum_{l=1}^{k_A}\beta_l^2 & \text{if } s = 0 \\ \frac{1}{k_A^2}\left(\sum_{l=1}^{k_A}\beta_l\right)^2 & \text{if } s \neq 0 \end{cases} . \tag{3.69}$$

Combining (3.66) and (3.67), and substituting in (3.62) results in

$$R_{d_F}(\tau) = \begin{cases} R_{d_F}[0]\left(1 - \frac{|\tau|}{T_F}\right) & \text{if } |\tau| < T_F \\ 0 & \text{if otherwise} \end{cases} . \tag{3.70}$$

Similarly, combining (3.68), (3.69), and substituting in (3.63) results in

$$R_{d_A}(\tau) = \begin{cases} R_{d_A}[0]\left(1 - \frac{|\tau|}{T_A}\right) & \text{if } |\tau| < T_A \\ 0 & \text{if otherwise} \end{cases} . \tag{3.71}$$

Substituting (3.61), (3.70), and (3.71), in (3.57), (3.58), (3.59), and (3.60) results in the following autocorrelations:

Case 1a) HF-AAS ($T_F \leq T_A$)

$$R_{z_a z_a}(\tau) = \begin{cases} R_{d_F}[0]R_m[0]\left(1 - \frac{|\tau|}{T_F}\right)\left(C_p(\tau)\right) + R_{d_A}[0]\left(1 - \frac{|\tau|}{T_A}\right) & \text{if } |\tau| \leq T_F \\ R_{d_A}[0]\left(1 - \frac{|\tau|}{T_A}\right) & \text{if } T_F \leq |\tau| \leq T_A \\ 0 \quad \text{if } |\tau| \geq T_A \end{cases} \tag{3.72}$$

Case 1b) HF-AAS ($T_F \geq T_A$)

$$R_{z_a z_a}(\tau) = \begin{cases} R_{d_F}[0]R_m[0]\left(1 - \frac{|\tau|}{T_F}\right)\left(C_p(\tau)\right) + R_{d_A}[0]\left(1 - \frac{|\tau|}{T_A}\right) & \text{if } |\tau| \leq T_A \\ R_{d_F}[0]R_m[0]\left(1 - \frac{|\tau|}{T_F}\right)\left(C_p(\tau)\right) & \text{if } T_A \leq |\tau| \leq T_F \\ 0 \quad \text{if } |\tau| \geq T_F \end{cases} \tag{3.73}$$

Case 2a) HF-AMS ($T_F \leq T_A$)

$$R_{z_m z_m}(\tau) = \begin{cases} R_{d_F}[0]R_{d_A}[0]R_m[0]\left(1 - \frac{|\tau|}{T_F}\right)\left(1 - \frac{|\tau|}{T_A}\right)\left(C_p(\tau)\right) & \text{if } |\tau| \leq T_F \\ 0 \quad \text{if } |\tau| \geq T_F \end{cases} \tag{3.74}$$

Case 2b) HF-AMS ($T_F \geq T_A$)

$$R_{z_m z_m}(\tau) = \begin{cases} R_{d_F}[0]R_{d_A}[0]R_m[0]\left(1 - \frac{|\tau|}{T_F}\right)\left(1 - \frac{|\tau|}{T_A}\right)\left(C_p(\tau)\right) & \text{if } |\tau| \leq T_A \\ 0 \quad \text{if } |\tau| \geq T_A \end{cases} \tag{3.75}$$

Case 3. HF

$$R_{y_f y_f}(\tau) = \begin{cases} R_{d_F}[0]R_m[0]\left(1 - \frac{|\tau|}{T_F}\right)\left(C_p(\tau)\right) & \text{if } |\tau| \leq T_F \\ 0 \quad \text{if otherwise} \end{cases} \tag{3.76}$$

Case 4. AMS

$$R_{y_m y_m}(\tau) = \begin{cases} R_{d_A}[0]R_m[0]\left(1 - \frac{|\tau|}{T_A}\right)\left(C_p(\tau)\right) & \text{if } |\tau| \leq T_A \\ 0 \quad \text{if otherwise} \end{cases} \tag{3.77}$$

Given the autocorrelation functions of the outputs of each algorithm, the corresponding power spectral density functions are easily obtained through numerical

integration. These can then be compared to the power spectral density of white noise. A similar power spectral density would be ideal for the scrambled signal at the output of a speech scrambling algorithm, since it would hide the frequency characteristics of the input speech signal, as well as any frequency manipulations performed on this signal. Of course, the power spectral density of white noise can never be attained since this would imply that the speech scrambling algorithm provides infinite bandwidth, and hence infinite power.

Having the functions for the autocorrelations of the outputs of the various analog speech scrambling algorithms, the final objective is to find the best algorithm, and the optimal configuration of this algorithm, by finding the best possible combination of number of PAM levels for the PAM process(es) in this algorithm. The remaining criteria in this case, defining the best algorithm optimally configured, is that this algorithm

1) provides the least residual intelligibility at the output of the transmitter by resembling white noise as much as possible, and

2) offers maximum cryptanalytical strength as a result of offering the largest number of possible transformations for the incoming signal.

The first comparison is carried out between the two two-dimensional algorithms. Since the number of levels of a particular PAM process can range from 2 to 16, the two two-dimensional analog speech scrambling algorithms are compared for the four extreme cases of $(k_F,k_A) = (2,2), (2,16), (16,2)$, and $(16,16)$ as shown in figs. 3.6, 3.7, 3.8, and 3.9 respectively, while keeping the actual values of the levels in each case the same between the two algorithms. Furthermore, the autocorrelation functions are normalized by the average power of the output scrambled signals in each case accordingly. As a result, it is found that in all of the four cases outlined above, the statistics of the HF-AMS algorithm approximate those of white noise more closely than do the statistics of the HF-AAS algorithm. In fact the same thing is found to apply irrespective of the actual PAM levels used in each case. The HF-AMS algorithm is thus chosen as the better of the two

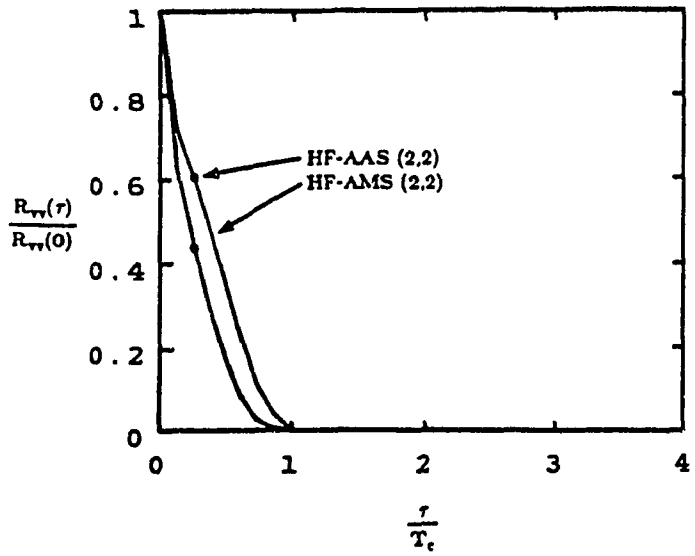

Figure 3.6 Autocorrelation comparison of the two-dimensional algorithms using (2,2).

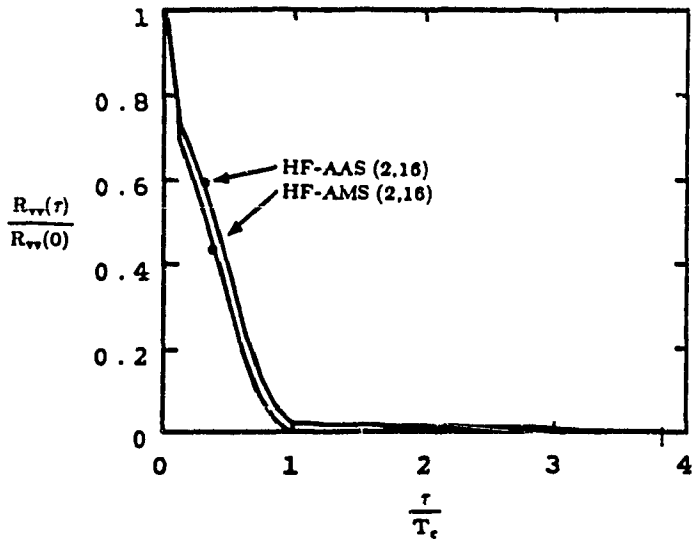

Figure 3.7 Autocorrelation comparison of the two-dimensional algorithms using (2,16).

two-dimensional analog speech scrambling algorithms compared.

Next, the various possible combinations of the number of levels for the AMS algorithm chosen above are compared. Keeping constant the number of levels in the seven identical PAM processes representing the HF algorithm, while varying the levels of the PAM process representing the AMS algorithm, results in the plots of figs. 3.10, 3.11,

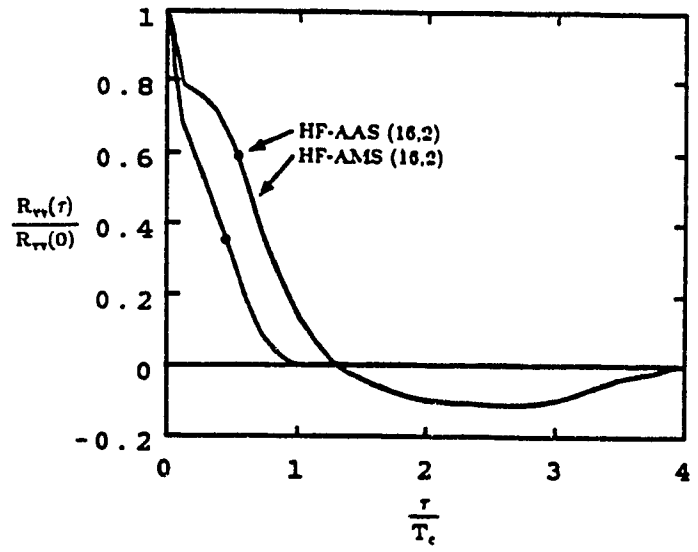

Figure 3.8 Autocorrelation comparison of the two-dimensional algorithms using (16,2).

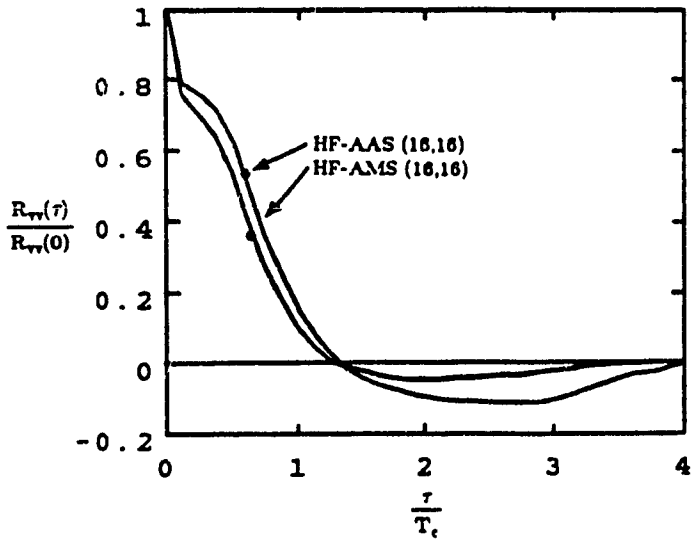

Figure 3.9 Autocorrelation comparison of the two-dimensional algorithms using (16,16).

3.12, and 3.13. In any case, it is seen that the autocorrelation of the output scrambled speech signal deviates more and more from those of white noise, as the number of levels of the PAM process representing the AMS algorithm increase from 2 to 16. Furthermore, this two-dimensional analog speech scrambling algorithm is such, that the plots of (k_F,k_A) are identical to those of (k_A,k_F). As a result, keeping constant the number of

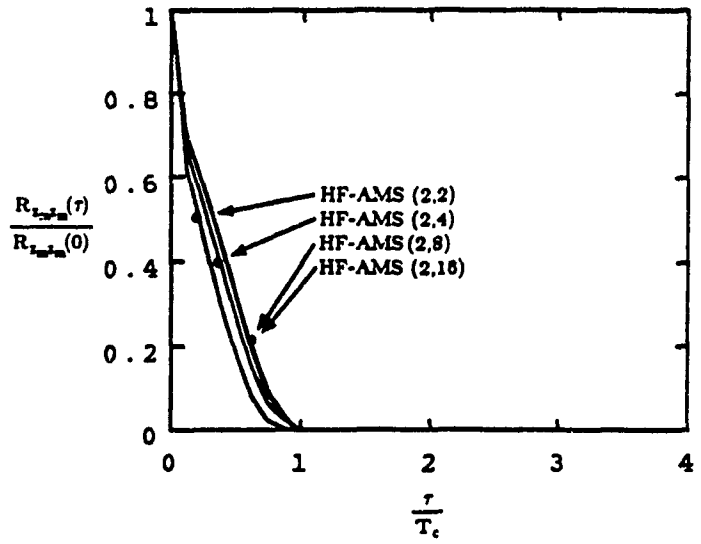

Figure 3.10 Autocorrelation of the HF-AMS algorithm using (2,2), (2,4), (2,8), and (2,16).

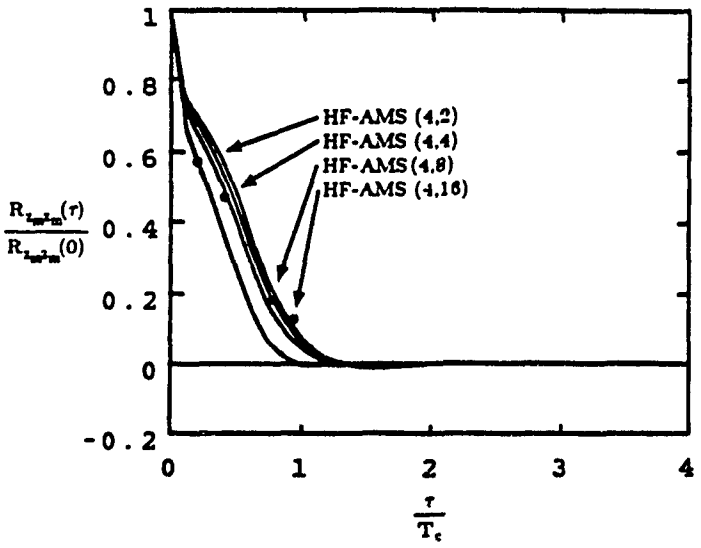

Figure 3.11 Autocorrelation of the HF-AMS algorithm using (4,2), (4,4), (4,8), and (4,16).

levels of the PAM process representing the AMS algorithm, and varying the number of levels, from 2 to 16, in the seven identical PAM processes representing the HF algorithm, results in the statistics of the output scrambled signal deviating more and more from those of white noise.

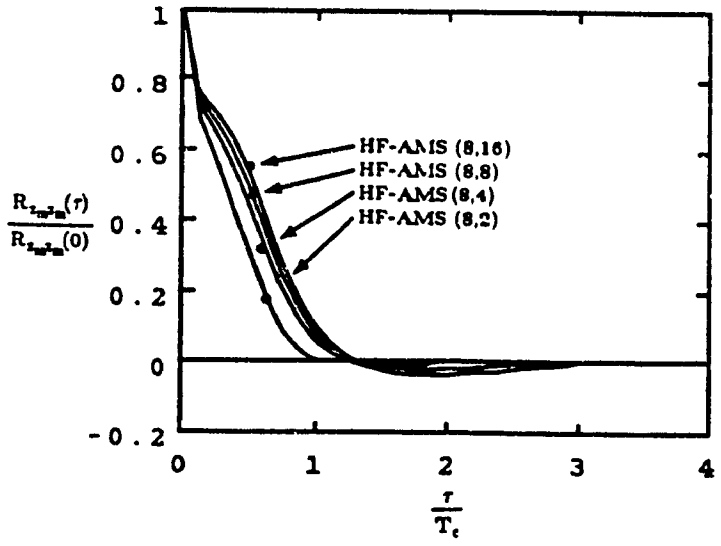

Figure 3.12 Autocorrelation of the HF-AMS algorithm using (8,2), (8,4), (8,8), and (8,16).

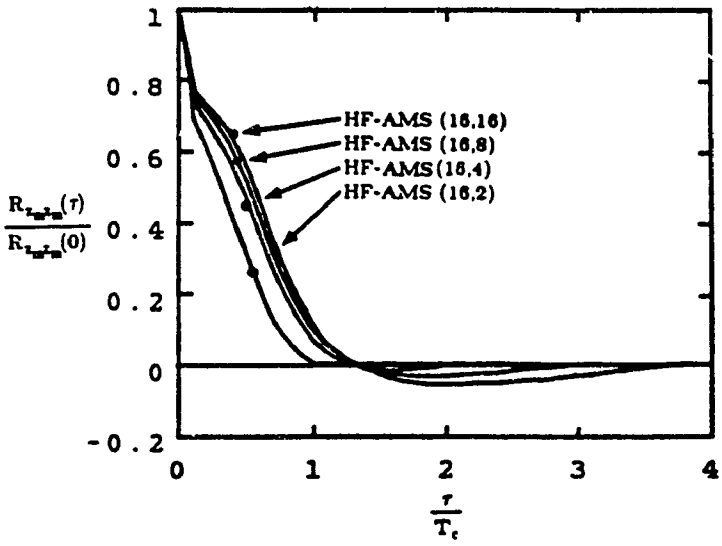

Figure 3.13 Autocorrelation of the HF-AMS algorithm using (16,2), (16,4), (16,8), and (16,16).

The above results, concerning the two-dimensional HF-AMS algorithm can be summarized by a directed graph (figure 3.14), where each combination is a node, and each directed edge joins two combinations pointing in the direction of the worst combination (i.e. combination which results in statistics furthest from those of white noise). From the directed graph it is seen that the statistics of the combinations (2,16) or (16,2) and

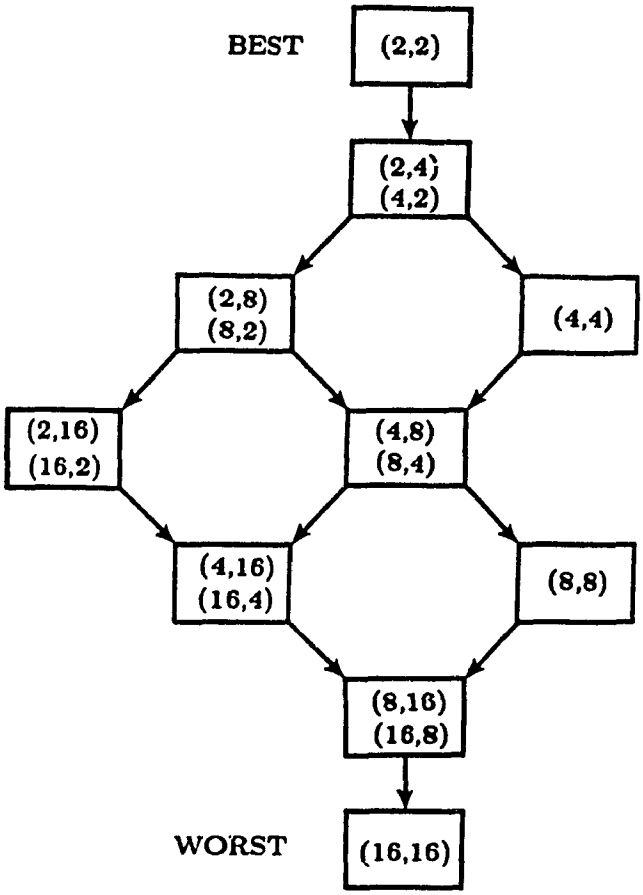

Figure 3.14 Combinations resulting in best to worst statistics

(4,4) have to be further compared, and the statistics of the combinations (4,16) or (16,4) and (8,8) have to be further compared. These comparisons are shown in figs. 3.15, and 3.16 respectively. As a result, two more edges are added to the directed graph of figure 3.14, and redundant edges are removed resulting in the directed graph of figure 3.17. This shows all the possible distinct combinations for the number of levels of the PAM processes representing the HF-AMS algorithm, in the direction from best to worst, as governed by how the statistics of each compares to those of white noise.

Another arrangement of the possible combinations of figure 3.17 is from best to worst, with respect to the number of combined possibilities of hopping filters and amplitude scrambling levels (i.e. the more, offering the largest number of possible

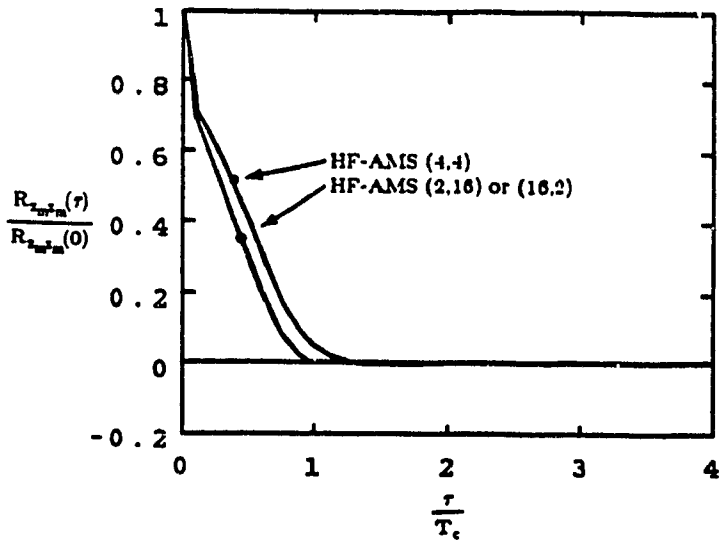

Figure 3.15 Autocorrelation of the HF-AMS algorithm using (2,16), (16,2), and (4,4).

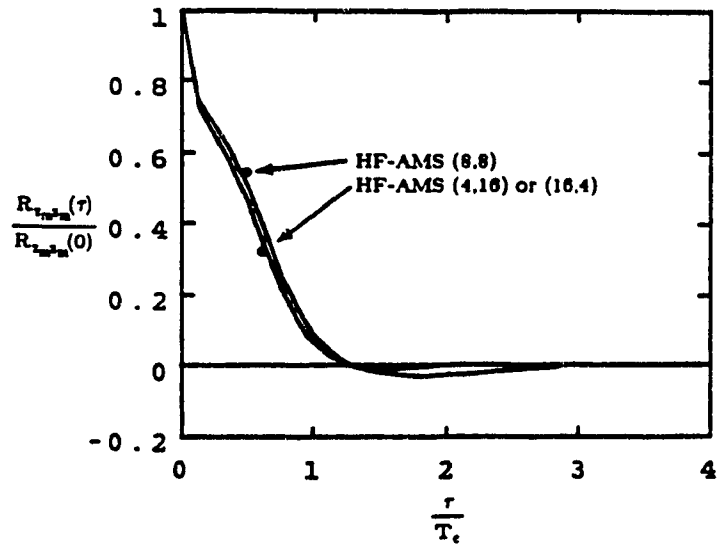

Figure 3.16 Autocorrelation of the HF-AMS algorithm using (4,16), (16,4), and (8,8).

transformations for the incoming speech signal), as shown in figure 3.18. Considering both of these arrangements, the only combinations which are among the top (half) best are (16,2) and (8,2). Since (16,2) is statistically identical to (2,16), (8,2) is statistically identical to (2,8), and from figure 3.10 it is seen that the combinations (2,16) and (2,8) are statistically very close, then (16,2) and (8,2) are also considered statistically very

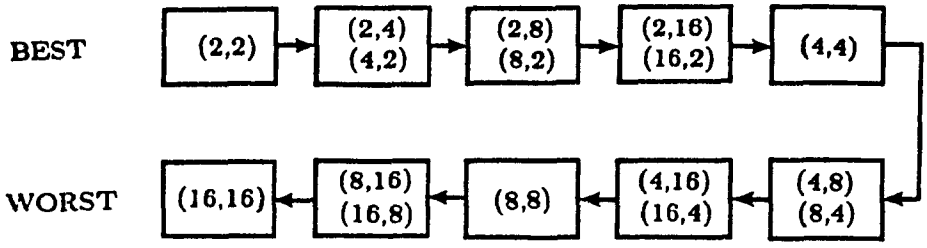

Figure 3.17 Strictly directed graph of best to worst statistics.

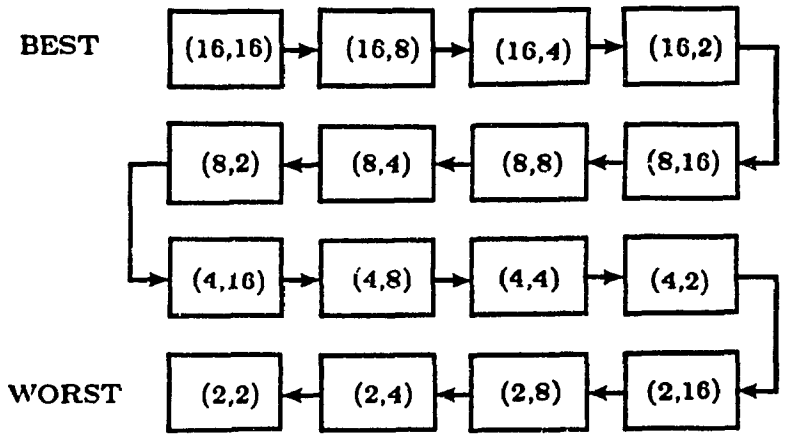

Figure 3.18 Best to worst combinations for minimum to maximum number of possible transformations

close. Furthermore, since the combination (16,2) provides more hopping filter possibilities, this is chosen as the best compromise combination of number of levels to define the two-dimensional analog speech scrambling algorithm already chosen.

In comparing the two one-dimensional analog speech scrambling algorithms (figure 3.19) it is seen that these are statistically identical. On the other hand, the HF algorithm provides more possible hopping filters than the AMS algorithm provides possible levels, for all cases of 2, 4, 8, and 16 PAM process levels. As a result, the HF algorithm is chosen as the best one-dimensional analog speech scrambling algorithm. For this algorithm it is seen (figure 3.19) that the smaller the number of PAM levels the better the

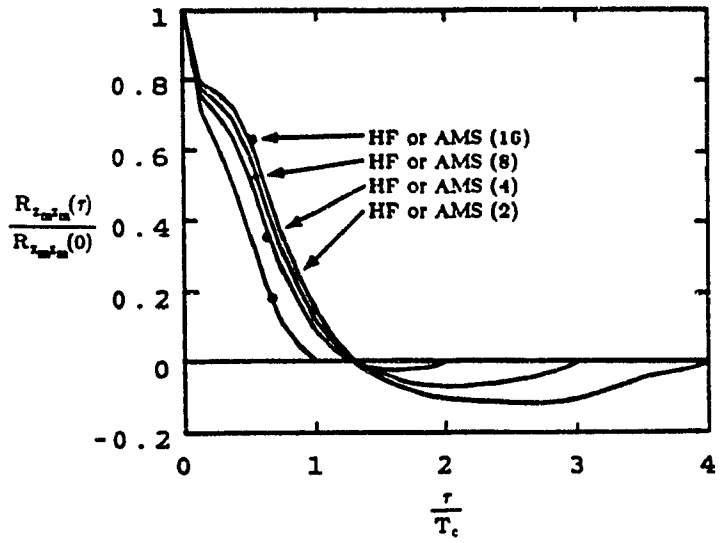

Figure 3.19 Autocorrelation comparison of the one-dimensional algorithms using (2), (4), (8), and (16).

statistics of its output speech scrambled signal. On the other hand, it is known that the larger the number of PAM levels the more the available hopping filters.

Finally, in comparing the HF-AMS algorithm using the previously chosen combination (16,2), with the HF algorithm alone using any possible number of PAM levels, it is seen that the two-dimensional analog speech scrambling algorithm has better statistics than the best statistics possible (ie 2 PAM levels), with the HF algorithm alone, as shown in figure 3.20. Furthermore, the two-dimensional analog speech scrambling algorithm with the chosen combination provides more combined possibilities of hopping filters, and amplitude scrambling levels, than the one-dimensional analog speech scrambling algorithm provides hopping filters with the maximum number of possible PAM levels (i.e. 16). As a result, the two-dimensional analog speech scrambling algorithm involving hopping filters with 16 PAM levels, and amplitude (multiplication) scrambling with 2 PAM levels is found to be the best algorithm from those examined, providing a compromise between the desired statistics of its output scrambled signal, and the maximum number of possible hopping filters and amplitude scrambling level combinations.

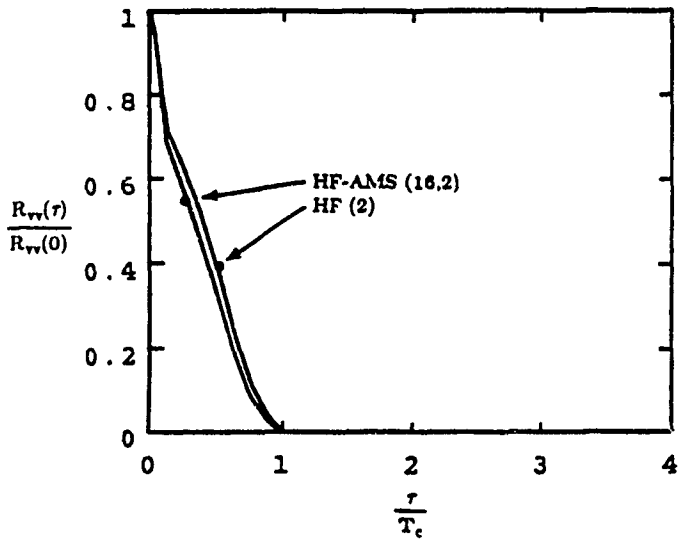

Figure 3.20 Autocorrelation comparison of the two-dimensional HF-AMS algorithm using (16,2) and the one-dimensional HF algorithm using (2).

3.4 Concluding Remarks

The security of various analog speech scrambling algorithms has been evaluated, based on how statistically uncorrelated the output of each scrambling algorithm was to the input, approaching perfect secrecy at best, on the degree to which the statistics of the output of each scrambling algorithm resembled those of white noise, attaining least residual intelligibility at best, and furthermore, on the number of possible transformations offered for the incoming speech signal, maximum at best.

As a result, based on the criteria outlined above, a new hybrid hopping filters/random amplitude multiplication scrambling (HF-AMS) algorithm was chosen as most secure. The security of this algorithm is valid only under certain constraints on the PAM processes involved. This algorithm should be operated with 16 PAM levels for each of the seven PAM processes representing the hopping filters algorithm (i.e. leading to 268 435 456 possible hopped filters), and 2 PAM levels for the PAM process representing the amplitude multiplication scrambling algorithm. This implies hopping between

filters every 1.25 ms, and changing linear amplitude multiplication levels possibly every 0.3125 ms. Further constraints on the 8 PAM processes require that these are uniformly distributed, statistically independent of each other, and have zero mean. Assuming that every PAM process is uniformly distributed, the constraint of zero mean is satisfied if the levels of these eight PAM processes are chosen, such that for every PAM level in a particular process the additive inverse of that level exists within the same PAM process, and the multiplicative inverse of that level exists within the corresponding descrambling PAM process. The actual values of these levels is not important, but should respect the general specifications outlined for the telephone system. The remaining two constraints for the 8 PAM process, namely uniform distribution within the levels of a particular PAM process and statistical independence between these PAM processes, are only assumed in this chapter, and will be supported in the following chapters to come.

CHAPTER IV

SECURITY EVALUATION OF A NEW DES-BASED KEYSTREAM GENERATOR

4.1 INTRODUCTION

An analog speech scrambling algorithm alone simply dictates a general procedure to be followed for scrambling speech. Such an algorithm is not efficient if it cannot be used by more than one user. Furthermore, if the same algorithm is used by more than one user then security is threatened. To resolve this issue an algorithm is designed in such a way that the general procedure is made public, and can be used by all, while at the same time making it possible to easily configure many instances of this algorithm. Each instance of this algorithm will dictate a particular distinct procedure different from that of all other instances, and identified by a key, the key used to configure the particular instance of the algorithm in the first place. The key referred to here not only tailors the algorithm for a particular user, but allows this user to change the configuration of the algorithm at will, creating a random operation of this algorithm, as random as the choice of key. This idea of using a key to manipulate the operation of a particular speech scrambling algorithm stems from cryptography, where keys are used in block or stream mode to manipulate the operation of data cipher systems. In fact, the particular methods of generating and applying keys to data cipher systems can be directly carried over to speech scrambling systems. As a result, most, if not all speech scrambling systems make use of digital keys in block mode, stream mode, or a combination of both as will be seen in this chapter.

Stream mode consists of generating and applying a sequence of bits referred to as a keystream, bit by bit, to the input, be it data or speech. In this mode, knowing a small portion of the input, as well as the corresponding output, readily allows for obtaining a

small portion of the keystream, proportional in length to that of the known input. Since the keystream is continuously being generated and hence continuously changing, the cryptanalyst's focus in this mode is to be able to predict a larger portion, or the entire period at best, of the generated keystream from a relatively short known portion of this keystream.

Shift registers are the most commonly used means of generating keystreams, but not all configurations of shift registers satisfy the requirements for a secure keystream generator as outlined in chapter II. One very common such generator is the linear feedback shift register of length n, with primitive characteristic polynomial. All sequences produced from such a generator are known to have the maximum length of 2^n-1, as well as to satisfy Golomb's three randomness postulates also outlined in chapter II. The only drawback with such a generator is its linear equivalence. A linear feedback shift register of length n has a linear equivalence of n. This implies that upon observing 2n-1 elements of this sequence, the linear feedback shift register configuration can be reconstructed so as to generate the rest of the elements. Thus, a relatively short portion of the sequence allows for generation of the entire sequence. This is very undesirable in terms of security. As shown in [11], this drawback concerning the linear equivalence of a linear feedback shift register can be remedied by introducing nonlinear logic in the feed-forward path of such a shift register. In this way the linear equivalence can be made to be maximum, or 2^n-1, while also maintaining a sequence length of 2^n-1, hence maximum complexity of 1. This configuration will satisfy the first and last requirements for a secure keystream generator, as outlined in chapter II, while affecting the random appearance requirements of the sequence as defined by Golomb's postulates. As shown in [11], this can be remedied by appropriately changing the nonlinear logic configuration in the feed-forward path, so that it not only provides maximum linear equivalence and complexity as it did before, but in addition injects randomness into the generated sequence by imposing on it noiselike characteristics.

Block mode consists of applying an entire fixed length block of bits, representing a key, to an equally long block of input at once. The same key is generally applied to many consecutive blocks of input. In this mode, knowing a few blocks of input as well as their corresponding output blocks does not allow for the key to be readily obtained. Unlike stream mode, the cryptanalyst's focus in this mode is to obtain the key block from known input-output block pairs.

Knowing several input-output block pairs, one way of attempting to obtain the key would be to try and exhaust all possible keys, in order see which is responsible for the transformations within these pairs. This can be made more difficult by increasing the key length, and hence the key space, making the exhaustive key search procedure very time consuming and hence impractical. Two methods of increasing the key size of a well known block cipher, the data encryption standard algorithm, as explained in chapter II, are put forth in [7] and [10].

Knowing several input-output block pairs, another way of attempting to obtain the key would be to collect these pairs forming a code book. Since the general procedure consists of maintaining a fixed key for many input blocks to come, future output blocks would be compared with output blocks already in the code book, obtaining the respective input blocks. This type of attack, as well as the exhaustive key search explained above, can both be stopped by changing the key at a faster rate, to the point where a key is changed with every incoming input block, as shown in figure 2.6b, and as put forth in [10]. The rate of change of the key in this configuration combines the principles of block mode with those of stream mode. With this configuration the cryptanalyst does not focus only on trying to obtain a particular key from a particular input-output block pair, since this is now more difficult to do. In fact, if such a key block is found the cryptanalyst will attempt to predict past and future key blocks relative to the known key block. These attempts will be rendered futile by the two one-way functions and the exclusive-OR operation respectively.

It is suggested in [10] that the two one-way functions can be represented more specifically with the DES algorithm. In this configuration the DES algorithm will have a fixed publicly known plaintext, and its output fed back into the key port. The two DES modules will be initiated with two different secret keys, and then allowed to run continuously. This particular configuration of the DES algorithm can be viewed as a random function, mapping the key space of DES into the same space as put forth in [8]. More specifically, it is shown that irrespective of the fixed plaintext, each and every key out of 80% of all such keys making up the DES key space can possibly map into any other key at random within the same group. This applies to both DES modules leading to the exclusive-OR of their outputs to also be random.

Another point of interest for the configuration of figure 2.6b is the repetition period of the keys at the output of the exclusive-OR. As with any random function mapping a finite set into itself, the function mapping the key space of the DES algorithm into the same space, with fixed plaintext, will eventually lead to periodic repetition of the same output keys. As this occurs with both DES modules in the configuration of figure 2.6b, the output of the exclusive-OR will also eventually lead to periodic repetition. While the period of repetition can at maximum be about 80% of the entire DES key space, the minimum guaranteed period of repetition, which is more important for security purposes, is not known. This period of repetition will depend on the initial input keys and fixed plaintext of the respective DES modules. As a result, the repetition period at the output of the exclusive-OR is also not known.

The objective of this chapter is to combine the stream mode sequence generator of [11], a variation of the block mode DES key scheduling algorithm of [10], and 8 linear sequence generators in order to create a stream mode DES-based keystream generator capable of generating eight practically independent long secure keystreams These will be such so as to satisfy the requirements for a secure keystream generator, as outlined in chapter II. Section 4.2 formally proposes the new DES-based keystream generator. Sec-

tion 4.3 outlines the development of the idea of this keystream generator, showing why it was put together as such. It shows that the decisions regarding the development of this generator were based on how well it satisfied the requirements for a secure keystream generator outlined earlier.

4.2 Proposed DES-Based Keystream Generator

The newly proposed DES-based keystream generator can be divided into three main sections, as shown in figure 4.1. The first section consists of a main multilayer nonlinear feed-forward generator, as described in [11]. The shift registers in this multilayer generator are of length 64, thus producing an output sequence of length 2^{64}-1 or 1.8447×10^{19}. As shown in [11], the minimum number of layers of shift registers of length 64 required for a sequence to have maximum complexity is q, such that $64 \leq 2^q < 2(64)$ is satisfied. This implies $q = 6$. There are thus 6 layers of shift registers alternating between 6 layers of nonlinear logic. Every nonlinear logic stage manipulates parallel input from one shift register, and feeds it serially to the next such register. The output of the last nonlinear logic stage is the output of this section. Each nonlinear logic stage is allowed to be configured by the user, with the constraint that not only maximum complexity is achieved with a minimum number of layers, but that also the resulting sequence has noiselike characteristics. Unlike the others which do not require any configuration, the very first shift register is allowed to be configured and initially loaded by the user, with the constraint that this configuration consists of linear feedback, leading to a maximal length sequence. The configuration of all the nonlinear logic stages, the configuration of the linear feedback shift register, and the initial loading of this register make up one part of the secret key of the entire keystream generator proposed [31].

The second section of the keystream generator is made up of a variation of the DES key scheduling algorithm put forth in [10]. It consists of a sliding window 64-bit serial to parallel converter, which produces a 64-bit word for every input bit resulting

from the first section. Each 64-bit word is then fed to the data port of an instance of the DES algorithm, as well as to the data port of an instance of the inverse of the DES algorithm. These algorithms are configured so that their outputs are fed back to their respective key ports, with the initial input to each key port being the all zero key. Furthermore, the outputs of both these algorithms are exclusive OR'ed, resulting in the final output of this section. For each incoming bit to this section there is one 64-bit word being produced. Since the input sequence to this section has a period of 2^{64}-1 this section will produce 2^{64}-1 64-bit words per cycle [31].

The third section of the proposed keystream generator consists of eight subordinate linear feedback shift registers, each of length 8, and operating in parallel. These are allowed to be configured by the user, with the constraint that each produces a distinct maximal length sequence. This configuration of all the subordinate linear feedback shift registers make up the remaining parts of the secret key of the entire keystream generator proposed. All the 8-bit shift registers are parallelly initialized by consecutive 8-bit sequences, making up one 64-bit sequence. Each 64-bit sequence is the result of the second section of the proposed keystream generator. It is possible that the 8-bit initialization sequence can lead to a particular subordinate linear feedback shift register having the all zero state. This is avoided by feeding the eight bits of the initialization sequence into a NOR gate. The output of the NOR gate, as well as the most significant bit of the initialization sequence are then fed through an OR gate, whose output is the input to the most significant bit position in the respective shift register. Finally, every second period of 255 bits for each of the eight output sequences, resulting from each of the eight subordinate linear feedback shift registers, is complemented. The output of this section, which is also the final output of the proposed keystream generator, produces eight (2^{64}-1) by (2^{8}-1) or 4.7039×10^{21}-bit sequences, which will be shown to satisfy the requirements of secure sequences [31]. Furthermore, it should be pointed out that seven of the eight keystreams produced are passed through seven corresponding 4-bit S/P converters,

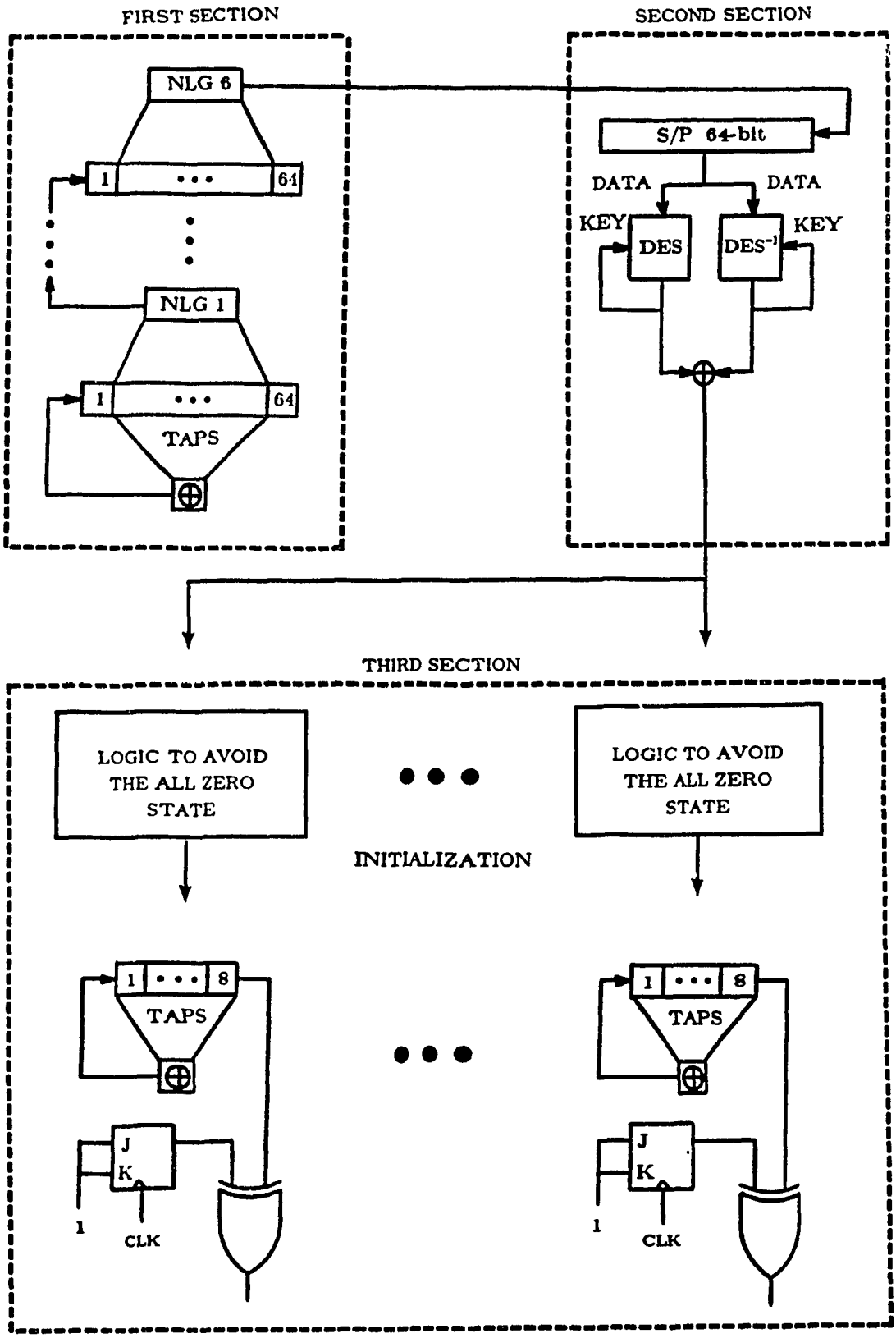

Figure 4.1 Proposed DES-based keystream generator.

whose outputs are used for the control of the hopping filters algorithm. The last keystream is used as is, to control the amplitude multiplication scrambling algorithm, while the same keystream passed through a 2-bit S/P converter is used as the control signal of a 4x4 multiplexer in the timing circuit of the transmitter unit. Similarly for the receiving unit.

4.3 Security Evaluation Of Proposed Keystream Generator

Having finally proposed the scheme of the new DES-based keystream generator, it is appropriate to outline the development process that led to this scheme, justifying the presence of its major components. This development process was based on certain requirements outlined in the theory of chapter II, as well as on actual simulations of the scheme proposed and variations thereof, where necessary.

Ideally, simulation of a particular scheme would require the generation of one period for each of the eight keystreams, repeated for all secret keys applied to this generator. Due to limitations of time and computing resources, simulation was selective and scaled down, but justified as adequate nonetheless, to yield conclusive results. Of all the possible secret keys that can be applied to the generator, yielding many possible configurations of this generator, any one secret key can be used in the simulation as long as it conforms to the design specifications. The secret key should be such, that the linear feedback shift register within the multilayer generator produces a maximal length sequence. Furthermore, this secret key should be such, so as to configure the nonlinear logic at every layer of the main multilayer generator appropriately, in order to produce a sequence with noiselike characteristics and maximum linear complexity at the output of the first section of this generator. Finally, the secret key should be such, so as to ensure that the eight subordinate generators are distinctly configured to produce maximal length sequences. Although every different key will produce a different set of eight keystreams, the statistics (i.e. autocorrelation and cross-correlation) of various sets of

keystreams will be more or less the same, as long as their respective keys follow the same design specifications, and use the same structure of the keystream generator algorithm. Thus, without loss of generality, a particular key will be selected for the simulations. This key will result in eight keystreams each of length 4.7039×10^{21} bits. Since working with such lengths is impractical considering the available computing resources, the keystream generator was scaled down. This was accomplished by scaling down the main multilayer generator from a length of 64 to a length of 8. This results in the length of each of the eight keystreams being 65 025 bits, as opposed to 4.7039×10^{21} bits. The simulations are carried out for the purpose of performing certain statistical tests on their results. The design of this keystream generator algorithm is such, that the results of the statistical tests can only improve (i.e. from the point of view of the designer) with length. It thus suffices to show that the required statistical properties of the scaled down versions of the eight keystreams generated through simulation are satisfied.

The original idea for the keystream generator involved the use of eight linear feedback shift register generators, each of length 64, configured to produce eight distinct maximal length sequences of length $2^{64}-1$, or 1.8447×10^{19} bits. The drawback of this scheme is that it produces sequences which do not have a large linear equivalence, as is required for security. The above scheme would result in eight keystreams each of linear equivalence equal to the length of the shift registers, thus 64. As compared to the lengths of the sequences that this scheme would produce, the linear equivalence mentioned would result in a complexity (which ranges from 0 to 1) of 3.4694×10^{-18}. To resolve this issue the eight linear feedback shift registers were replaced by eight multilayer nonlinear generators [11], as described in [32], but with shift registers of length 64, producing sequences of length 1.8447×10^{19} bits. By choosing the nonlinear logic appropriately, the resulting sequences, in addition to having maximum length, each have maximum linear equivalence equal to the length of the sequence, resulting in maximum complexity of 1. Furthermore, the configuration of nonlinear logic was responsible for

imposing noiselike characteristics on the output sequences, and doing all this with the minimum number of required layers, namely 6. Before formally examining this scheme for satisfaction of the requirements for a secure keystream generator, as outlined in chapter II, it was realized that this would not be practical to operate, in the sense that it requires a large amount of key material. This key material would be about 8 times that of the proposed keystream generator, and a burden to the user. To resolve this, a compromise was reached between the use of eight linear feedback shift registers, and the use of eight multilayer nonlinear generators. It was decided to use one main multilayer nonlinear generator of shift register length 64, and 8 subordinate linear feedback shift register generators each of length 8 and maximal length configuration, in such a way that every 64-bit sliding window sequence, produced by the main generator, would be used to initialize simultaneously all 8 linear feedback shift register generators. Using this scheme, the main multilayer nonlinear generator will supply a large linear equivalence (i.e. increased security) for all eight generated keystreams, with a reduced amount of key material. Furthermore, the eight linear feedback shift registers will provide a means to produce 8 keystreams from the single keystream of the multilayer generator. Each of the linear feedback shift registers will allow for its unique configuration from the others, hence ensuring a unique keystream. Finally, each 255-bit sequence produced from a particular initialization of a particular linear feedback shift register will be a maximal length sequence. This will allow for the autocorrelation of a particular keystream (at least within a 255-bit shift) to resemble that of a maximal length sequence, a desired property for ensuring randomness as outlined in chapter II.

As a result of initializing the subordinate generators periodically for every 255 bits produced by these generators, a drawback to this scheme results. This is the great dependency between the 8-bit initialization sequences of one subordinate generator, and the same generator one 255-bit period later, one being a shifted replica of the other with the addition of one bit. This produces undesirable results for the autocorrelation

(beyond the 255-bit shift) of the particular keystream resulting from this generator, hence affecting the randomness properties of this keystream. Without loss of generality, a scaled down version of this keystream generator scheme was simulated, specifically for one subordinate linear feedback generator, using the secret key SK1 shown in figure 4.2.

```
MAIN GENERATOR
REGISTER LENGTH
8
SPANS
1 1 4 2 3 2 4 3
1 1 3 4 2 3 2 4
4 1 1 3 4 2 3 2
TAPS
0 1 1 1 0 0 0 1
INITIAL CONDITION
1 0 1 0 1 0 0 1
SUB GENERATOR
3
REGISTER LENGTH
8
TAPS
1 0 1 1 0 1 0 1
```

Figure 4.2 Key material SK1 used in simulation.

The autocorrelation of the keystream KS1 produced as a result of this simulation is as shown in figure 4.3. The dependency mentioned earlier, between consecutive initialization sequences of a particular subordinate generator from one 255-bit period to the next, shows up in the autocorrelation plot as 'spikes' spaced about one 255-bit period away from each other. The height of these 'spikes' diminish to about 0 when half of the original initiating sequence has been shifted away from the particular subordinate generator. This occurs after four 255-bit periods have elapsed. This scheme thus violates one of the three requirements for randomness, requiring that the out-of-phase autocorrelation of the generated keystream be constant. To remedy this violation it was thought appropriate to add a module between the main multilayer nonlinear generator and the eight subordinate linear feedback generators, which would dissolve this dependency between consecutive initialization sequences of a particular subordinate linear feedback generator. This

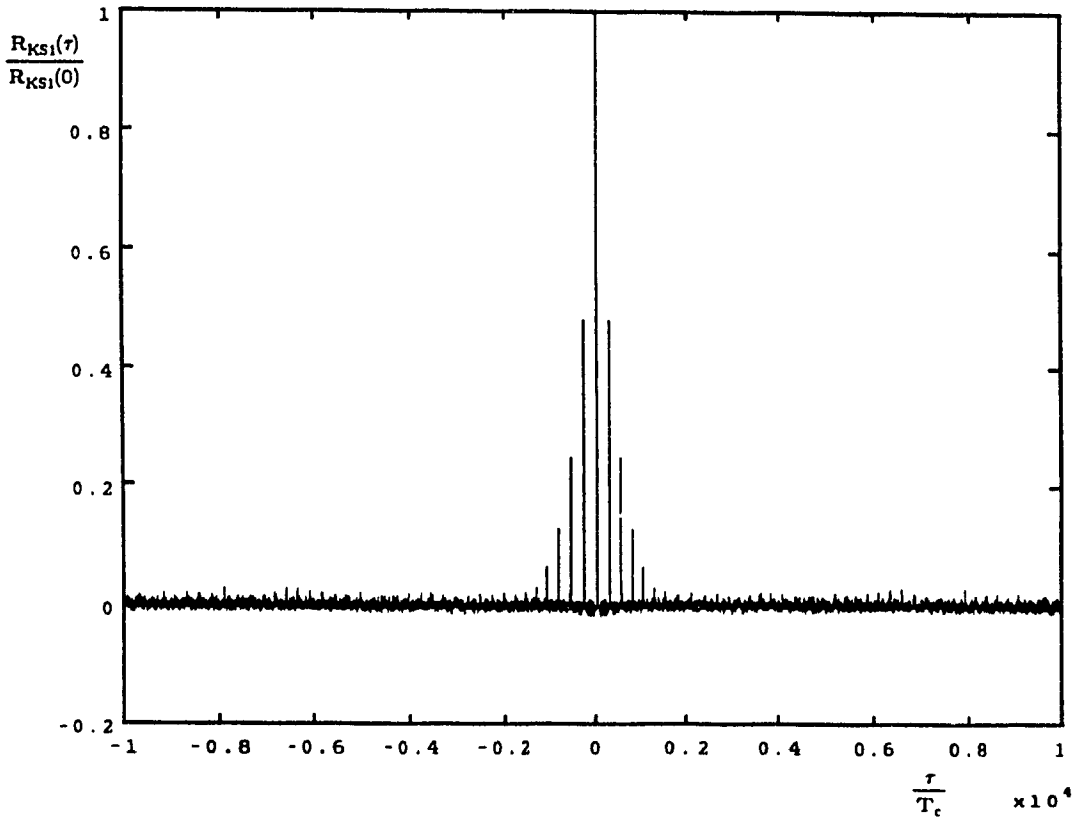

Figure 4.3 Autocorrelation of keystream KS1 resulting without the use of the DES-based key scheduling scheme.

module introduced here is none other than a variation of the DES key scheduling algorithm introduced in [10]. The variation consists of using the DES algorithm and its inverse, as opposed to two instances of DES. As a consequence, the key ports of both DES and DES^{-1} can be initialized with the all zero sequence or key, as opposed to two different secret keys. Finally, the data ports of both DES and DES^{-1} do not contain a fixed sequence, but instead, sliding window 64-bit sequences produced by the main multilayer nonlinear generator. Besides dissolving the dependency between consecutive initializing sequences of a particular subordinate linear feedback generator, the variation of the DES key scheduling algorithm of [10] also provides a one-way function, between the sliding window 64-bit sequences produced by the multilayer nonlinear generator, and the

corresponding 64-bit sequences used to initialize the eight subordinate linear feedback generators. This enhances security by discouraging a cryptanalyst from attempting to work backwards. As explained in [10], having two one-way functions tied with an exclusive-OR is far more secure than only one one-way function. Furthermore, changing one of the DES modules in the DES key scheduling algorithm of [10], to a DES^{-1} module, relaxes the requirement of needing to initialize the key ports of those modules with two different 64-bit sequences. At this point, a scaled down version of the improved keystream generator scheme was simulated, specifically for the same subordinate linear feedback generator as before, using the same secret key depicted in figure 4.2. The autocorrelation of the keystream KS1' produced as a result of this simulation is as shown in figure 4.4. It is seen that the introduction of the module between the main generator and the eight subordinate generators is responsible for removing the undesirable 'spikes', or in other words breaking the dependency between the initialization sequences of a particular subordinate generator. One other requirement for the randomness of a particular keystream as outlined in chapter II is that the number of ones is one more or one less than the number of zeros. In the case of the above keystream generator, a particular keystream out of the eight generated can be divided into 1.8447×10^{19} 255-bit sequences, such that each 255-bit sequence is a maximal length sequence. Maximal length sequences have well known properties one of which is that they always contain one more one than zeros. As a result, the sequence resulting from a particular subordinate generator in this case will contain 1.8447×10^{19} more ones than zeros, thus violating one of the requirements for randomness. To remedy this violation it was decided to complement every second 255-bit sequence of bits being generated. This way every second 255-bit sequence will consist of one more zero than ones. Since there is an odd number of 255-bit sequences making up the entire generated sequence, the resulting generated keystream will consist of one more one than zeros, thus satisfying the requirement.

Having traced the development process of the keystream generator, and having

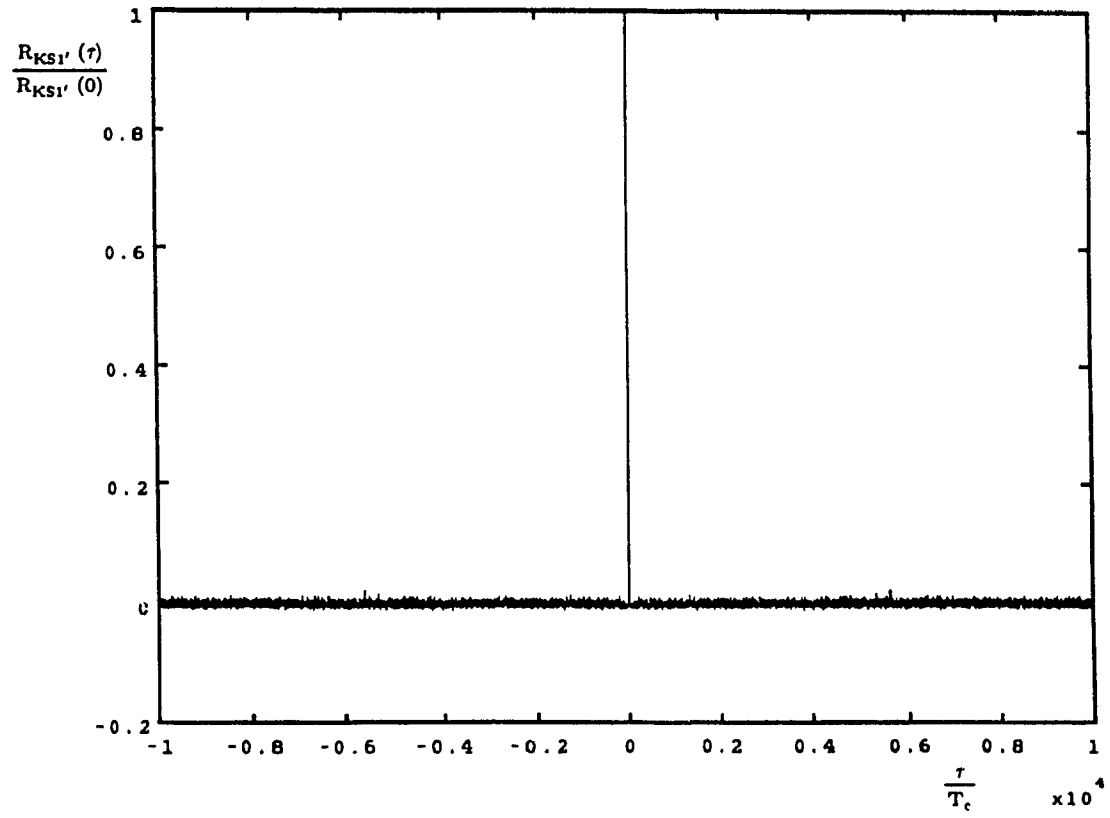

Figure 4.4 Autocorrelation of keystream KS1' resulting with the use of the DES-based key scheduling scheme.

reached the particular scheme proposed in section 4.1, it is appropriate at this point to examine the extent to which this scheme satisfies all the requirements, as specified in chapter II, for a secure keystream generator. The requirement for the number of ones being one more or one less than the number of zeros has been satisfied for this scheme as shown previously.

Another requirement for randomness specifies the fraction of the number of runs of both ones and zeros that should occur for the various run lengths, as well as the relationship between the number of runs of ones and the number of runs of zeros, for each particular run length. Upon simulation of a scaled down version of the above keystream generator for a particular keystream, using the key depicted in figure 4.2, it was found

that the runs requirement is somewhat violated as shown in tables 4.1 and 4.2.

Table 4.1 Consistency of runs (of both ones and zeros) expected and obtained.

Length	Runs		
	Contained	Least Required	Expected
1	16 337	4	16 324
2	8 158	8	8 162
3	4 084	16	4 081
4	2 025	32	2 041
5	1 020	64	1 020
6	512	128	510
7	256	256	255
8	251	512	Not Applicable
9	4	1 024	Not Applicable
10	0	2 048	Not Applicable
11	0	4 096	Not Applicable
12	1	8 192	Not Applicable

Table 4.2 Breakdown of number of ones to number of zeros.

Length	Runs	
	Ones	Zeros
1	8 172	8 165
2	4 077	4 081
3	2 036	2 048
4	1 015	1 010
5	513	507
6	254	258
7	128	128
8	128	123
9	1	3
10	0	0
11	0	0
12	0	1

The runs requirement to be satisfied happens to be one of the well known properties of maximal length sequences. The irony in this case is that the keystreams generated are piece-wise made up of 255-bit maximal length sequences concatenated together. The individual pieces of a keystream thus satisfy the runs requirement, while the entire sequence as a whole does not. This is a result of common, ending and starting bits between consecutive 255-bit sequences, thus allowing runs to overlap between these sequences. Furthermore, runs of length greater or equal to two might be divided

between the beginning and end of a particular 255-bit sequence, thus creating two runs of shorter length from one larger run. These possible discrepancies at the beginning and end of a particular 255-bit sequence force the consistency of runs of this sequence to deviate from those of a maximal length sequence. As a result, the consistency of runs for the entire keystream deviates from the required consistency as outlined in chapter II, but not much.

The final requirement for randomness is a constant out-of-phase autocorrelation. Without loss of generality, a scaled down version of the proposed scheme was simulated (see Appendix A) using key SK1 depicted in figure 4.2. The autocorrelation of the keystream KS1" produced as a result of this simulation is as shown in figure 4.5a, with a more close-up view shown in figure 4.5b. From this it is seen that the constant out-of-phase autocorrelation requirement is satisfied. Because this keystream generator produces eight keystreams to be used simultaneously, an additional requirement related to autocorrelation that should be imposed on the eight keystreams is that of constant cross-correlation between them. Simulating the scaled down version of the adjacent keystream to the one already produced by using the secret key SK2 depicted in figure 4.6, and cross-correlating this keystream KS2" with the one already obtained, results in the cross-correlation depicted in figure 4.7. This is practically constant and zero, hence satisfying the additional requirement, and declaring keystreams produced from this keystream generator as random.

The next requirement to be satisfied is that the keystreams produced from the proposed keystream generator have a guaranteed minimum length. The first section of the proposed keystream generator scheme is guaranteed to produce a sequence of maximum length, as a result of the linear feedback shift register configuration at its base. This sequence will then be 2^{64}-1 or 1.8447×10^{19} bits long. The second section of the proposed keystream generator is made up of the DES key scheduling module. As mentioned in the introduction, the original configuration of this module as put forth in [10] is such,

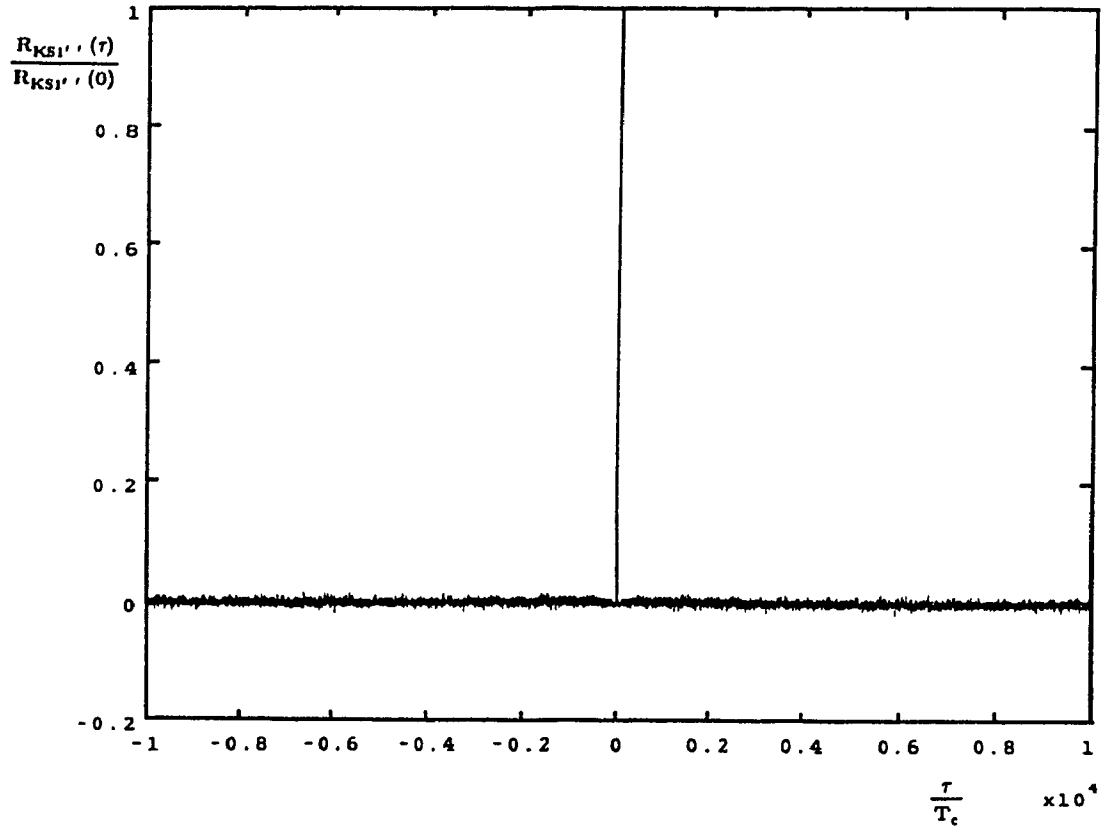

Figure 4.5a Autocorrelation of keystream KS1'' resulting from proposed keystream generator.

that the repetition period is particular to the initial input keys and fixed sequence on the data ports of the two instances of the DES algorithm, and thus cannot be guaranteed in advance. The variation of this module though as put forth in the proposed keystream generator is such, that a repetition is guaranteed to occur whenever the sliding window 64-bit input sequences to the DES, and DES^{-1} algorithms, as well as the initialization of the respective key ports of these two algorithms with the all-zero key begin to repeat. The first of these requirements is ensured by the repetition of the $(2^{64}-1)$-bit sequence at the output of the first section of the proposed keystream generator. On the other hand, the second requirement pertaining to the repetition of the initialization of the key ports, with the all zero key, will have to be forced as such periodically. If the key ports of the

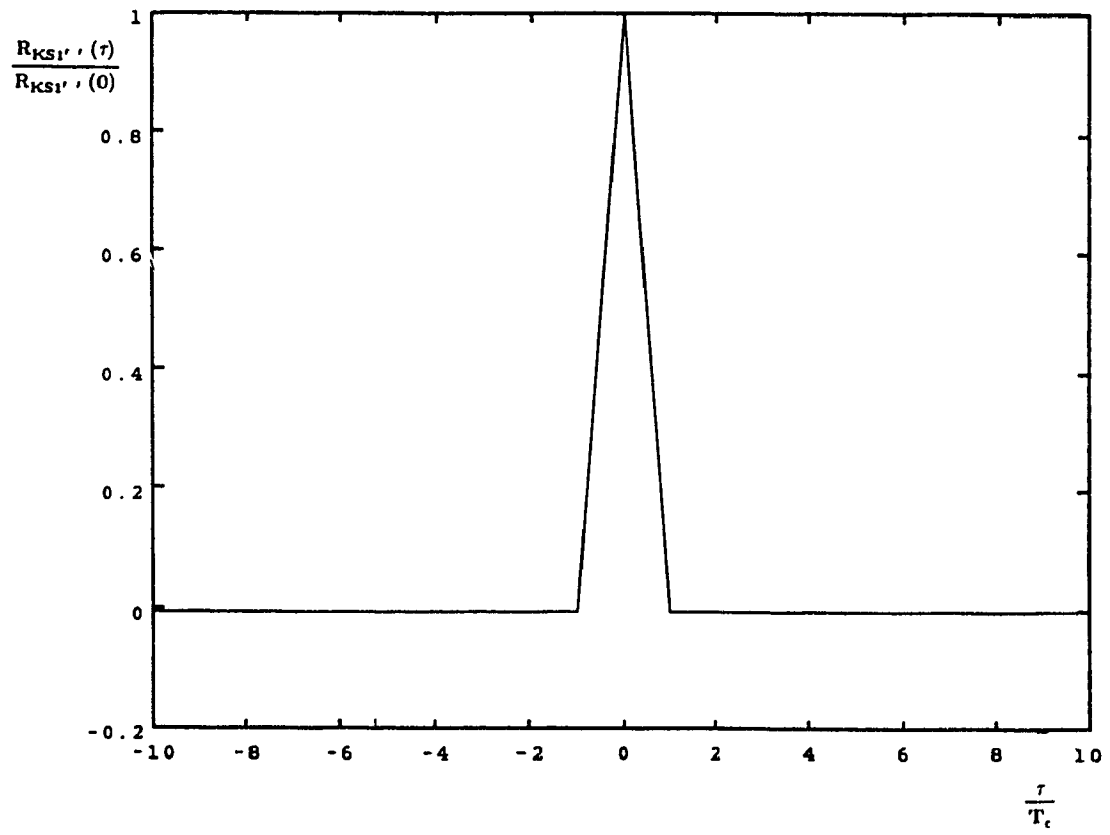

Figure 4.5b Close-up view of in-phase out-of-phase boundary of autocorrelation of KS1''

DES and DES^{-1} algorithms are not initialized periodically as required, then the 64-bit sequences at the output of the second section of the proposed keystream generator may have a period much greater than 2^{64}-1 (a definite advantage), but without any guarantee (a serious disadvantage). Initializing the key ports periodically guarantees 64-bit sequences of minimum repetition period 2^{64}-1, at the output of the second section of the proposed keystream generator. Each of these 64-bit sequences then initializes the eight linear feedback shift registers simultaneously. Logic is provided to ensure that these shift registers are never initialized with the all zero state. These registers being of maximal length configuration, and length eight, each generate a sequence of guaranteed minimum period 2^8-1 or 255. Thus, for each and every initialization sequence out of

```
MAIN GENERATOR
REGISTER LENGTH
8
SPANS
1 1 4 2 3 2 4 3
1 1 3 4 2 3 2 4
4 1 1 3 4 2 3 2
TAPS
0 1 1 1 0 0 0 1
INITIAL CONDITION
1 0 1 0 1 0 0 1
SUB GENERATOR
3
REGISTER LENGTH
8
TAPS
1 0 1 1 0 1 0 1
```

Figure 4.6 Key material SK2 used in simulation.

2^{64}-1 such sequences at the output of the second section of the proposed keystream generator, there will be another sequence generated of period 255. Thus, each of the 8 sequences produced from the proposed keystream generator will have a minimum guaranteed period of $(2^{64}-1)(2^{8}-1)$, or 4.0739×10^{21}, thus satisfying the minimum guaranteed period requirement.

The final requirement for a secure keystream generator is a large linear equivalence. In general, linear equivalence is the least number of memory elements required to generate a sequence in a linear fashion. The linear equivalence of a sequence generated linearly is the length of the shift register generating this sequence, while the linear equivalence of a sequence generated nonlinearly is much larger than the length of the shift register. As shown in [11], the linear equivalence of a sequence from a nonlinear generator can be made to be maximum, equal to the length of the sequence generated. Related to linear equivalence is the number of bits required to define the rest of the sequence. This in general is equal to $\min(2\alpha-1, L)$, where α is the linear equivalence of the sequence, and L is its length. A large linear linear equivalence would then imply the requirement of a large number of bits to define the remainder of a sequence, with the

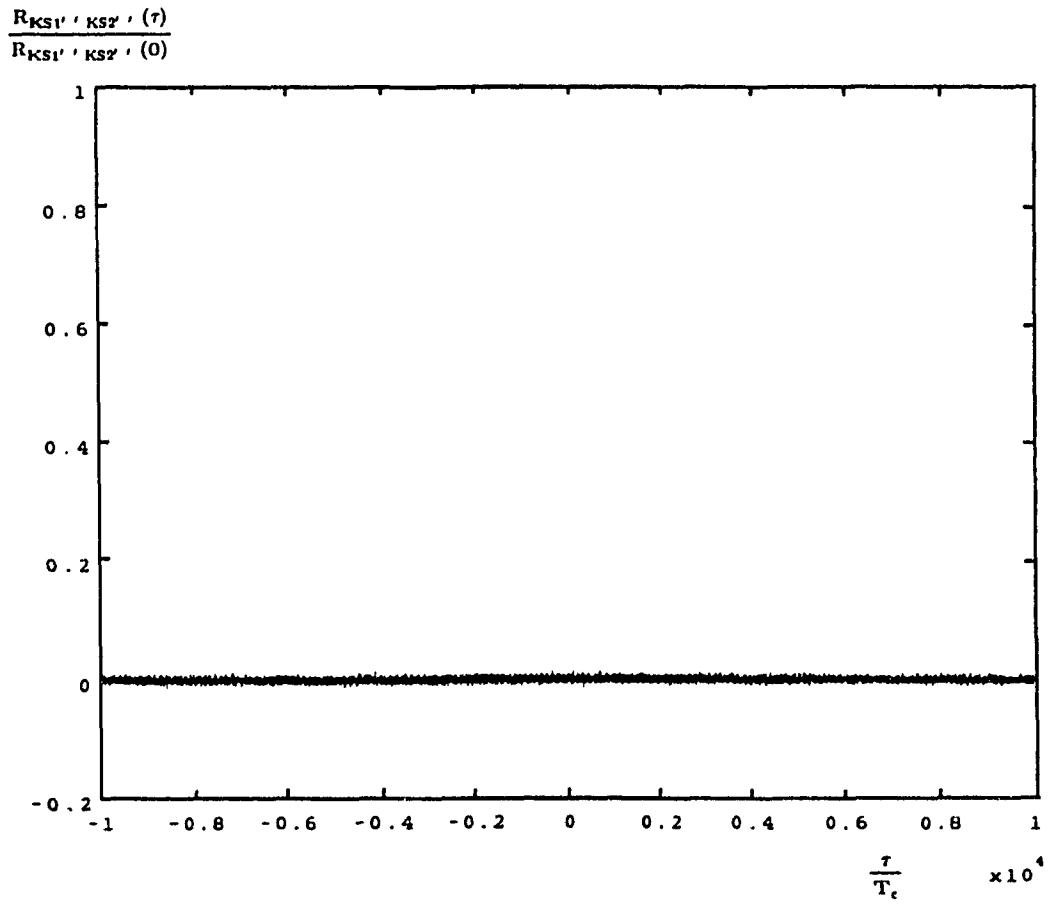

Figure 4.7 Cross-correlation of keystreams KS1'' and KS2'' produced from proposed keystream generator.

upper bound being all bits within one period of the sequence. Because of the direct relationship between linear equivalence and the number of bits required to define the remainder of a sequence, the proposed keystream generator is evaluated with respect to the latter. It will be shown that the number of bits required to define the remainder of a sequence, generated by the proposed keystream generator, is practically equal to an entire period of this sequence. The main multilayer nonlinear generator making up the first section of the proposed keystream has maximum linear equivalence. This implies that no fractional part of a sequence resulting from this generator can define the remainder of this sequence. Since this sequence is fed to the second unit of the proposed

keystream generator through a 64-bit sliding window, no group of 64-bit input sequences to the second section can define the remainder of 64-bit input sequences to this section. Because of the exclusive-OR operation leading to the output of the second section of the proposed keystream generator, no output 64-bit sequence from this section or group of such 64-bit sequences can define subsequent output 64-bit sequences from this unit. These are the input sequences to the third section of the proposed keystream generator. Thus, no initialization sequence or group of such sequences can define subsequent initialization sequences for a particular linear feedback shift register. Since the third section consists of linear feedback shift registers of length 8, knowing the first fifteen bits of one of the eight keystreams will allow for the configuration of the respective linear feedback shift register, as well as its initialization sequence to be revealed. This will allow for the next 240 bits of the corresponding keystream to be readily generated. After this part though the initialization sequence will be changed, and the new initialization sequence will not be obtainable from the previous such sequence. As a result the next 255 bits will not be known until this new initialization sequence is generated and hence revealed. Knowing the new initialization sequence or the first eight bits of the next 255-bit sequence will allow for the rest 247 bits of this sequence to be revealed. Similarly, for all following 255-bit sequences. Since a particular keystream is made up of 2^{64}-1 255-bit linearly generated sequences, one would have to know a fraction of each 255-bit sequence to be able to obtain the rest of this 255-bit sequence. Furthermore, for a fraction of a particular 255-bit sequence to be generated, and hence revealed, all previous 255-bit sequences have to be generated. Table 4.3 shows the lengths of the sequences within a particular keystream that can be generated, as compared to the lengths of the sequences required for these generations. Table 4.3 shows that the percentage ratio of required bits to generated bits rises to 99% for the first 97 255-bit sequences. This implies that to generate the first 24 735 bits 99% of these bits are required to have been generated previously. This includes all 96 previous 255-bit sequences, as well as the first eight bits of

Table 4.3 Number of bits required to generate other bits.

Sequence Length		%Ratio of
Required Bits	Generated Bits	Required to Generated
15	255	5.88%
263	510	51.57%
518	765	67.71%
773	1 020	75.78%
1 028	1 275	80.63%
.	.	.
.	.	.
.	.	.
24 488	24 735	99.00%

the 97th 255-bit sequence. Only these latter 8 bits are actually required for the generation of the 97th 255-bit sequence, but these will not be revealed until all previous 96 255-bit sequences have been generated. Extending table 4.3 it can be seen that the last 255-bit sequence of a particular keystream cannot be generated, before all previous 255-bit such sequences, as well as the first eight bits of the last sequence, have been revealed. Since $255(2^{64}-2)+8 \approx 255(2^{64}-1)$, the number of bits required to define the remainder of a keystream, generated by the proposed keystream generator, is practically equal to the entire period of this keystream. Since this would be the resulting effect of maximum linear equivalence, the requirement for a large linear equivalence has thus been satisfied.

4.4 Concluding Remarks

This chapter put forth a DES-based keystream generator operating in stream mode and producing eight keystreams simultaneously instead of one. This generator basically consists of three sections, one operating in block mode placed in between two operating in stream mode. The resulting keystreams of this generator, each consist of the concatenation of $2^{64}-1$ 255-bit maximal length sequences, with zero cross-correlation between them. All 255-bit sequences within a particular keystream are generated with the same maximal length sequence generator, but with different initialization sequences, resulting in different starting points. Without loss of generality a scaled down version of this keys-

tream generator, operated with a particular key, was simulated. This simulation proved helpful, in showing that the proposed keystream generator more or less satisfied all requirements necessary for a secure keystream generator.

CHAPTER V

A NEW ANALOG SPEECH SCRAMBLING DEVICE

5.1 INTRODUCTION

The two previous chapters have put forth two algorithms, one for performing analog speech scrambling, and one for generating pseudorandom keystreams. Both these algorithms were chosen among others in their respective groups, as being most secure, and the security of both has been evaluated on the basis of respective criteria within the respective chapters. The underlying objective of proposing these two algorithms and evaluating their security was so as to eventually put them to work together, making up an analog speech scrambling device. This is accomplished by introducing some additional circuitry for timing and synchronization, as well as a user interface between the user of the device and the algorithms. The goal of this chapter as outlined in section 5.2 is to show how the two algorithms can work together, by evaluating their security as a whole. Furthermore, section 5.3 introduces the timing and synchronization circuitry required for all parts of the device to operate effectively, as well as the user interface allowing a user to take advantage of this device.

5.2 SECURITY EVALUATION OF THE ANALOG SPEECH SCRAMBLING AND KEYSTREAM GENERATOR ALGORITHMS IN UNISON

The analog speech scrambling algorithm proposed is a two-dimensional algorithm, which passes the analog speech signal through two phases. The first phase consists of separating the frequency components of the speech signal between seven bands, and manipulating these distinctly within each band. The frequency manipulation consists of enhancing or attenuating the frequency components by any of 16 possible levels, for a duration of 1.25 ms. This is equivalent to configuring a particular filter, and passing the

incoming speech signal through this filter for a duration of 1.25 ms, after which the configuration of the filter changes. The total number of possible filters is set at 268 435 456. The filtered output speech signal is then passed through an amplitude multiplication scrambling algorithm, in which it is multiplied by one of two possible levels for a duration of 0.3125 ms. This algorithm, operated with the specified number of filters and amplitude multiplication levels, was found to be most secure on the basis of its output being statistically uncorrelated from its input, on its output statistically resembling white noise as much as possible, and furthermore on the number of possible transformations it offers to the incoming speech signal. To support this level of security, the PAM processes whose levels are responsible for configuring the filters, as well as operating the amplitude multiplication scrambling algorithm, should be statistically independent of each other. Furthermore, the levels within a particular PAM process should have zero mean, and a uniform distribution. The burden of supporting the above mentioned security level is thus placed on the system or algorithm selecting the various PAM levels.

The keystream generator proposed makes use of a secret key and in return provides eight long practically independent secure keystreams. It consists of a multilayer nonlinear generator operating in stream mode, and a DES-based key scheduling algorithm operating in block mode. Together they provide 2^{64}-1 64-bit sequences, each of which is broken up into eight 8-bit sequences, and used to initialize eight 8-bit linear feedback shift registers in parallel. These shift registers each produce one period of a 255-bit maximal length sequence, before being reinitialized. This is repeated until all 2^{64}-1 64-bit sequences have been exhausted, thus resulting in eight parallel keystreams each of length 4.7039×10^{21} bits. This keystream generator was put together as such so that its keystreams satisfy the three requirements of guaranteed minimum period, large linear equivalence, and randomness. The latter requirement is satisfied by satisfying three other requirements, namely a discrepancy of only one between the number of ones and the number of zeros in one period of the keystream, a certain specified consistency of

runs, and a constant out-of-phase autocorrelation.

It is now obvious that the analog speech scrambling algorithm and the keystream generator algorithm can be put to work together. More specifically, the eight keystreams produced by the keystream generator will be used to choose between the various PAM levels of the eight PAM processes of the analog speech scrambling algorithm. The burden is thus on the keystream generator to ensure that the PAM processes of the analog speech scrambling algorithm are statistically independent, and that the levels within a PAM process are uniformly distributed, and have zero mean. The constraint of zero mean is readily satisfied by choosing the value of the levels of the PAM processes appropriately. The constraint of uniform distribution on the eight PAM processes implies that the probability of a particular PAM level occurring is equal to the probability of any other PAM level occurring, for a particular PAM process. This means that a particular PAM level within a particular PAM process will have no effect or influence on the next level of this PAM process, hence no statistical dependence between PAM levels. On the other hand, the constraint of statistical independence between levels of different PAM processes implies that a particular level, within a particular PAM process, will not dictate a past, present, or future level of another PAM process.

Before attempting to show to what extent the proposed keystream generator will support the constraints necessary for the secure operation of the analog speech scrambling algorithm, it is appropriate to examine what are the possible ways in which the cryptanalyst may attack the analog speech scrambling device, based on the amount of information he/she may have.

One source of information that the cryptanalyst is believed to have ample of is scrambled speech from the output of the analog speech scrambling device. Let this type of attack be referred to as the "scrambled speech only" attack. For the proposed analog speech scrambling algorithm, and hence the proposed analog speech scrambling device to be able to withstand such an attack, the proposed keystream generator of this device

should be able to satisfy two weaker constraints than those of uniform distribution of PAM levels within a PAM process, and statistical independence between PAM processes. These are the constraints of maximum in-phase and constant out-of-phase autocorrelation of a particular PAM process, as well as constant cross-correlation between any two PAM processes respectively.

The autocorrelation constraint is a weaker constraint than that of uniform distribution of the PAM levels within a PAM process. The former constraint implies that PAM levels within a particular PAM process do not resemble previous or subsequent levels within the same PAM process. Even if this constraint is satisfied there still may be certain known relationships between PAM levels, which allow one level to be obtained from another, thus violating the uniform distribution of PAM levels constraint, even though these levels have no visible resemblance.

The cross-correlation constraint is a weaker constraint than that of statistical independence between PAM processes. The former constraint implies that PAM levels between two different PAM processes do not resemble each other. Even if this constraint is satisfied there still may be certain known relationships between PAM levels of different PAM processes, which allow for one level to be obtained from another thus violating the statistical independence constraint between PAM processes, even though levels between these processes have no visible resemblance.

Assuming no loss of generality from the simulation of a scaled down version of the proposed keystream generator configured with a particular chosen key in the previous chapter, the autocorrelation and cross-correlation plots of keystreams resulting from the simulated scheme are shown in figure 4.5 and figure 4.7 respectively. From these it is seen that the autocorrelation and cross-correlation constraints are satisfied by the proposed keystream generator. As a result, the analog speech scrambling device can withstand a "scrambled speech only" attack.

Another source of information that the cryptanalyst may have obtained, by some means other than descrambling, is original unscrambled speech corresponding to known scrambled speech. Of course, the cryptanalyst is not considered to have ample original speech, but enough to qualify for what may be referred to as a "known original speech" attack. This type of attack exceeds the previously mentioned "scrambled speech only" attack in strength, because the cryptanalyst has some original speech in addition to ample scrambled speech. Thus, he/she has the flexibility to perform a "scrambled speech only" attack, as well as an attack on the generated keystreams. The latter consists of using the original speech to scrambled speech relationship, to obtain portions of the keystreams equivalent in length to portions of the known original speech. There even exist tools like the spectrograph which can help the cryptanalyst in this procedure. Having portions of keystreams, the attack on these keystreams then consists of attempting to predict the remaining respective portions of these keystreams, or portions of other keystreams. Obviously, because of the larger amount of information assumed to be at the cryptanalyst's disposal, this attack is much more powerful than the "scrambled speech only" attack. As a result, the analog speech scrambling device would be considered most secure if it could withstand this attack.

For the proposed analog speech scrambling algorithm and hence the proposed analog speech scrambling device to be able to withstand an attack on the keystreams as part of a "known original speech" attack, the proposed keystream generator should be able to satisfy the two constraints mentioned earlier of statistical independence between PAM processes, as well as uniform distribution of the PAM levels within a PAM process. In a few words, these constraints should be satisfied in order to withstand an attack on the keystreams as part of a "known original speech" attack, because as mentioned previously this attack will allow the cryptanalyst to obtain portions of the respective keystreams. The cryptanalyst will use these portions in an attempt to find relationships between them, so that he/she can then use these relationships to generate future por-

tions of keystreams in advance so as to descramble forthcoming speech. If the constraints of statistical independence between PAM processes and uniform distribution of PAM levels within a PAM process are satisfied, then even if the cryptanalyst obtains the portions of keystreams he/she cannot use them in any productive way because there will be no existent relationships between these portions to be found. As a result, the attack on the keystreams will be unfruitful. The security level offered by the analog speech scrambling algorithm earlier, contingent upon satisfaction of the constraints of statistical independence between PAM processes, and uniform distribution of PAM levels within a particular PAM process, must thus be the same security level required to withstand an attack on the keystreams as part of a "known original speech" attack, since this is also contingent upon satisfaction of the same constraints.

The analog speech scrambling device would be considered most secure if it could withstand a "known original speech" attack. As mentioned previously, such an attack can be looked upon as a "scrambling speech only" attack, as well as an attack on the keystreams. As shown previously, the analog speech scrambling device can withstand a "scrambled speech only" attack. Whether or not this device can also withstand an attack on the keystreams will depend on whether or not it can satisfy the constraints of statistical independence between PAM processes, and uniform distribution of PAM levels within a particular PAM process as mentioned earlier.

The above constraints are partially satisfied, but may be totally satisfied in all practical consideration. This is because an attack on the keystreams will yield some information to the cryptanalyst but not in any practical way. As mentioned in the previous chapter, having the first 15 bits of a keystream one can obtain the configuration of the respective linear feedback shift register, and thus obtain the next 240 bits of this keystream. Since four bits are required to define a PAM level for any of the seven PAM processes representing the hopping filters algorithm, knowing the first four consecutive PAM levels will allow the prediction of the next 59 such PAM levels. Furthermore, one

bit is required to define a PAM level of the PAM process representing the amplitude multiplication scrambling algorithm. Thus, knowing the first 15 consecutive PAM levels will allow the prediction of the next 240 such PAM levels. To obtain any further information the cryptanalyst would then have to know the first eight bits of every 255-bit sequence of the keystream following the first. He/she would thus have to know the first two PAM levels to obtain the next 61 PAM levels of this sequence, assuming this keystream is applied to any of the seven PAM processes representing the hopping filters algorithm. Furthermore, the cryptanalyst would have to know the first eight PAM levels of a particular 255-bit sequence to obtain the following 247 PAM levels, assuming this keystream is applied to the PAM process representing the amplitude multiplication scrambling algorithm.

A complete keystream is made up of 2^{64}-1 255-bit concatenated maximal length sequences. It is seen that PAM levels of a particular PAM process can be obtained from previous PAM levels resulting from within the same 255-bit sequence. On the other hand, PAM levels of a particular PAM process resulting from a particular 255-bit sequence cannot be obtained from previous PAM levels, resulting from another 255-bit sequence. This is because there is uniform distribution of PAM levels across the 255-bit sequence boundaries. Since having knowledge of various PAM levels throughout the entire keystream is impractical, the PAM levels can be considered practically uniformly distributed.

The remaining constraint of statistical independence between PAM processes implies that there should not be any relationship between PAM levels of different PAM processes. The eight different PAM processes are controlled by eight different keystreams, and the eight different keystreams are generated by eight independent differently configured linear feedback shift registers. Thus, there is no relationship between PAM levels of different PAM processes, at least between PAM levels defined by the keystream portions coming after the initialization sequences of each 255-bit sequence. The indepen-

dence of the PAM levels defined by the initialization sequences of each respective keystream can be questioned since they are not generated as independent entities, but rather as one entity in the form of a 64-bit word. In other words what is questioned here is how dependent are the 8-bit sequence portions of a particular 64-bit word, resulting from the second section of the keystream generator. Beyond the exclusive-OR operation the second section consists of the DES and DES^{-1} algorithms. Every bit of a particular DES or DES^{-1} output is a function of all bits of the data port and key port of the respective algorithms. Thus, all bits within one word resulting from either the DES or DES^{-1} algorithms are related to each other through the data port and key port bits. This relationship though is not visible outside the DES or DES^{-1} algorithms. Finding such a relationship would be equivalent to breaking the DES algorithm. This, besides being very difficult due to the strength of DES, it is also further discouraged by the nonlinear operation of the exclusive-OR within the second section of the keystream generator, which provides a one-way mixing operation of the outputs of the DES and DES^{-1} algorithms, thus hiding either one. As a result, every 8-bit initialization sequence is practically independent from other such sequences within one particular 64-bit word. Thus, the PAM levels defined by the initialization sequences resulting from different respective keystreams can be practically considered statistically independent. Consequent to practically satisfying the required constraints mentioned above, the analog speech scrambling device can also withstand an attack on the keystreams, thus being able to withstand a "known original speech" attack. The proposed analog speech scrambling device, involving the proposed analog speech scrambling algorithm and proposed keystream generator operating together, can thus be considered most secure.

5.3 The Analog Speech Scrambling Device Proposed

Although in the previous section the analog speech scrambling device referred to the combined secure operation of the analog speech scrambling algorithm, and the

keystream generator algorithm, there is slightly more to it. There is additional circuitry necessary for the operation of the device as a whole, as well as circuitry necessary for providing an interface between the device and the user. A formal description of the entire device follows.

The proposed analog speech scrambling device consists of two main units, one performing the scrambling operation on outgoing speech, referred to as the transmitter unit, and one performing the descrambling operation on incoming speech referred to as the receiver unit. The transmitter unit and receiver unit can each be further divided into three sub-units. The first sub-unit of the transmitter unit consists of the analog speech scrambling algorithm as put forth in chapter III. Correspondingly, the first sub-unit of the receiver unit consists of an analog speech descrambling algorithm. This is similar to the analog speech scrambling algorithm, with the difference that the scrambled speech input to this algorithm is first passed through the amplitude multiplication scrambling algorithm, and then through the hopping filters algorithm. The PAM levels of the PAM processes representing the amplitude multiplication descrambling algorithm in this case are such, that division is effectively performed on the scrambled speech as opposed to the multiplication operation in the transmitter unit. Furthermore, the PAM levels of the seven PAM processes representing the hopping filters algorithm in this case are such, that the scrambled speech is passed through the inverse filter configurations relative to the filter configurations in the transmitter unit.

The second sub-units of both transmitter and receiver units consist of independent versions of the DES-based keystream generator, as put forth in chapter IV.

Finally, the third sub-units of the transmitter and receiver units consist of timing and synchronization circuitry respectively. These are needed to maintain the generation of the keystreams, and to keep the operation of the receiver unit at either end of the communication path synchronized with the operation of the respective transmitter unit at the opposite end, a basic requirement for reliable recovery of original speech at the

receiver.

As shown in figure 5.1, the third sub-unit of the transmitter unit of the analog speech scrambling device is mainly responsible for controlling the timing in the transmitter unit.

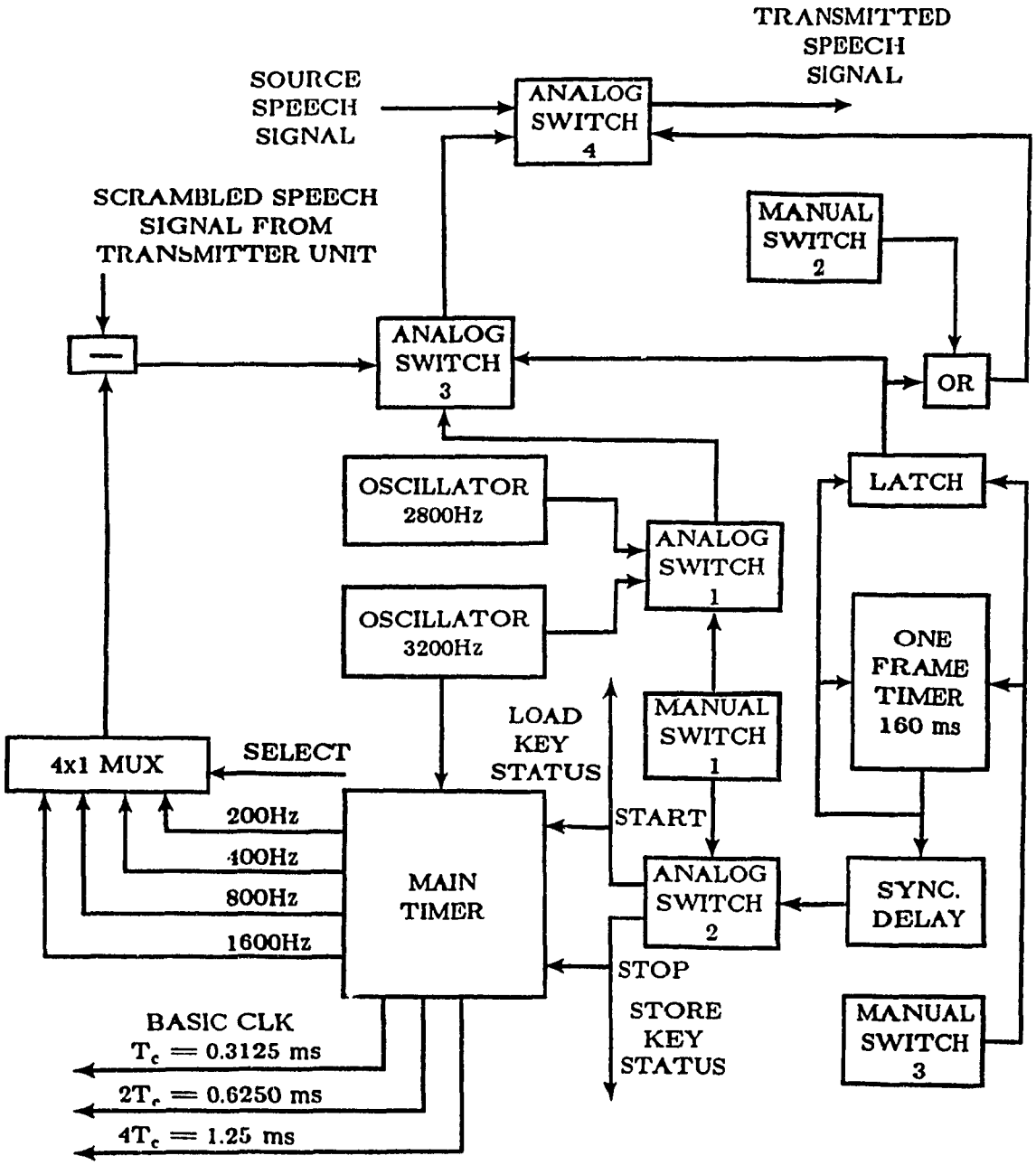

Figure 5.1 Timing circuit in transmitter unit.

It consists of a subtractor, a 4x1 multiplexer, two oscillators, a main timer, a one-frame timer, four analog switches, and three manual switches. Part of this sub-unit manipulates the already scrambled signal resulting from the scrambling algorithm, by subtracting from it one of four possible binary signals of various frequencies produced by the main timer. The frequencies of these signals are multiples of two from each other, and the choice between them is controlled through the 4x1 multiplexer, using two bits from a S/P converter of length two at the output of the keystream generator responsible for operating the amplitude multiplication scrambling algorithm. Before pressing the third manual switch, the first manual switch should be set accordingly so as to choose either the 2800 Hz oscillator output, or the 3200 Hz oscillator output, through the first analog switch, to appear as input to the third analog switch. The 3200 Hz oscillator output should be chosen at the beginning of a session between two communicating parties, or throughout such a session if necessary, while the 2800 Hz oscillator output should be chosen at the end of a session between two communicating parties. When the third manual switch is pressed it acts as a clock to the latch whose input is wired high, and thus latches a high which selects the already chosen oscillator output through the third analog switch. The latched high disables the operation of the second manual switch through the OR gate, and selects the output of the third analog switch (which is one of the oscillator outputs in this case) to pass through the fourth analog switch as output of the transmitter unit, thus resulting in the output of a sync. clock to the receiver. When pressed, the third manual switch also sends a pulse to the one-frame timer which in turn produces an end-of-frame pulse after 160 ms. The latter pulse resets the latch to low, thus choosing to pass the scrambled signal through the third analog switch and thus stopping the oscillator signal from being sent to the output of the transmitter unit after 160 ms. Furthermore, resetting the latch to low enables the second manual switch once again, allowing through the fourth analog switch the choice between the original speech signal when the select is low, and its scrambled version when the select is high to pass to

the receiver. From this point on, the third analog switch will remain with its select low until the third manual switch is pressed once again. The end-of-frame pulse besides resetting the latch sends a signal through the second analog switch, for either of two purposes as chosen by the first manual switch. At the beginning of a session between two communicating parties, and throughout this session, the position of the first manual switch is such, so as to allow the signal through the second analog switch to load the most recent reference key status on the device, and initially start or restart the main timer. Similarly, at the end of a session between two communicating parties the position of the first manual switch is such, so as to allow the signal through the second analog switch to store the most recent reference key status, and finally stop the main timer. The notion of a reference key status will be explained shortly.

As shown in figure 5.2, the third sub-unit of the receiver unit of the analog speech scrambling device is mainly responsible for providing the synchronization in the receiver unit. It consists of an adder, a main timer, an oscillator, one manual and one analog switch, two bandpass filters, two envelope detectors, and two sync. delays. Part of this sub-unit performs the descrambling procedure corresponding to the scrambling procedure performed by the analogous part of this sub-unit in the transmitter unit. The manual switch is used to control the analog switch in choosing between scrambled speech or descrambled speech to appear at the output of the receiver unit. Besides the scrambled speech signal there are two other signals coming into the receiver. These are the 3200 Hz oscillator output and the 2800 Hz oscillator output from the opposite-end transmitter unit, having duration of only 160 ms. Only one such signal can come into the receiver at any one time. When this happens, the signal is identified by bandpass filtering is passed through an envelope detector, and used for either of two purposes depending if it is identified as the 3200 Hz signal or the 2800 Hz signal. The 3200 Hz signal is used to load the most recent reference key status on the device and initially start or restart the main timer of the receiver, in synchronization with the opposite-end transmitter. The

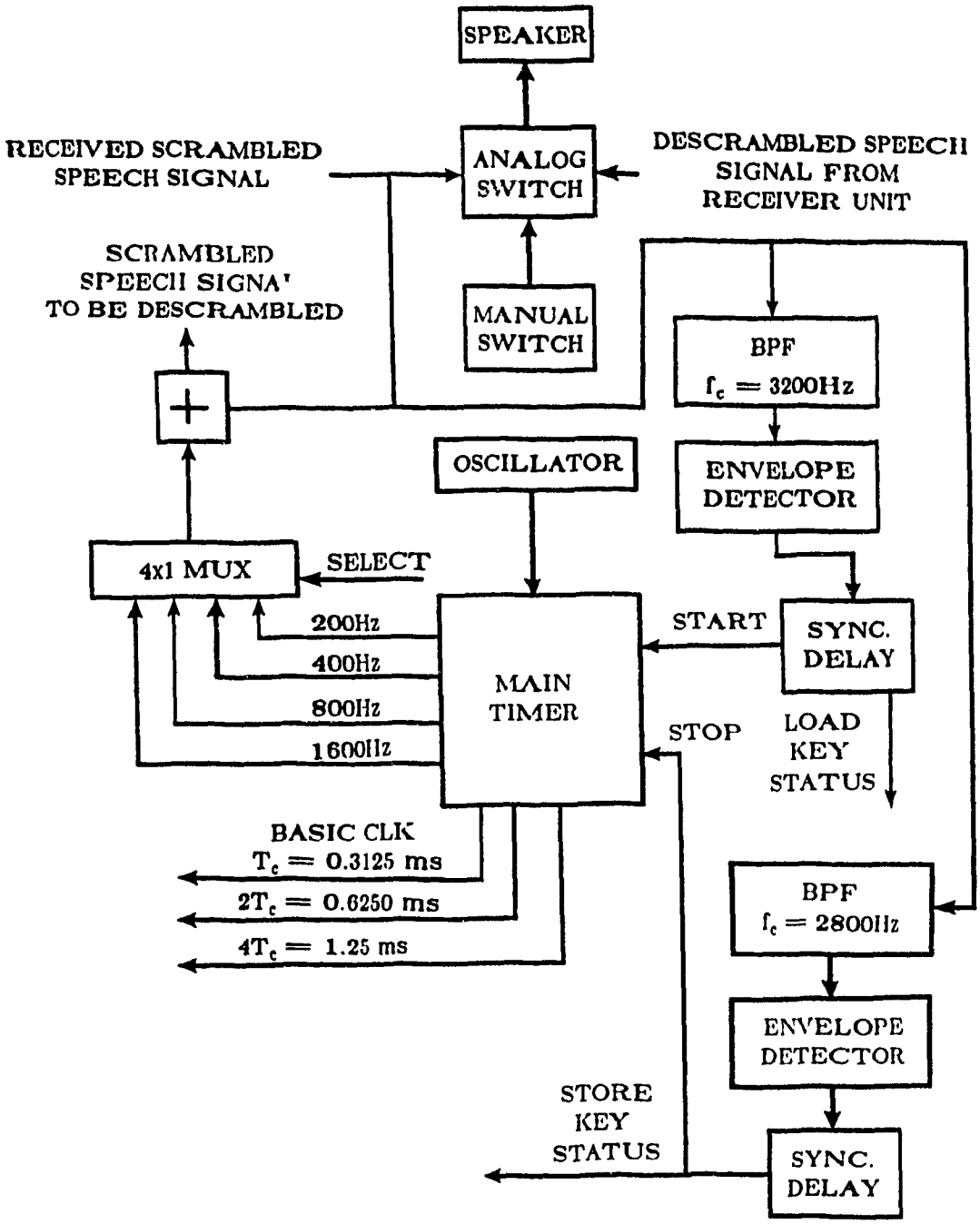

Figure 5.2 Synchronization circuit in receiver unit.

2800 Hz signal is used to store the most recent reference key status, and stop the main timer of the receiver, again in synchronization with the opposite-end transmitter.

Having fully described the operation of the proposed analog speech scrambling device, it remains to propose a user interface which can facilitate the user's use of this device. The need for a user interface arises from two factors namely,

1) a large key material, and

2) the need to make efficient use of the large period of the generated keystreams.

Practically, the entire configuration of the keystream generator is left to the user's discretion through the key material. The key material includes the maximal length configuration of the 64-bit linear feedback shift register at the base of the main multilayer nonlinear generator, as well as the maximal length configurations of the eight subordinate linear feedback shift registers. The constraint here is that these configurations have to lead to maximal length sequences. In this case, for a shift register of length L there exist $\frac{\phi(2^L-1)}{L}$ possible configurations to choose from, all leading to distinct sequences at the output of the shift register. These configurations can either be set through switches on the device itself, or supplied by the manufacturer as plug-in modules. Furthermore, the key material includes the nonlinear logic configurations for all six layers of the multilayer generator, as well as the initial condition or initial load of the 64-bit linear feedback shift register at the base of this multilayer generator. The multiplier connections within each nonlinear logic layer should satisfy certain constraints. They should be such that the minimum number of nonlinear logic layers are used, producing a sequence at the output of the multilayer generator which has maximum linear equivalence, and noiselike characteristics. It was found through simulation that changing the nonlinear logic of at least one layer, or changing nonlinear logic between layers leads to a different sequence being generated at the output of the multilayer generator. The number of all possible nonlinear configurations among all layers was found through simulation for registers up to length 14, as shown in table 2.2. There is no closed form function relating the length of the shift registers in a multilyer

generator, to the number of all possible nonlinear logic configurations among all layers, leading to distinct sequences at the output of this multilayer generator and hence at the outputs of the keystream generator. These configurations can be supplied by the manufacturer as plug-in modules identified numerically.

There are 2^{64}-1 possible initial loads for the 64-bit linear feedback shift register at the base of the multilayer generator. It was found through simulation that changing the initial load of this shift register resulted in the same sequence with different phase at the output of this generator. Furthermore, two sequences of different phase at the output of the multilayer generator were found through simulation to produce different keystreams at the outputs of the keystream generator.

The required key material is for the operation of one keystream generator. The analog speech scrambling device proposed has two such keystream generators, one for its transmitter unit, and one for its receiver unit. There are thus two sets of key materials required for the operation of the device. The keystream generator of the receiver unit of a particular device will consist of the key material identifying the opposite-end user, while the keystream generator of the transmitter unit of this device will consist of the key material identifying the owner user of the device, with respect to the opposite-end user. Thus, a particular user may be identified by a different key material depending with whom he/she is communicating. The user interface can thus be used to store the large amounts of key material as well as to easily load this key material on the device when necessary.

Any keystream at the output of the keystream generator has a length of 4.7039×10^{21} bits. Since this keystream is generated at the rate of 3200 Hz, one period of this keystream will theoretically be generated in approximately 46.6 billion years. The security benefits of having long keystreams are not fully exercised, if every time two users engage in a communication session their respective sets of keystreams are generated once again from the very beginning. In this way, only a very short portion of

each keystream (the same size on average) will be used repeatedly, thus giving the cryptanalyst further opportunity to compromise the device. The user interface can thus be used not only to store the initial key material identifying two users, and allow easy loading of this key material, but to also maintain a reference key status for every subsequent communication session between these two particular users. The reference key status will consist of key material as well as the states of all shift registers in the keystream generator algorithm. The reference key status will be loaded on the device at the beginning of a particular communication session between two users, and its updated version stored back into the user interface at the end of this communication session. It is possible that at some point within a communication session the users may feel that they are losing synchronization, as a result of not being able to understand what each other is saying. This can be resolved by loading once again the same reference key status that was loaded on the device at the beginning of this particular communication session. The user interface should allow all the reference key material between the owner of the device and a particular user to be stored under this user's name, for identification purposes. Similarly for all other users. Furthermore, all this reference key status information may be stored in encrypted form, possibly encrypted by the DES algorithm, and the portion referring to a particular user decrypted when necessary for use. This serves as a precaution against someone stealing the memory units containing all the reference key status information. Finally, as an added precaution to prevent others from tampering with the analog speech scrambling device, a password system should be employed allowing only authorized users to make use of this device.

5.4 Concluding Remarks

The analog speech scrambling algorithm of chapter III and the DES-based keystream generator of chapter IV are compatible. Together they make up the major part of an analog speech scrambling device which can be used for secure speech communications

even under a "known original speech" attack. The remaining portion of the device consists of timing and synchronization circuitry, needed in order to keep the keystream generator of the transmitter of one device, running synchronously with the keystream generator of the receiver of the opposite-end device, a basic requirement for reliable recovery of original speech at the receiver. Finally, as a result of a large key material and the need to make efficient use of the large period of the generated keystreams, a user interface is suggested, which allows a user to load the appropriate key material easily on the device. Furthermore, it will allow the status of all shift registers as well as the key material, collectively referred to as the reference key status, to be stored after the end of one communication session between two users, so that it may be loaded as is again on the device at the beginning of the next communication session between these users.

CHAPTER VI

SUMMARY, CONCLUSIONS AND FUTURE WORK

6.1 SUMMARY AND CONCLUSIONS

The telecommunication system offers endless services to the public within the communication environment, either directly, or indirectly through computers. As a result, it is very widely used, and thus very dependent upon by most, if not all. To be able to satisfy almost everyone at everyone's own will, and many times simultaneously, the telecommunication system has to be vast, flexible, and easily accessible. These qualities, besides making the telecommunication system very useful, they also make it very vulnerable to intrusion with malicious intent. Such intrusion can be active or passive. Being an environment in which the slightest inoperation or deviation from the expected operation can create problems, active intrusion can be disastrous. Passive intrusion on the other hand can be even more disastrous as a result of providing a false sense of security, which is why it was made the concern of this work. In assessing possible intrusion one has to realize that the capabilities of the intruder, and the resources at his/her disposal, will be proportional to the possible gains as a result of this intrusion. This is in turn proportional to the importance of the information being communicated. Furthermore, an intrusion can theoretically take place anywhere within the telecommunication system, but is more practically apt to occur as close to the end-user as possible. Similarly, it is just as practical to secure the telecommunication system at that point. The amount of security required will depend on one's assessment of the threat of possible intrusion. It may be that a particularly high security level can be attained in theory, without being able to support this level of security practically. On the other hand, a particularly high level of security may be attained in practice, which under theoretical conditions may be meaningless. Assessing the threat of intrusion practically, results in choosing a practical

security level appropriate for a particular intruder, if this intruder is known. On the other hand, if an intruder is not known then worst-case conditions should be assumed, but within practical limits.

After assessing the threat of possible intrusion, and settling on a particular required security level, the next step is to actually provide this security. This can be done through the use of secure algorithms in the form of a system or device placed as close to the end-user as possible. The general form of such a device would consist of two distinct algorithms. One would be responsible for disguising the confidential information entering the telecommunication system, or similarly revealing the disguised information coming from the telecommunication system. The other algorithm would be responsible for operating the former in a random fashion, through the use of a key or keystream accordingly, as would be required for security purposes.

One class of secure algorithms are those operating on data, and leading to data cipher systems. Data cipher systems can be further classified into various categories based at least on the secrecy of their keys (private or public), the type of transformation which these keys perform (substitution, transposition, or algebraic), and the structure of the entire algorithm (block or stream). A block cipher transforms an input block of characters into an output block of characters as governed by a fixed-size key. On the other hand, a stream cipher consists of encrypting a stream of plaintext characters, by combining or mixing these on a one-to-one basis with a relatively long or infinite sequence of similar characters. The difference between stream ciphers and block ciphers lies mainly in the manipulation of the key. The basic ideas behind data cipher systems can be applied directly or indirectly to speech security. This is mainly as a result of digitized speech resembling data, as well as speech secure systems in general requiring pseudorandom operation.

Speech secure systems can be classified into three categories; digital speech secure or encryption systems, analog speech encryption systems, and analog speech scrambling

systems. Digital speech encryption techniques have been known to achieve the highest degrees of security over all speech encryption techniques. Despite this, these are not readily used since digital transmission is still not quite compatible with today's technical environment, the major part of which is geared towards analog transmission. The next best thing to digital speech encryption is analog speech encryption. An analog speech encryption system allows for compatibility with today's technical environment, while at the same time taking advantage of the high security level inherent in digital speech encryption. Although very strong, this area of speech security has been fully exhausted, especially with the constant developments of digital signal processing. The only area of speech security which at present does not offer the highest levels of security, but which still allows room for further development is the area of analog speech scrambling. Analog speech scrambling algorithms can be classified as one-dimensional or two-dimensional. One-dimensional algorithms manipulate the time, frequency, or amplitude characteristics of an analog speech signal, while two-dimensional algorithms manipulate a combination of the above characteristics. Frequency manipulation techniques produce scrambled speech signals which are not sensitive to real channel imperfections, and hence allow for good transmission. Furthermore, they allow a speech-like rhythm to pass at the output of the transmitter leading to high values of residual intelligibility. On the other hand, time manipulation techniques provide scrambled speech signals without any speech-like rhythm, but they are sensitive to real channel imperfections. Furthermore, they induce large values of communication delay. Finally masking, or amplitude manipulation techniques are transmission sensitive, but produce a noiselike output at the transmitter. They are similar to time manipulation techniques without the communication delay.

This work examined one-dimensional frequency manipulation as well as amplitude scrambling algorithms, in addition to various combinations of these as two-dimensional algorithms. The one-dimensional algorithms consisted of the already known linear addi-

tion and multiplication algorithms, as well as a new frequency manipulation algorithm referred to as the hopping filters algorithm. This algorithm divides the spectrum into seven sub-bands, similarly to a rolling code bandsplitter, but instead of using the key for permuting these sub-bands it uses it to manipulate the amplitude of the frequency components in these sub-bands, which is equivalent to filtering the incoming speech with various hopping filters. The filtering operation within a particular sub-band is represented by one out of seven statistically independent PAM processes with uniformly distributed PAM levels. Furthermore, the level to be added or multiplied in the amplitude addition or multiplication scrambling algorithms respectively is also represented by such a PAM process. These algorithms were first evaluated in terms of how statistically uncorrelated their output was to their input, or in other words how close they came to perfect secrecy. This was carried out by finding the cross-covariance of the respective algorithms. It was found that the amplitude addition scrambling algorithm can never attain statistical uncorrelation between input and output, and was thus discarded. The remaining algorithms were able to achieve statistical uncorrelation of input to output, hence approaching perfect secrecy, under the constraint of their processes having zero mean. These algorithms were then evaluated on the basis of how closely their autocorrelation resembled that of white noise. This was crossed with the conflicting requirement of maximum cryptanalytical strength, as a result of offering the largest number of possible transformations for the incoming speech signal. Of the four remaining algorithms it was found that the one satisfying the above criteria, as closely as possible, was the two-dimensional algorithm consisting of the hopping filters algorithm followed by the amplitude multiplication scrambling algorithm. Furthermore, the ideal combination of number of PAM levels was found to be 16 for each of the seven PAM processes representing the hopping filters algorithm, resulting in 268 435 456 possible hopped filters, as well as 2 PAM levels for the PAM process representing the amplitude multiplication scrambling algorithm. This implies hopping between filters every 1.25 ms and

possibly multiplying by a new level every 0.3125 ms. The results of the above security evaluation hold as long as the 8 PAM processes have zero mean, are statistically independent, and their levels are uniformly distributed.

Having chosen a particular analog speech scrambling algorithm as most secure, a pseudorandom element was required to operate this algorithm in a correspondingly pseudorandom fashion. Since speech readily lends itself to stream mode, a particular keystream generator was necessary, similar to that which would form the basis of stream cipher. The only difference is that the required algorithm should provide 8 keystreams simultaneously and independently. A scheme for this keystream generator was developed based on the requirements that it produce random keystreams, each having a guaranteed minimum length and a large linear equivalence. The randomness requirement itself represented three other requirements, namely a discrepancy of only one between the number of ones and the number of zeros, a certain specified consistency of runs of ones and runs of zeros, as well as a constant out-of-phase autocorrelation for each keystream. The proposed keystream generator algorithm was a combination of three sections. The first section consisted of a multilayer nonlinear generator, and supported the requirements of minimum guaranteed keystream length and large linear equivalence. The second section, developed from a DES-based key scheduling algorithm, enhanced the large linear equivalence, supported the randomness requirement of constant out-of-phase autocorrelation, and contributed to the requirement of minimum guaranteed keystream length. Finally, the last section consisted of eight maximal length linear sequence generators, each with a complementing facility. These contributed to the minimum guaranteed keystream length requirement, and supported the requirement concerning the relative number of ones to number of zeros in a keystream. The proposed keystream generator algorithm satisfying the above requirements resulted in eight keystreams with zero cross-correlation between them, each made up of 2^{64}-1 255-bit concatenated maximal length sequences.

Having the major components of an analog speech scrambling device these were then put together and evaluated as a whole with respect to security. More specifically, they were evaluated on the basis of being able to withstand a "known original speech" attack, which is more powerful than, and covers, a "scrambled speech only" attack. Thus, to be able to withstand a "known original speech" attack the analog speech scrambling device should be able to withstand a "scrambled speech only" attack, as well as an attack on the keystreams. It was found that the autocorrelation and cross-correlation requirements, imposed on the eight keystreams controlling the eight PAM processes, allowed for the ability of the analog speech scrambling device to withstand a "scrambled speech only" attack. To be able to withstand an attack on the keystreams, the analog speech scrambling device should have a keystream generator, whose eight keystreams control the eight PAM processes in such a way that these are statistically independent, and have PAM levels of uniform distribution, the two conditions originally assumed when evaluating the security of the analog speech scrambling algorithm alone. It was found that the DES-based key scheduling scheme within the keystream generator algorithm, as well as the distinct configurations of the eight subordinate generators, contributed to the production of the eight keystreams in such a way that these satisfied the requirement of statistical independence between PAM processes. Furthermore, the linear equivalence of the generated keystreams although not strictly satisfying the requirement of uniform distribution of PAM levels, satisfied this requirement within all practical considerations. Thus, with the support of the keystream generator algorithm the security evaluation of the analog speech scrambling algorithm holds. Furthermore, the analog speech scrambling algorithm can withstand an attack on the keystreams, allowing this device to withstand a "known original speech" attack, and hence allowing for this device to be declared as most secure.

Having the major components of an analog speech scrambling device, and having declared the combined operation of these components as most secure, it was necessary to

introduce the timing and synchronization circuitry of this device. This circuitry allows for the operation of the receiving unit of one analog speech scrambling device, to be synchronized with the operation of the transmitting unit of the opposite-end analog speech scrambling device, a basic requirement for recovery of the original speech at the receiver. This is accomplished through the transmission of a preamble at the beginning and end of secure operation, sent from the transmitter at one end, to the opposite-end receiver. Finally, to complete the analog speech scrambling device, a user interface was put forth, which would allow storage and retrieval of the large key material applicable to any combination of two users. Furthermore, this would allow for the keystream generation between communication sessions of two particular users to be continuous, as opposed to reinitializing the keystreams at every communication session between the same two users. This allows the full advantage of having keystreams of large periods.

6.2 Future Work

This work may be considered as one-sided since most of it concentrated mainly on the analog speech scrambling algorithm, as opposed to the corresponding analog speech descrambling algorithm. One course of action for future work is to go through a similar analysis as the one in this work, but for the analog speech descrambling algorithm. This analysis could be split into two cases as governed by the input to the descrambling algorithm. One case would consist of the input being basic scrambled speech, while the other would consist of scrambled speech as well as additional noise and channel degradations imposed, which would be inevitable in a more realistic system. It would also be interesting to analyze the effect of lack of synchronization between corresponding transmitting and receiving units. Obviously, the evaluation carried out in all these cases would not be on the basis of security, but rather on the quality of speech measured through speech degradation. At the onset of these analyses an acceptable level of degradation should be decided upon, and this made the basic requirement.

Finally, the entire system, consisting of a transmitter and a receiver can be simulated. Towards this direction this work has already provided the simulation of the keystream generator through the simulation program in Appendix A. This represents a scaled down version of the proposed generator scheme. Given the appropriate key material, it can produce each of the eight keystreams one at a time. Having this part of the simulation it then remains to simulate the analog speech scrambling and descrambling algorithms, as well as to provide a suitable input signal, which represents the analog speech signal most appropriately. The results of the scrambling as well as descrambling algorithms can then be observed, and compared to the analytic results, which include the results of this work.

REFERENCES

[1] H.J. Beker and F.C. Piper, *Secure Speech Communications*, (London : Academic Press, 1985).

[2] C.E. Shannon, "Communication Theory of Secrecy Systems," *Bell Syst. Tech. J.*, vol. 28, pp. 656-715, 1949.

[3] H.J. Beker and F.C. Piper, *Cipher Systems : The Protection of Communications*, (London : Northwood Publications, 1982).

[4] H. Katzan Jr., *The Standard Data Encryption Algorithm*, (New York : Petrocelli Books, Inc., 1977).

[5] C.H. Meyer and S.M. Matyas, *Cryptography : A New Dimension in Computer Data Security*, (New York : John Wiley & Sons, 1982).

[6] B. Beckett, *Introduction to Cryptology*, (Oxford : Blackwell Scientific Publications, 1988).

[7] T.A. Berson, "Long Key Variants of DES," in *Advances in Cryptology : Proceedings of Crypto '82*, (New York : Plenum Press, 1983), pp. 311-313.

[8] M.E. Hellman, and J.M. Reyneri, "Drainage and the DES - Summary," in *Advances in Cryptology : Proceedings of Crypto '82*, (New York : Plenum Press, 1983), pp. 129-131.

[9] M.E. Hellman "A Cryptanalytic Time - Memory Trade-Off," *IEEE Trans. Inform. Theory*, vol. IT-26, no. 4, pp. 401-406, July 1980.

[10] J.J. Quisquater et al., "The Importance of 'Good' Key Scheduling Schemes (How to Make a Secure DES Scheme With $\leq$ 48-Bit Keys ?)" in *Advances in Cryptology - CRYPTO '85*, (Berlin : Springer-Verlag, 1986), pp. 537-542.

[11] E.J. Groth, "Generation of Binary Sequences With Controllable Complexity," *IEEE Trans. Inform. Theory*, vol. IT-17, no. 3, pp. 288-296, May 1971.

[12] A. Gersho "Perfect Secrecy Encryption of Analog Signals," *IEEE J. Select. Areas Commun.*, vol. SAC-2, no. 3, pp. 460-466, May 1984.

[13] C.E. Shannon, "Coding Theorems for a Discrete Source With a Fidelity Criterion," *IRE Nat. Conv. Rec.*, Part 4, pp. 142-163, March 1959.

[14] A. Gersho and R. Steele, "Guest Editorial : Encryption of Analog Signals - A Perspective," *IEEE J. Select. Areas in Commun.*, vol. SAC-2, no. 3, pp. 423-425, May 1984.

[15] G. Chou and L. Lee, "A New Efficient and Practical DFT Scrambling System for Secure Speech Communications," *Nat. Telecommun. Conf.*, New Orleans, LA. pp. E8.5.1-E8.5.5, Dec. 1981.

[16] L. Lee et al., "A New Frequency Domain Speech Scrambling System Which Does Not Require Frame Synchronization," *IEEE Trans. Commun.*, vol. COM-32, no. 4, pp. 444-456, April 1984.

[17] G. Chou and L. Lee, "An Efficient Time Domain Sample Value Scrambling Scheme Eliminating Frame Synchronization Requirement for Secure Speech Communications," *Global Telecommun. Conf.*, Miami, FL, pp. B7.6.1-B7.6.5, Dec. 1982.

[18] L. Lee et al., "A New Time Domain Speech Scrambling System Which Does Not Require Frame Synchronization," *IEEE J. Select. Areas Commun.*, vol. SAC-2, no. 3, pp. 443-455, May 1984.

[19] L. Lee and G. Chou, "A General Theory for Asynchronous Speech Encryption Techniques," *IEEE J. Select. Areas Commun.*, vol. SAC-4, no. 2, pp. 280-287, March 1986.

[20] N.S. Jayant "Analog Scramblers for Speech Privacy," *Computers & Security 1*, pp. 275-289, 1982.

[21] S.C. Kak and N.S. Jayant, "On Speech Encryption Using Waveform Scrambling," *Bell Syst. Tech. J.*, vol. 56, pp. 781 - 808, May-June 1977.

[22] E. Belland and N. Bryg, "Speech Signal Privacy System Based on Time Manipulation," *Proc. Carnahan Conf. Crime Countermeasures*, University of Kentucky, 1978.

[23] N.S. Jayant et al., "A Comparison of Four Methods for Analog Speech Privacy," *IEEE Trans. Commun.*, vol. COM-29, pp. 18-23, Jan. 1981.

[24] N.S. Jayant et al., "Analog Scramblers for Speech Based on Sequential Permutations in Time and Frequency," *Bell Syst. Tech. J.*, Jan. 1983.

[25] S.T. Hong and W. Kuebler, "An Analysis of Time Segment Permutation Methods in Analog Voice Privacy Systems," *Proc. Carnahan Conf. Crime Countermeasures*, University of Kentucky, 1981.

[26] A. Goniotakis and A.K. Elhakeem, "Security Evaluation of a New Analog Speech Scrambling/Privacy Device Using Hopping Filters," to be published in *J. Select. Areas Commun.*, June 1990.

[27] L.L. Flanagan et al., "Speech Coding," *IEEE Trans. Commun.*, vol. COM-27 no. 4, pp. 710-737, April 1979.

[28] A. Pappoulis, *Probability, Random Variables and Stochastic Processes*, (New York : McGraw-Hill, Inc., 1984).

[29] S. Haykin, *Communication Systems*, (New York : John Wiley & Sons, 1983).

[30] L.W. Couch II, *Digital and Analog Communications Systems*, (New York : Macmillan Publishing Co., Inc., 1983).

[31] A. Goniotakis et al., "Security Evaluation of a New Analog Speech Scrambling Device Using a Nonlinear DES-Based Keystream Generator," to be presented in *ICC/SUPERCOMM '90, April 1990.*

[32] A. Goniotakis and A.K. Elhakeem, "Introduction and Security Evaluation of a New Analog Speech Encryption Device," in *Proceedings of the 12th Symposium of Information Theory and its Applications (SITA '89)*, vol. 2, pp. 625-630, Dec. 1989.

APPENDIX A

The following program referred to as "regtogdes.p" was written to simulate the operation of the DES-based keystream generator used in the proposed analog speech scrambling device.

```
program simcode(input, desin, inlindat, output, codout);

const
  POINTS = 65025;
  MAXLENGTH = 8;
  MAXLAYER = 3;
  SUBSP = 1;
  MAINSP = 64;

type
  layerind = 1..MAXLAYER;
  inex = 1..MAXLENGTH;
  binary = 0..1;
  binarr = array[inex] of binary;
  intarr = array[inex] of inex;
  contstuff = array[layerind] of binarr;
  spanstuff = array[layerind] of intarr;
  multistuff = record
                 taps : binarr;
                 contents : contstuff;
                 spans : spanstuff;
                 length,layers : integer;
               end;
  linstuff = record
               taps, contents : binarr;
               length : integer
             end;
  bin64 = array[1..64] of binary;
  bin48 = array[1..48] of binary;
  keyarr = array[1..16] of bin48;
  bin28 = array[1..28] of binary;
  int16 = array[1..16] of integer;
  bin32 = array[1..32] of binary;
  selfuncarr = array[1..32] of int16;

var
  inlindat, desin, codout : text;
  maingen : multistuff;
  subgen : linstuff;
  subnum, b, base, m, s, t, tot, val : integer;
  pmc1c, pmc1d : bin28;
  pmc2, selectt : bin48;
  numshift : int16;
  perm : bin32;
  iperm, invperm, dat, key1, key2, initcond, outword : bin64;
  selfunc : selfuncarr;
  mainout, subout, tog, orsum : binary;
```

```
function power(bas, expon : integer) : integer;

  var
    t, pow : integer;

  begin
    pow := 1;
    for t := 1 to expon do
      pow := pow * bas;
    power := pow
  end;

function complement(bitt : binary) : binary;.

  begin
    if bitt = 1
      then
        complement := 0
      else
        complement := 1;
  end;

function orr(bitt1, bitt2 : binary) : binary;

  begin
    if (bitt1 = 1) and (bitt2 = 1)
      then
        orr := 1
      else
        orr := bitt1 + bitt2;
  end;

function logicand(bitt1, bitt2 : binary) : binary;

  begin
    logicand := bitt1 * bitt2
  end;

function xor(bitt1, bitt2 : binary) : binary;

  begin
    xor := logicand(bitt1, complement(bitt2)) + logicand(complement(bitt1), bitt2)
  end;

function togle(bitt : binary) : binary;

  begin
    if bitt = 1
      then togle := 0
      else if bitt = 0
           then togle := 1;
  end;
```

```
procedure update(sp : intarr; var reg : binarr; lenreg : inex; var upout : binary);

 type
  tapstuff = record
               left, right : binary;
               first : boolean
             end;
  tapstor = array[inex] of tapstuff;

 var
  tapsearch : tapstor;
  sum, prod : binary;
  t : integer;

 begin
  for t := 1 to lenreg do
    begin
      tapsearch[t].left := 0;
      tapsearch[t].right := 0;
      tapsearch[t].first := true
    end;
  for t := 1 to lenreg do
    if tapsearch[sp[t]].first
      then
        begin
          tapsearch[sp[t]].left := t;
          tapsearch[sp[t]].first := false
        end
      else
        tapsearch[sp[t]].right := t;
  sum := 0;
  for t := 1 to lenreg do
    if not(tapsearch[t].first)
      then
        begin
          prod := logicand(reg[tapsearch[t].left], reg[tapsearch[t].right]);
          sum := xor(sum, prod)
        end;
  upout := sum;
  for t := lenreg downto 1 do
    reg[t] := reg[t - 1];
 end;

procedure generatornonlin(var gen : multistuff; var genout : binary);

 var
  resbit, shiftbit, xorsum : binary;
  t : integer;

 begin
  xorsum := 0;
  for t := 1 to gen.length do
    if gen.taps[t] = 1
```

```
      then
        xorsum := xor(xorsum, gen.contents[1][t]);
    shiftbit := xorsum;
    for t := 1 to gen.layers do
      begin
        update(gen.spans[t], gen.contents[t], gen.length, resbit);
        gen.contents[t][1] := shiftbit;
        shiftbit := resbit
      end;
    genout := shiftbit
  end;

procedure generatorlin(var gen : linstuff; var genout : binary);

  var
    xorsum : binary;
    t : integer;

  begin
    xorsum := 0;
    for t := 1 to gen.length do
      if gen.taps[t] = 1
        then
          xorsum := xor(xorsum, gen.contents[t]);
    genout := gen.contents[gen.length];
    for t := gen.length downto 2 do
      gen.contents[t] := gen.contents[t-1];
    gen.contents[1] := xorsum
  end;

procedure storshift(var reg : bin64; siz : integer; bitin : binary);

  var
    t : integer;
  begin
    for t := 1 to (siz - 1) do
      reg[t] := reg[t+1];
    reg[siz] := bitin;
  end;

procedure shift(var sh : bin28; shiftval: integer);

  var
    t, m : integer;
    temp : binary;

  begin
    for t := 1 to shiftval do
      begin
        temp := sh[1];
        for m := 1 to 27 do
          sh[m] := sh[m+1];
        sh[28] := temp
      end;
```

```
  end;

procedure keygen(totkey : bin64; md : integer; var skeys : keyarr);

  var
    c, d : bin28;
    t, keyind, direc : integer;

  begin
    for t := 1 to 28 do
      c[t] := totkey[pmc1c[t]];
    for t := 1 to 28 do
      d[t] := totkey[pmc1d[t]];
    for keyind := 1 to 16 do
      begin
        shift(c, numshift[keyind]);
        shift(d, numshift[keyind]);
        if md = 1
          then
            direc := keyind
          else
            direc := 17 - keyind;
        for t := 1 to 48 do
          if pmc2[t] <= 28
            then
              skeys[direc][t] := c[pmc2[t]]
            else
              skeys[direc][t] := d[(pmc2[t] - 28)];
      end;
  end;

procedure des(datin : bin64; var key : bin64; mode : integer);

  type
    bin6 = array[1..6] of binary;
    bin4 = array[1..4] of binary;

  var
    s, w, ind, row, column, resval : integer;
    lefdatin, ridatin, selres : bin32;
    datselect, sum : bin48;
    subkeys : keyarr;
    bitsin : bin6,
    bitsout : bin4;
    tempstor : binary;

  begin
    keygen(key, mode, subkeys);
    for s := 1 to 32 do
      begin
        lefdatin[s] := datin[iperm[s]];
        ridatin[s] := datin[iperm[s + 32]]
      end;
    for ind := 1 to 16 do
```

```
      begin
       for s := 1 to 48 do
         begin
          datselect[s] := ridatin[selectt[s]];
          sum[s] := xor(datselect[s], subkeys[ind][s])
         end;
       for s := 1 to 8 do
         begin
          for w := 1 to 6 do
            bitsin[w] := sum[(6 * (s - 1)) + w];
            row := ((4 * (s - 1)) + 1) + ((2 * bitsin[1]) + (1 * bitsin[6]));
            column := (8 * bitsin[2]) + (4 * bitsin[3]) + (2 * bitsin[4]) + (1 * bitsin[5]) + 1;
            resval := selfunc[row][column];
            for w := 4 downto 1 do
              begin
               bitsout[w] := resval mod 2;
               resval := resval div 2
              end;
            for w := 1 to 4 do
              selres[(4 * (s - 1)) + w] := bitsout[w];
         end;
       for s  := 1 to 32 do
         begin
          tempstor := xor(selres[perm[s]], lefdatin[s]);
          lefdatin[s] := ridatin[s];
          ridatin[s] := tempstor
         end;
      end;
    for s := 1 to 64 do
      if invperm[s] <= 32
       then
        key[s] := ridatin[invperm[s]]
       else
        key[s] := lefdatin[invperm[s] - 32];
  end;

procedure initdes(var wword : bin64);

  var
    t : integer;

  begin
    for t := 1 to 64 do
      wword[t] := 0;
  end;

begin
  reset(desin);
  reset(inlindat);
  rewrite(codout);
  readln(desin);
  for m := 1 to 8 do
    begin
     for t := 1 to 8 do
```

```
      read(desin, iperm[((m -1) * 8) + t]);
    readln(desin)
  end;
readln(desin);
for m := 1 to 8 do
  begin
    for t := 1 to 4 do
      read(desin, perm[((m -1) * 4) + t]);
    readln(desin)
  end;
readln(desin);
for m := 1 to 8 do
  begin
    for t := 1 to 8 do
      read(desin, invperm[((m -1) * 8) + t]);
    readln(desin)
  end;
readln(desin);
for m := 1 to 4 do
  begin
    for t := 1 to 7 do
      read(desin, pmc1c[((m -1) * 7) + t]);
    readln(desin)
  end;
readln(desin);
for m := 1 to 4 do
  begin
    for t := 1 to 7 do
      read(desin, pmc1d[((m -1) * 7) + t]);
    readln(desin)
  end;
readln(desin);
for m := 1 to 8 do
  begin
    for t := 1 to 6 do
      read(desin, pmc2[((m -1) * 6) + t]);
    readln(desin)
  end;
readln(desin);
for m := 1 to 8 do
  begin
    for t := 1 to 6 do
      read(desin, selectt[((m -1) * 6) + t]);
    readln(desin)
  end;
readln(desin);
for m := 1 to 16 do
  read(desin, numshift[m]);
readln(desin);
readln(desin);
readln(desin);
for m := 1 to 32 do
  begin
    for t := 1 to 16 do
```

```
      read(desin, selfunc[m][t]);
    readln(desin);
    if ((m mod 4) = 0) and (m <> 32)
      then
        readln(desin)
  end;
readln(inlindat);
readln(inlindat);
readln(inlindat, maingen.length);
maingen.layers := round(ln(maingen.length)/ln(2));
if maingen.layers < (ln(maingen.length)/ln(2))
 then
   maingen.layers := maingen.layers + 1;
readln(inlindat);
for m := maingen.layers downto 1 do
  begin
    for t := 1 to maingen.length do
      read(inlindat, maingen.spans[m][t]);
    readln(inlindat);
  end;
readln(inlindat);
for t := 1 to maingen.length do
  read(inlindat, maingen.taps[t]);
readln(inlindat);
readln(inlindat);
for t := 1 to maingen.length do
  read(inlindat, maingen.contents[1][t]);
readln(inlindat);
readln(inlindat);
readln(inlindat, subnum);
readln(inlindat);
readln(inlindat, subgen.length);
base := (subnum - 1) * subgen.length;
readln(inlindat);
for t := 1 to subgen.length do
  read(inlindat, subgen.taps[t]);
readln(inlindat);
initdes(key1);
initdes(key2);
tog := 1;
for t := 1 to ((((maingen.layers - 1) * maingen.length) + (MAINSP - 1))) do
  begin
    generatornonlin(maingen, mainout);
    storshift(dat, MAINSP, mainout)
  end;

for m := 1 to (power(2, maingen.length) - 1) do
  begin
    generatornonlin(maingen, mainout);
    storshift(dat, MAINSP, mainout);
    des(dat, key1, 1);
    des(dat, key2, -1);
    for t := 1 to MAINSP do
      initcond[t] := xor(key1[t], key2[t]);
```

```
        orsum := 0;
        for t := 1 to subgen.length do
          orsum := orr(orsum, initcond[base+t]);
        for t := 2 to subgen.length do
          subgen.contents[t] := initcond[base+t];
        subgen.contents[1] := orr(initcond[base+1], complement(orsum));
        tog := togle(tog);
        for t := 1 to (power(2, subgen.length) - 1) do
          begin
            for s := 1 to SUBSP do
              begin
                generatorlin(subgen, subout);
                subout := xor(tog, subout);
                storshift(outword, SUBSP, subout)
              end;
            if SUBSP = 1
              then
                writeln(codout, outword[1]:1)
              else
                begin
                  tot := 0;
                  for b := 1 to SUBSP do
                    begin
                      val := outword[b] * power(2, (SUBSP - b));
                      tot := tot + val
                    end;
                  writeln(codout, tot:2)
                end;
          end;
      end;
end.
```

The following are the permutation and selection functions particular to the DES algorithm. They are used as input to the program regtogdes.p, in order for part of this program to simulate the operation of the DES algorithm.

```
IPERM
58 50 42 34 26 18 10 2
60 52 44 36 28 20 12 4
62 54 46 38 30 22 14 6
64 56 48 40 32 24 16 8
57 49 41 33 25 17  9 1
59 51 43 35 27 19 11 3
61 53 45 37 29 21 13 5
63 55 47 39 31 23 15 7
PERM
16  7 20 21
29 12 28 17
 1 15 23 26
 5 18 31 10
 2  8 24 14
32 27  3  9
19 13 30  6
22 11  4 25
INVPERM
40  8 48 16 56 24 64 32
39  7 47 15 55 23 63 31
38  6 46 14 54 22 62 30
37  5 45 13 53 21 61 29
36  4 44 12 52 20 60 28
35  3 43 11 51 19 59 27
34  2 42 10 50 18 58 26
33  1 41  9 49 17 57 25
PMC1C
57 49 41 33 25 17  9
 1 58 50 42 34 26 18
10  2 59 51 43 35 27
19 11  3 60 52 44 36
PMC1D
63 55 47 39 31 23 15
 7 62 54 46 38 30 22
14  6 61 53 45 37 29
21 13  5 28 20 12  4
PMC2
14 17 11 24  1  5
 3 28 15  6 21 10
23 19 12  4 26  8
16  7 27 20 13  2
41 52 31 37 47 55
30 40 51 45 33 48
44 49 39 56 34 53
46 42 50 36 29 32
SELECT
32  1  2  3  4  5
 4  5  6  7  8  9
 8  9 10 11 12 13
12 13 14 15 16 17
16 17 18 19 20 21
20 21 22 23 24 25
24 25 26 27 28 29
```

```
28 29 30 31 32  1
NUMSHIFT
1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1
SELECTION FUNCTIONS
S1
14  4 13  1  2 15 11  8  3 10  6 12  5  9  0  7
 0 15  7  4 14  2 13  1 10  6 12 11  9  5  3  8
 4  1 14  8 13  6  2 11 15 12  9  7  3 10  5  0
15 12  8  2  4  9  1  7  5 11  3 14 10  0  6 13
S2
15  1  8 14  6 11  3  4  9  7  2 13 12  0  5 10
 3 13  4  7 15  2  8 14 12  0  1 10  6  9 11  5
 0 14  7 11 10  4 13  1  5  8 12  6  9  3  2 15
13  8 10  1  3 15  4  2 11  6  7 12  0  5 14  9
S3
10  0  9 14  6  3 15  5  1 13 12  7 11  4  2  8
13  7  0  9  3  4  6 10  2  8  5 14 12 11 15  1
13  6  4  9  8 15  3  0 11  1  2 12  5 10 14  7
 1 10 13  0  6  9  8  7  4 15 14  3 11  5  2 12
S4
 7 13 14  3  0  6  9 10  1  2  8  5 11 12  4 15
13  8 11  5  6 15  0  3  4  7  2 12  1 10 14  9
10  6  9  0 12 11  7 13 15  1  3 14  5  2  8  4
 3 15  0  6 10  1 13  8  9  4  5 11 12  7  2 14
S5
 2 12  4  1  7 10 11  6  8  5  3 15 13  0 14  9
14 11  2 12  4  7 13  1  5  0 15 10  3  9  8  6
 4  2  1 11 10 13  7  8 15  9 12  5  6  3  0 14
11  8 12  7  1 14  2 13  6 15  0  9 10  4  5  3
S6
12  1 10 15  9  2  6  8  0 13  3  4 14  7  5 11
10 15  4  2  7 12  9  5  6  1 13 14  0 11  3  8
 9 14 15  5  2  8 12  3  7  0  4 10  1 13 11  6
 4  3  2 12  9  5 15 10 11 14  1  7  6  0  8 13
S7
 4 11  2 14 15  0  8 13  3 12  9  7  5 10  6  1
13  0 11  7  4  9  1 10 14  3  5 12  2 15  8  6
 1  4 11 13 12  3  7 14 10 15  6  8  0  5  9  2
 6 11 13  8  1  4 10  7  9  5  0 15 14  2  3 12
S8
13  2  8  4  6 15 11  1 10  9  3 14  5  0 12  7
 1 15 13  8 10  3  7  4 12  5  6 11  0 14  9  2
 7 11  4  1  9 12 14  2  0  6 10 13 15  3  5  8
 2  1 14  7  4 10  8 13 15 12  9  0  3  5  6 11
```