EMMANUEL FELIPE DA SILVA CÔRTES

CIFRAÇÃO DE ARQUIVOS WAVE

Brasília-DF, Dezembro de 2006

EMMANUEL FELIPE DA SILVA CÔRTES

CIFRAÇÃO DE ARQUIVOS WAVE

Monografia apresentada ao Centro Universitário de Brasília, UNICEUB, como pré-requisito para a conclusão do curso de bacharelado em Engenharia da Computação.

Professor Orientador: Claudio Penedo de Albuquerque

Brasília-DF, Dezembro de 2006

AGRADECIMENTOS

Ao Grande Criador do mundo!

Aos meus familiares e amigos pela compreensão e apoio nas horas mais complicadas,

Ao professor Claudio Penedo por ter me orientado com autoridade nos momentos certos.

RESUMO

Este trabalho tem como objetivo a implantação de segurança em arquivos do tipo Wave. Para tal, é proposto e implementado no software Matlab, um algoritmo que efetua a cifragem de sinais de áudio armazenados neste tipo de arquivo.

Para a execução dos processos de cifração e decifração dos sinais são utilizadas as técnicas de transposição e geração de números pseudo-aleatórios, bem como as operações de xor (ou-exclusivo). Também implementa-se a geração de duas chaves que desempenham papel fundamental para a segurança do algoritmo.

Palavras-chave: software, sinais, cifrados, aleatórios, chaves.

ABSTRACT

This work has the objective of implanting security in Wave files. In order to do such task, it is proposed and developed with the Matlab software, an algorithm that does the ciphering of audio signals stored in that kind of file.

To the execution of the processes of ciphering and deciphering it is used the techniques of transposition and generation of pseudo-random numbers, as well as XOR operations (Or-exclusive). There is also the generation of two keys, that ensemble fundamental role to the algorithm's security.

Keywords: software, signals, ciphered, random, keys.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

D.O.S. – Denial of Service

Hz – Hertz

B – Bel

dB – Decibel

PAM – Pulse Amplitude Modulation

PCM – Pulse Code Modulation

DES – Data Encryption Standard

AES – Advanced Encryption Standard

NIST – National Institute of Standards and Technology

RSA – Ron **R**ivest, Adi **S**hamir e Len **A**dleman

LSB – Least Significant Bit

1- INTRODUÇÃO

A necessidade de uma interligação mundial imposta pelo processo de globalização, no qual a troca de informações é de vital importância para a vida de todos, e a crescente expansão dos meios de comunicação, tais como internet, telefonia fixa, telefonia móvel, faz com que um grande número de usuários tenha acesso a essas tecnologias. Isso ocasiona um aumento considerável no número de mensagens circulando pelos canais de dados. Esse grande fluxo de dados gera grandes preocupações em torno do processo de comunicação, principalmente no que diz respeito à segurança no tráfego das informações. [Soares,Lemos,Colcher,1995]

A segurança para as mensagens pode ser provida de duas maneiras: a primeira é garantir um canal de transmissão seguro, ou seja, garantir que as mensagens só sejam acessadas por usuários autorizados; a segunda é garantir segurança nas próprias mensagens, de forma que, se forem acessadas por indivíduos intrusos, sejam ininteligíveis ou imperceptíveis a eles. [Soares,Lemos,Colcher,1995]

Aplicações de segurança baseadas na confiabilidade do canal de transmissão podem ser garantidas em redes pequenas, onde todos os pontos são controláveis e todos os acessos são monitorados. Em redes de maior magnitude, nas quais o número de usuários é muito alto e o controle sobre os pontos não é viável, como a internet por exemplo, esse tipo de segurança torna-se falho, pois a rede pode ser acessada por pessoas mal intencionadas. Nessas redes, o melhor método a ser utilizado é o método de garantir a segurança nas próprias mensagens.

A metodologia de tornar a mensagem segura por si mesma é uma medida eficaz tanto para transmissão de dados quanto para seu armazenamento, visto que existe uma série de equipamentos e técnicas que podem ser usados para exploração da vulnerabilidade de canais e de terminais. Escutas telefônicas e receptores/transmissores de radiofreqüência são exemplos de ferramentas que exploram pontos fracos de canais de comunicação; backdoors, D.O.S. (Denial of

Service) e fllooding, são técnicas utilizadas para exploração de pontos fracos nos terminais ou hosts [Buchmann,2002].

Com o intuito de garantir maior segurança às informações que são armazenadas em hosts ou que trafegam pelos canais de transmissão algumas técnicas de manipulação de mensagens foram elaboradas, de forma que a informação inicial não possa ser vista ou entendida por algum agente intruso ou não autorizado. As três principais técnicas de provimento de segurança por meio da manipulação das mensagens são a criptografia, esteganografia e o scrambler.

A criptografia é uma técnica que objetiva tornar uma informação inicial ininteligível, por meio de sua manipulação. Essa técnica pode ser aplicada a qualquer tipo de arquivo.

A esteganografia tem como objetivo esconder uma informação inicial dentro de uma outra informação, chamada de hospedeira, de forma que não seja percebida a existência da informação inicial. Esta técnica pode ser usada para qualquer tipo de arquivo.

O scrambler é a criptografia aplicada à manipulação de sinais, podendo ser aplicada a sinais de áudio, de vídeo, ou radiofônicos. Têm o objetivo de codificar um sinal de forma que ele se torne ilegível a pessoas que não tenham autorização para acessá-lo.

Este projeto visa à utilização da técnica de scrambler para prover segurança a arquivos do tipo Wave. A aplicação do scrambler aos arquivos Wave é demonstrada por meio de um algoritmo implementado com o uso da ferramenta Matlab. O resultado da execução desse algoritmo é a formação de um sinal de áudio ininteligível, a partir de um sinal de áudio originalmente legível.

Após este capítulo (Capítulo 1), as abordagens feitas nos capítulos seguintes são descritas abaixo.

No Capítulo 2 são abordados os conceitos de áudio, som, ondas sonoras e digitalização de sinais sonoros.

O Capítulo 3 trata do tema da segurança em arquivos de áudio, abordando assuntos como criptografia, esteganografia e scrambler.

O Capítulo 4 fala sobre o software implementado, mostrando o funcionamento das funções utilizadas e a maneira com que elas foram implantadas, mostrando também algumas telas da implementação feita.

O Capítulo 5 é a conclusão, na qual são abordados os pontos principais que foram apreendidos ao longo do desenvolvimento deste projeto. Nesse capítulo, algumas propostas para projetos futuros também são apresentadas.

2- ÁUDIO

A palavra áudio está relacionada à manipulação do som, ou seja, diz respeito ao conjunto de técnicas utilizadas para reprodução, armazenamento e transmissão sonora. Por conseguinte, o entendimento sobre o som é o ponto de partida para o estudo do áudio.

2.1 – O Som

O som pode ser definido como a sensação criada pelo cérebro, ao receber impulsos provenientes do ouvido; este por sua vez transmite esses impulsos quando é perturbado por uma sucessão de compressões e rarefações ocorridas anteriormente em um meio físico (sólido, líquido ou gasoso); tal efeito é provocado pela oscilação de moléculas no meio, que, quando se propagam, formam uma onda longitudinal.

Por exemplo, quando uma pessoa fala, o ar que é expulso dos pulmões pelo músculo diafragma, ao passar pelas pregas vocais (cordas vocais), provoca vibrações. Essas vibrações são transmitidas às moléculas de ar que estão em contato com a superfície das pregas vocais, fazendo com que se movimentem e se choquem com as moléculas a sua frente. Após o contato, a segunda camada de moléculas também se movimenta e se choca às moléculas seguintes e assim sucessivamente. Esse processo de oscilações e choques entre as moléculas no meio físico é a forma de propagação da onda sonora. O movimento das moléculas no sentido de se aproximarem umas das outras é o que caracteriza a compressão. O oposto, ou seja, o movimento das moléculas no sentido de se afastarem umas das outras é o que caracteriza a rarefação. A Figura 2.1 é um exemplo de onda longitudinal

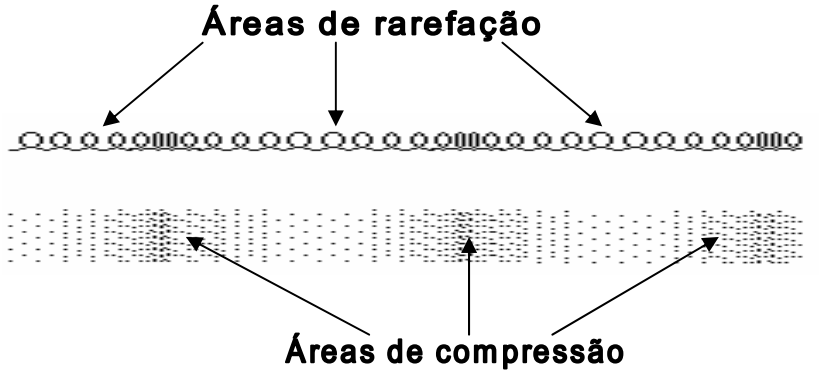


Figura 2.1 – Exemplo de ondas longitudinais.
(Fonte: [WEB3])

Na Figura 2.1, os locais de maior densidade de partículas no meio físico são as áreas de compressão. Os locais onde a densidade é menor são as áreas de rarefação.

A freqüência da onda sonora dependerá da freqüência de oscilação da fonte geradora da onda. Por exemplo, podemos imaginar uma barra de ferro fixada a um anteparo de madeira, como ilustra a Figura 2.2. Quando colocada em movimento oscilatório, a barra produz uma onda longitudinal no ar devido ao contato das moléculas de sua superfície com as moléculas de ar. Quanto maior a vibração da barra de ferro, maior será a freqüência da onda gerada.

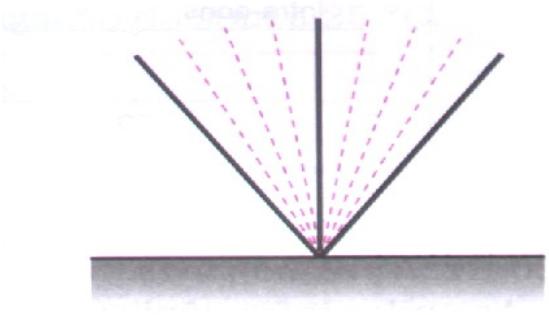


Figura 2.2 – Barra em movimento oscilatório.
(Fonte: [WEB3])

Outra característica importante da onda sonora é a de ser uma onda mecânica, ou seja, ela precisa de um meio físico (sólido, líquido ou gasoso) para se propagar. No vácuo não há propagação de ondas sonoras.

As ondas sonoras geradas em um meio são percebidas pelo ser humano, principalmente pelo sentido da audição. Quando chega à orelha , o som é captado e repassado por um canal, o conduto auditivo externo, ao tímpano. O tímpano é uma membrana que ao ser perturbada pelas oscilações da onda, passa a vibrar com a mesma freqüência desta. Essas vibrações são transmitidas ao martelo, à bigorna e ao estribo, ossos que estão ligados diretamente ao tímpano. Do estribo, as vibrações passam ao líquido existente no ouvido interno, chegando aos nervos ópticos que ao serem estimulados emitem impulsos elétricos. Estes impulsos chegam ao cérebro e são interpretados como sensações sonoras [WEB3]. A estrutura do ouvido humano pode ser vista na Figura 2.3.

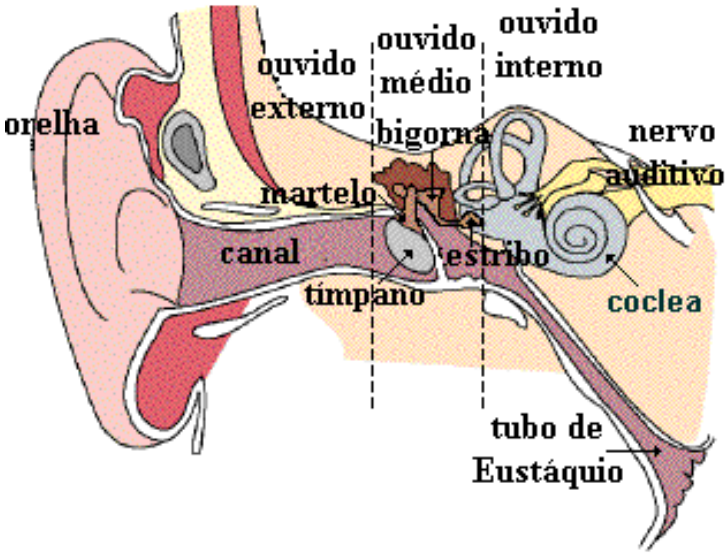


Figura 2.3 – Estrutura do ouvido humano.
(Fonte: [WEB5])

O ser humano consegue ouvir ondas sonoras que estão entre 20 Hz e 20.000 Hz. Abaixo dessa faixa o som recebe o nome de infra-som. Acima desses

valores recebe o nome de ultra-som. Alguns animais conseguem captar sinais provenientes da faixa do ultra-som e do infra-som como morcegos, cães e golfinhos. A Figura 2.4 ilustra as faixas de freqüência do som e suas classificações.

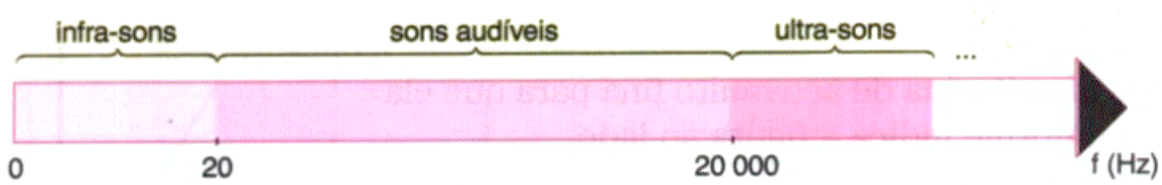


Figura 2.4 – Freqüências da onda sonora e suas classificações.
(Fonte: [WEB3])

Se a energia emitida pela fonte sonora é grande, isto é, se o som é muito forte, temos uma sensação desagradável no ouvido, pois a quantidade de energia transmitida exerce sobre o tímpano uma pressão muito forte. Quanto maior a vibração da fonte, maior a energia sonora. Portanto, quanto maior a amplitude da onda, maior será a intensidade do som. [WEB3]

O bel (B) é a medida utilizada para medir o nível de intensidade sonora. Essa unidade recebeu esse nome em homenagem a Alexander Graham Bell, inventor do telefone. Essa unidade relaciona a razão de potências em um logaritmo [Tipler,2000]:

$$I = \log_{10}\left(\frac{p}{p_0}\right) \qquad \textbf{(2.1)}$$

onde p é a potência do sinal avaliado e p_0 é a potência de referência.

É comum o uso de uma unidade múltipla do bel, o Decibel (dB), que representa a décima parte do bel. Em decibéis, a Equação **(2.1)** ficaria :

$$I = 10\log_{10}\left(\frac{p}{p_0}\right) \qquad \textbf{(2.2)}$$

Por essa relação, se dobramos a potência do som teremos um acréscimo de 3 dB no nível de intensidade. Se a potência for aumentada em 10 vezes, teremos um acréscimo de 10db no nível de intensidade. Veja:

$$I = 10\log_{10}\left(\frac{2p}{p_0}\right) = 10\log_{10}(2) + 10\log_{10}\left(\frac{p}{p_0}\right) = 3 + 10\log_{10}\left(\frac{p}{p_0}\right) \quad \textbf{(2.3a)}$$

$$I = 10\log_{10}\left(\frac{10p}{p_0}\right) = 10\log_{10}(10) + 10\log_{10}\left(\frac{p}{p_0}\right) = 10 + 10\log_{10}\left(\frac{p}{p_0}\right) \quad \textbf{(2.3b)}$$

Os sons com níveis muito intensos são desagradáveis ao ouvido humano. Sons com níveis de intensidade acima de 130 dB provocam uma sensação dolorosa e sons acima de 160 dB podem romper o tímpano e causar surdez. [WEB3]. A Tabela 2.1 mostra a variação dos níveis de intensidade do som, relacionados a situações cotidianas.

Tabela 2.1 - Exemplos de níveis de som típicos

(Fonte: [WEB6])

Pressão do som 2×10^{-5} N/m^2	Nível de Intensidade (dB)	Intensidade do som 10^{-12} W/m^2	Exemplos típicos
63,2	130	10	limiar da percepção
20	120	1,0	grande avião a jato
6,3	110	0,1	grande orquestra
2,0	100	0,01	arrebitamento
0,63	90	10^{-3}	trem
0,2	80	10^{-4}	escritório ruidoso
0,063	70	10^{-5}	motor de carro
0,02	60	10^{-6}	discurso
$6,3 \times 10^{-3}$	50	10^{-7}	escritório médio
2×10^{-3}	40	10^{-8}	escritório quieto
$6,3 \times 10^{-4}$	30	10^{-9}	biblioteca
2×10^{-4}	20	10^{-10}	sussurro
$6,3 \times 10^{-5}$	10	10^{-11}	sussurro bem baixo
2×10^{-5}	0	10^{-12}	limiar da audibilidade (a 1000 Hz)

Na Tabela 2.1, N/m^2 e W/ m^2 são as unidades utilizadas para medir a pressão e a intensidade do som, respectivamente.

2.2 – Superposição e Interferência Sonora

Uma onda senoidal pode ser entendida como um movimento circular que se propaga ao longo de um eixo, o qual pode representar uma distância ou tempo, por exemplo. A relação desse movimento com um ponto de referência é chamada de fase. [WEB7] A Figura 2.5 ilustra a representação da fase.

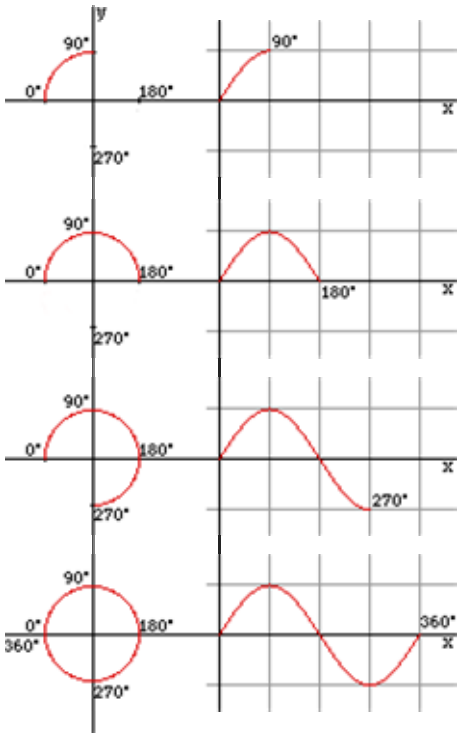


Figura 2.5 – Representação da fase
(Fonte: [WEB7])

Se dois sinais senoidais de mesma freqüência são emitidos por uma mesma fonte geradora em tempos distintos, a diferença entre os tempos pode ocasionar uma discrepância de fase entre os sinais. A Figura 2.6 mostra dois sinais com diferença de fase.

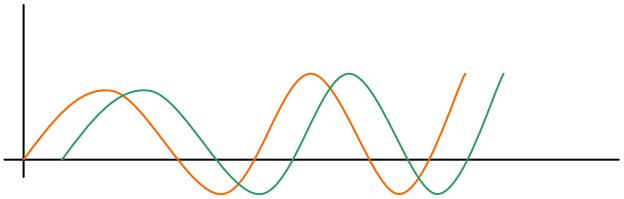


Figura 2.6 – Representação de dois sinais com fases distintas.

O princípio da superposição nos diz que *quando duas ou mais ondas se sobrepõem, a onda resultante é a soma algébrica das ondas individuais.*

Na ocorrência de superposição de ondas, as interferências causadas pelas ondas,umas nas outras, podem ser classificadas como interferências construtivas e interferências destrutivas. Os pontos de interferência construtiva ocorrem quando o módulo da resultante do somatório das ondas em determinado ponto for maior que o módulo das componentes dos sinais originais naquele ponto. Se o módulo da resultante for menor que o módulo de um dos sinais, o ponto será de interferência destrutiva.

Podemos observar na Figura 2.7a uma superposição de ondas com a resultante sendo composta apenas por pontos de interferência destrutiva, como mostra a Figura 2.7b.

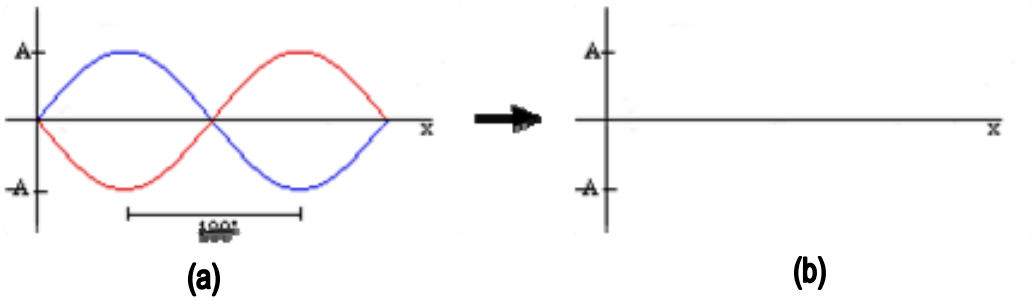
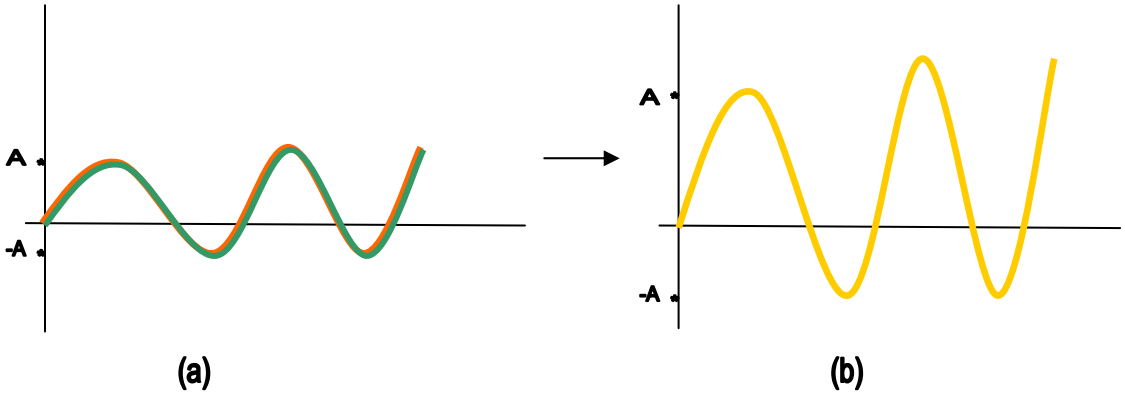


Figura 2.7 – (a) Duas ondas defasadas de 180º.
(b) Resultante da superposição das ondas.
(Fonte: [WEB7])

Na Figura 2.8a temos sobreposição de ondas com resultante (Figura 2.8b) formada apenas por pontos de interferência construtiva.



Figuras 2.8 – (a) Superposição de ondas de mesma fase.
(b) Resultante da superposição.

2.3 – Manipulação do Som

Ao longo da evolução humana, vários aparelhos foram inventados a fim de proporcionarem diferentes tipos de manipulação para o som, seja com intuito de possibilitar uma melhor comunicação entre as diversas partes do globo, ou com a finalidade de armazenar sinais sonoros como músicas, palestras, entrevistas. O telefone, o microfone, o gravador de áudio, o alto-falante e uma infinidade de aparelhos foram e são utilizados para transmissão, recepção e armazenamento sonoro.

Para que um sinal sonoro possa ser manipulado por computador, ele precisa ser representado digitalmente, pois na natureza, este sinal aparece na forma analógica .

Um sinal analógico é um sinal contínuo que varia em função do tempo, possuindo um número infinito de amostras dentro de um intervalo de tempo. Um sinal digital é um sinal discreto, ou seja, que não possui valores para representar todo instante *t* dentro de um intervalo de tempo. Portanto, para que o sinal sonoro possa ser manipulado em um meio digital, um computador por exemplo, é necessário que este tenha um número finito de amostras. A Figura 2.9 ilustra suscintamente o processo de discretização de um sinal analógico.

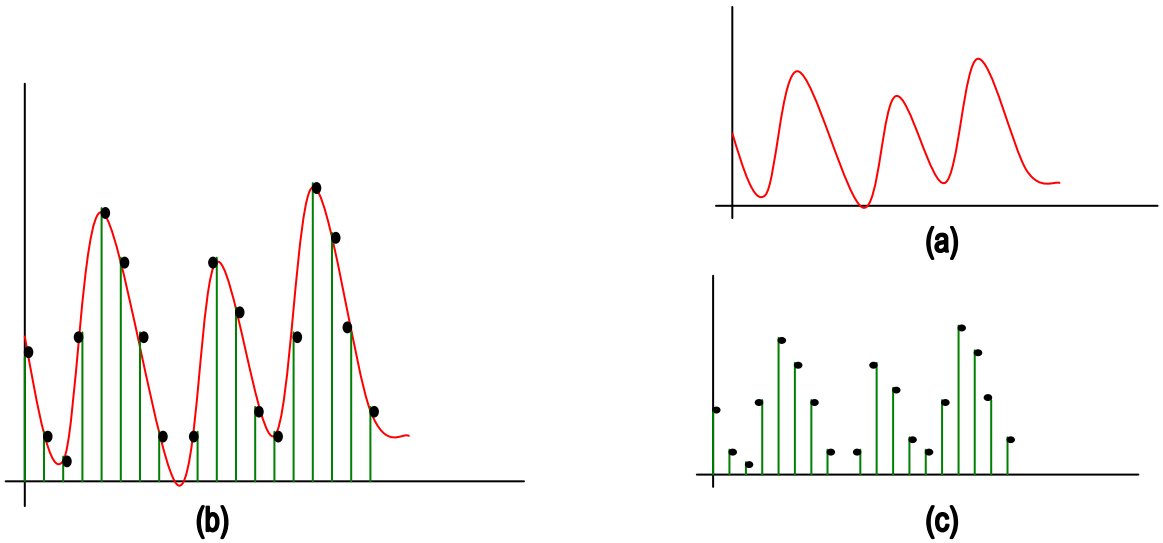


Figura 2.9 – (a) Sinal analógico.
(b) Discretização do sinal analógico.
(c) Sinal Discretizado.

Para que um sinal analógico possa ser representado digitalmente, três parâmetros devem ser levados em conta: amostragem, quantização e codificação

2.3.1 – Amostragem

Quando um sinal analógico é digitalizado, alguns pontos dentro do sinal são escolhidos para pertencerem ao sinal digital. Essa etapa é chamada de amostragem. Quanto maior a taxa de amostragem, ou seja, quanto maior o número de pontos colhidos do sinal analógico, melhor será a representação digital do sinal. A taxa de amostragem é medida em Hz. Uma taxa de amostragem de 44 kHz, significa que a cada segundo, 44.000 amostras do sinal estão sendo armazenadas. A Figura 2.10 ilustra o processo de amostragem de um sinal .

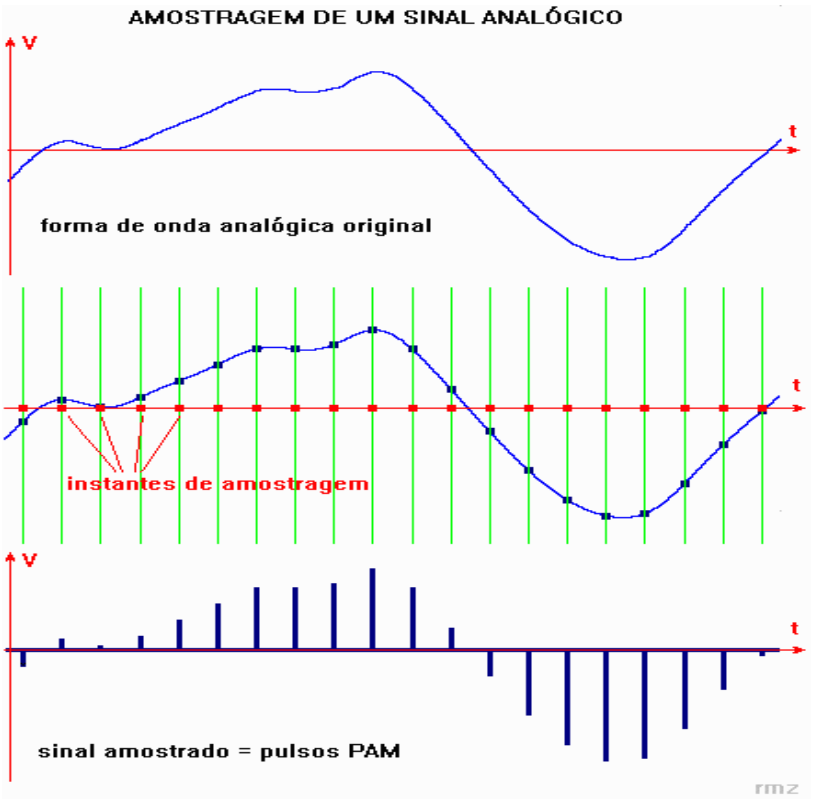


Figura 2.10 – Amostragem de um sinal analógico.
(Fonte: [WEB8])

Seja analisada a Figura 2.10. Partindo-se do sinal analógico original, são escolhidos alguns pontos (instantes de amostragem) para representarem o sinal digital. Depois de escolhidos esses pontos, um clock é acionado no intervalo de tempo determinado pela escolha dos pontos. Para cada instante de tempo escolhido, o clock comanda o fechamento do circuito e a amplitude do sinal é capturada naquele instante, formando o chamado pulso PAM (pulsos modulados em amplitude). Se uma amostra tem taxa de amostragem de 20kHz, terá 20.000 pulsos PAM por segundo.

A taxa de amostragem mínima que deve ser usada na digitalização de um sinal, para que este possa ser devidamente recuperado quando desejado foi proposta por Nyquist. O teorema de Nyquist nos diz que a taxa de amostragem deve ser igual a duas vezes o valor da maior freqüência do sinal que se quer amostrar. Por exemplo, para que seja representada de forma satisfatória toda a

gama de sons audíveis pelo ser humano, 20Hz a 20kHz aproximadamente, a taxa de amostragem mínima para essa operação deve ser de 40kHz, pois é duas vezes maior que a máxima freqüência audível.

Se a taxa de amostragem for menor que duas vezes a freqüência máxima do sinal, pode ocorrer um efeito chamado de aliasing, onde o sinal recuperado a posteriori não representará satisfatoriamente o sinal digitalizado inicialmente. As Figuras 2.11 e 2.12 ilustram um sinal sem efeito aliasing e com efeito aliasing, respectivamente.

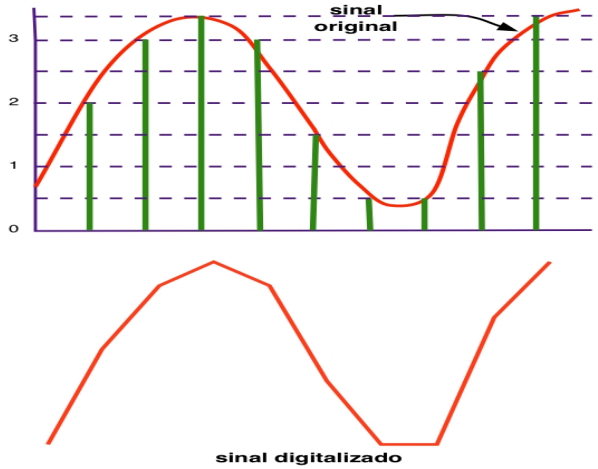


Figura 2.11 – Sinal digitalizado sem efeito aliasing.
(Fonte: [WEB10])

Os sinais que sofrem efeito aliasing podem ficar ininteligíveis ou não dependendo do grau de corruptividade recebido durante a formação a representação do sinal.

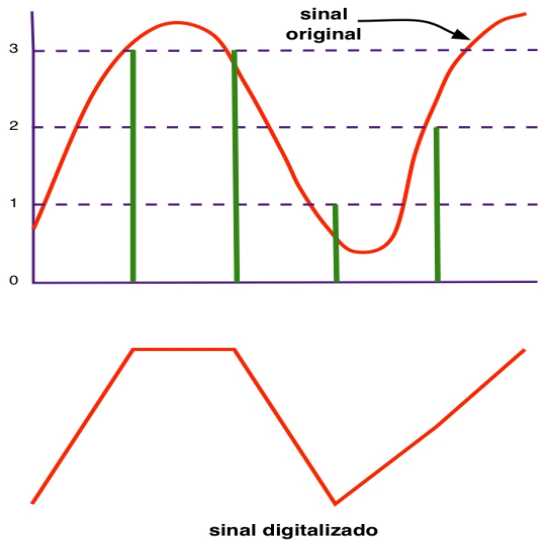


Figura 2.12 – Sinal digitalizado com efeito aliasing.
(Fonte: [WEB10])

2.3.2 – Quantização e Codificação

Assim como os pontos de tempo são escolhidos no sinal original segundo a uma determinada taxa de freqüência, a freqüência de amostragem, os pontos de amplitude também devem ser escolhidos para representação do sinal. A quantização é a escolha de pontos para representarem a amplitude do sinal. Assim sendo, quanto maior for o número de pontos escolhidos para essa representação, melhor será a forma discreta do sinal analógico original e mais fiel será a representação do sinal. A Figura 2.13 ilustra o processo de quantização.

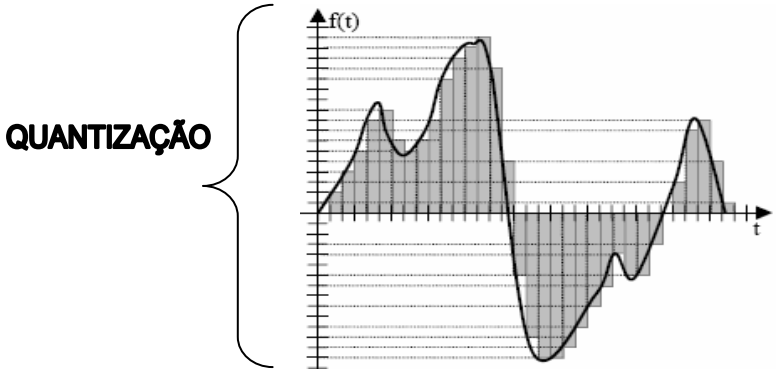


Figura 2.13 – Eixo de quantização de um sinal.
(Fonte: [WEB9])

Na quantização, um número de valores discretos (finitos) é usado para a representação da amplitude do sinal. Logo, os valores de amplitude do sinal real devem se adequar à gama de valores disponíveis dentro do intervalo de quantização. Por exemplo, se o conjunto de valores de quantização for composto pelos números inteiros entre 0 e 255, significa que 256 níveis de amplitude poderão ser representados. Logo, se a amplitude em um instante *t* no sinal real, for 10,45 V ela deverá ser ajustada para um número dentro do conjunto de valores de quantização (deve ser um número inteiro entre 0 e 255), passando a ser 11 V ou 10 V dependendo do ajuste estipulado. O número de pontos de quantização é determinado pelo número de bits usados na codificação.

A codificação é o processo de representação de um valor decimal de amplitude em um equivalente binário. Ou seja, se o número de bits disponíveis para a codificação é 16, significa que podem ser representadas por essa codificação 216 números = 65536 amplitudes diferentes. Após a representação de cada amplitude por uma seqüência binária correspondente, um novo sinal é formado, tendo como base os valores binários anteriormente gerados. Esse sinal é chamado de sinal digital PCM (Pulse Code Modulation – Modulação de Pulsos por Código). A Figura 2.14 exemplifica os processos de codificação e geração do sinal PCM.

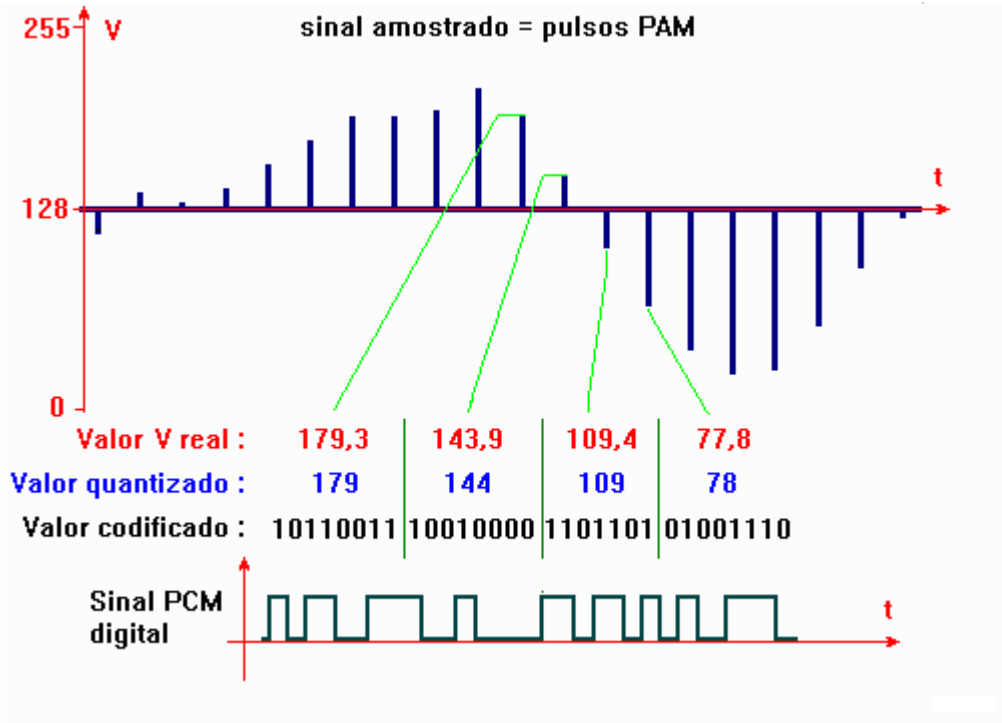


Figura 2.14 – Quantização, codificação e o sinal PCM digital.
(Fonte: [WEB8])

2.3.3 – Armazenamento de áudio

Para o armazenamento dos dados em um arquivo de áudio é necessário que os valores referentes à taxa de amostragem, à taxa de quantização e ao número de bits usados na codificação do sinal sejam também escritos no arquivo. De maneira geral, essas informações formam o chamado cabeçalho do arquivo. Existem vários formatos de arquivos de áudio, entre eles os formatos .wav, .au, .mid, .mp3. Cada um destes tipos de arquivo possui seu cabeçalho próprio.

2.3.4 – Arquivos Wave

Os arquivos Wave são formados por um cabeçalho e a parte de dados. O cabeçalho possui 44 bytes, onde estão colocadas informações sobre a taxa de amostragem, número de bits usados na codificação, tamanho do arquivo, número

de canais e taxa de transferência. A Tabela 2.2 mostra a estrutura do cabeçalho de um arquivo wave.

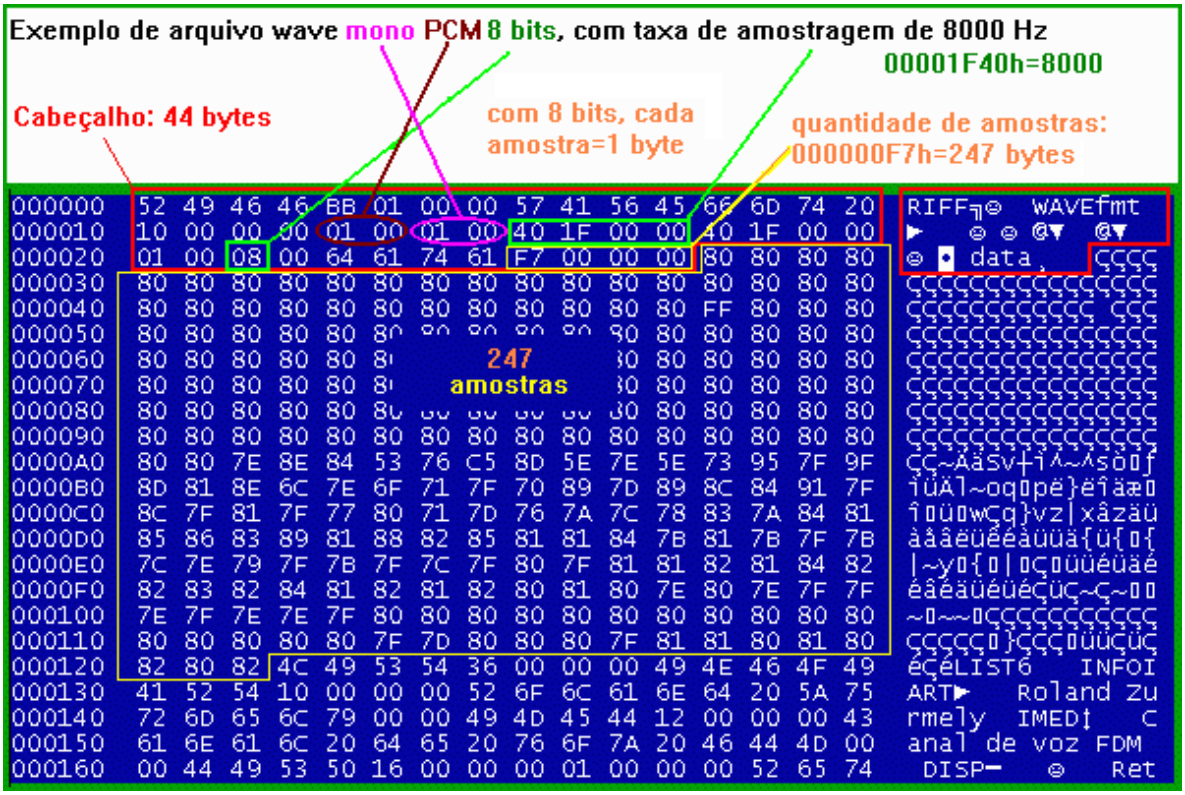
Tabela 2.2 – Estrutura do cabeçalho do arquivo Wave

(Fonte: [WEB8])

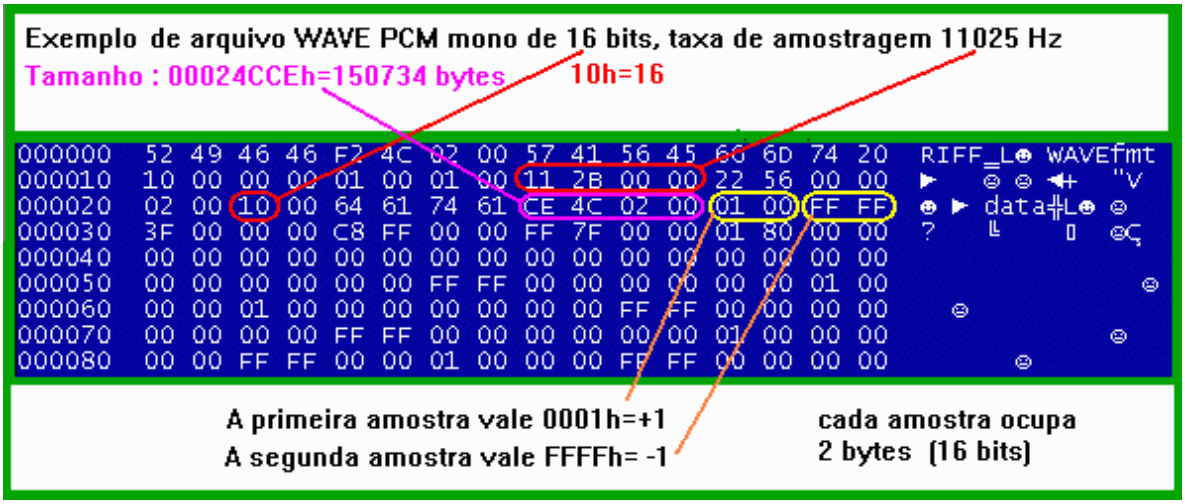
ITEM	TAMANHO	DESCRIÇÃO
RIFF	4 BYTES (0 – 3)	Código Hexadecimal da string "RIFF" (52 49 46 46)
Tamanho	4 BYTES (4 – 7)	Tamanho total do arquivo
Wavefmt	4 BYTES (8 – 15)	Código Hexadecimal da string "WAVEfmt" (57 41 56 45 66 6D 74 20)
Tamanho do sub-bloco	4 BYTES (16 – 19)	Tamanho do sub-bloco de formato
Estrutura	2 BYTES (20 – 21).	Tipo de Estrutura. - PCM (Pulse Code Modulation) Modulação de Código de Pulso)
Canais	2 BYTES (22 – 23).	Quantidade de Canais.
TxAmostragem	4 BYTES (24 – 27).	Taxa de Amostragem para PCM: 5000Hz, 11025Hz, 22050Hz, 44100Hz. Comercialmente as principais taxas de amostragem são as seguintes: 11.025 Hz: padrão geralmente utilizado para qualidade de telefone; 22.050 Hz: padrão geralmente utilizado para qualidade de rádio; 44.100 Hz: padrão geralmente utilizado para

ITEM	TAMANHO	DESCRIÇÃO
		qualidade de CD. 32.000 Hz: proporciona bons resultados para fins auditivos, com quantidade superior ao padrão de rádio e telefone e inferior ao de CD.
Tmedia	4 Bytes (28 – 31).	Taxa Média de Transferência
Qmínima	2 Bytes (32 – 33).	Quantidade mínima de bytes utilizados para representar uma amostra. Para PCM: é o número de bytes utilizados para representar uma amostra simples, incluindo os dados para ambos os canais caso seja no formato estéreo. 8 bits mono: 01, 8 bits estéreo: 02. 16 bits mono: 02, 16 bits estéreo: 04.
Nbits	2 Bytes (34 – 35).	Número de bits por amostra. 08 00: oito bits 10 00: dezesseis bits
Mark	4 Bytes (36 – 39).	Código Hexadecimal da string “data” (64 61 74 61)
SizeData	4 Bytes (40 – 43).	Número de bytes de dados a serem lidos.

As Figuras 2.15a e 2.15b, são exemplos de arquivos wave abertos em um editor de hexadecimal.



(a)



(b)

Figura 2.15 – (a) Arquivo wave em hexadecimal e ascii.
(b) Arquivo wave em hexadecimal e ascii.
(Fonte: [WEB8])

O cálculo do tamanho aproximado de um arquivo wave pode ser feito da seguinte maneira:

$$Tam = Ta * (Nb/8) * T * Nº \text{ de canais} \quad \textbf{(2.4a)}$$

Onde :
Tam = Tamanho do arquivo em bytes
Ta é a Taxa de amostragem do arquivo,
Nb é o número de bits usados na codificação,
T é o tempo total do arquivo em segundos,
Nc é o número de canais utilizados.

Por exemplo, um arquivo wave de 10 segundos, com taxa de amostragem de 44,100 kHz (qualidade de CD), usando 8 bits de codificação e 1 canal terá o tamanho de aproximadamente:

$$Tam = 44100 * (8/8) * 10 * 1 = 441000 \text{ bytes} \cong 430 \text{ KB} \quad \textbf{(2.4b)}$$

3- TÉCNICAS DE SEGURANÇA EM ÁUDIO

Um algoritmo que tem o objetivo de proporcionar segurança a um arquivo de áudio, assim como a qualquer outro arquivo, deve atender a quatro pilares básicos: integridade, sigilo ou confidencialidade, não repúdio e autenticidade.

A integridade significa o fato de a mensagem original ser recebida pelo receptor da mesma forma que foi enviada, ou seja, sem sofrer alterações que prejudiquem o entendimento correto da mensagem. Por exemplo, se a mensagem original enviada pelo emissor for “Eu fui ao dentista ontem.”, no destino ela deve chegar desta mesma forma. Se a mensagem for interceptada por um agente mal intencionado e a integridade não for garantida, ele poderá mudar conteúdo fazendo com que chegue ao destino algo do tipo “Eu não fui ao dentista ontem.” o que significa uma mudança completa no sentido da mensagem.

O sigilo por sua vez é a garantia de que apenas agentes autorizados tenham permissão de acesso a certas informações. Isto significa que se uma mensagem for enviada de um usuário X para um usuário Y, apenas X e Y poderão ter acesso à mensagem. Se um usuário Z tentar acessar a mensagem ele deve ser barrado.

O não repúdio diz respeito à confirmação do conteúdo de uma mensagem, ou seja, se na mensagem enviada pelo emissor estiver escrito “123456” ele não poderá dizer no futuro que o conteúdo da mensagem era “654321”.

A autenticidade é o processo responsável pela identificação dos interlocutores. Isto que dizer que antes de uma comunicação segura ser estabelecida é preciso que se tenha certeza sobre quem são os destinatários da mensagem e se eles são quem realmente dizem ser.

3.1 – Criptografia

A Criptografia (Do Grego *kryptós*, "escondido", e *gráphein*, "escrever") é geralmente entendida como sendo o estudo dos princípios e das técnicas pelas quais a informação pode ser transformada de sua forma original para outra ilegível

(Wikipédia, 2006), sendo que somente o conhecedor das técnicas e/ou das chaves utilizadas no processo conseguirá colocar a informação em sua forma inteligível.

No estudo da criptografia, importantes conceitos são utilizados como o de texto claro, texto criptografado e composição de cifra. A mensagem original ou mensagem legível a qual se deseja criptografar recebe a denominação de texto claro ou mensagem clara. O produto do processo de encriptação da mensagem, ou seja, a mensagem ilegível, é chamado de texto cifrado ou mensagem cifrada. O conjunto de operações envolvidas na transformação de um texto claro em um texto cifrado é denominado composição de cifra.

Quanto à técnica utilizada, os sistemas de cifras podem ser agrupados em três grandes grupos: os que utilizam transposição, os que utilizam substituição e os sistemas que utilizam as duas técnicas, chamados de sistemas híbridos

Os sistemas de transposição são aqueles os quais as letras, sílabas ou palavras têm seus lugares invertidos, ou seja, faz-se uma permutação ou embaralha mento do componente escolhido da mensagem, causando maior dispersão no texto claro. Como ilustração um simples sistema de transposição será montado. Como exemplo é tomada a palavra “Engenharia da Computação”. Agrupando as letras e os espaços vazios em conjuntos com 5 componentes e levando em conta que cada letra ocupa uma posição dentro do conjunto, a Tabela 3.1 é montada:

Tabela 3.1 – Tabela inicial utilizada pelo exemplo de transposição.

GRUPO	LETRA	POSIÇÃO NO GRUPO
1	“e”	1ª
1	“n”	2ª
1	“g”	3ª
1	“e”	4ª
1	“n”	5ª
2	“h”	1ª
2	“a”	2ª
2	“r”	3ª

Tabela 3.1 – Tabela inicial utilizada pelo exemplo de transposição. (continuação)

GRUPO	LETRA	POSIÇÃO NO GRUPO
2	“i”	4ª
2	“a”	5ª
3	“ “	1ª
3	“d”	2ª
3	“a”	3ª
3	“ ”	4ª
3	“c”	5ª
4	“o”	1ª
4	“m”	2ª
4	“p”	3ª
4	“u”	4ª
4	“t”	5ª
5	“a”	1ª
5	“ç”	2ª
5	“a”	3ª
5	“o”	4ª

Foram formados 4 grupos com 5 componentes cada (grupos 1,2,3 e 4), e um grupo com 4 componentes (grupo 5). Para completar o grupo 5 será acrescentada a letra “y” que ocupará a 5ª posição do conjunto. A Tabela 3.1 modificada ficará como mostra a Tabela 3.2.

Tabela 3.2- Tabela intermediária utilizada pelo exemplo de transposição.

GRUPO	LETRA	POSIÇÃO NO GRUPO
1	“e”	1ª
1	“n”	2ª
1	“g”	3ª
1	“e”	4ª
1	“n”	5ª
2	“h”	1ª

Tabela 3.2- Tabela intermediária utilizada pelo exemplo de transposição. (continuação)

GRUPO	LETRA	POSIÇÃO NO GRUPO
2	“a”	2ª
2	“r”	3ª
2	“i”	4ª
2	“a”	5ª
3	“ “	1ª
3	“d”	2ª
3	“a”	3ª
3	“ ”	4ª
3	“c”	5ª
4	“o”	1ª
4	“m”	2ª
4	“p”	3ª
4	“u”	4ª
4	“t”	5ª
5	“a”	1ª
5	“ç”	2ª
5	“a”	3ª
5	“o”	4ª
5	“y”	5ª

Com o acréscimo da letra “y” no grupo 5, foram formados cinco grupos com cinco componentes cada. Em seguida, será feita uma permutação das letras de acordo com suas posições dentro dos grupos, passando a serem dispostas da seguinte forma:

3ª - 1ª - 5ª - 2ª - 4ª posição. Na Tabela 3.3 é ilustrada a nova ordenação das letras nos grupos.

Tabela 3.3 – Tabela final utilizada pelo exemplo de transposição.

GRUPO	LETRA	POSIÇÃO NO GRUPO
1	“g”	3ª

Tabela 3.3 – Tabela final utilizada pelo exemplo de transposição. (continuação)

GRUPO	LETRA	POSIÇÃO NO GRUPO
1	“e”	1ª
1	“n”	5ª
1	“n”	2ª
1	“e”	4ª
2	“r”	3ª
2	“h”	1ª
2	“a”	5ª
2	“a”	2ª
2	“i”	4ª
3	“a”	3ª
3	“ “	1ª
3	“c”	5ª
3	“d”	2ª
3	“ ”	4ª
4	“p”	3ª
4	“o”	1ª
4	“t”	5ª
4	“m”	2ª
4	“u”	4ª
5	“a”	3ª
5	“a”	1ª
5	“y”	5ª
5	“ç”	2ª
5	“o”	4ª

A mensagem cifrada fica então “gennerhaaia cd potmuaayço”. Para decifrar a mensagem basta reordenar as letras novamente.

Os sistemas de substituição têm como princípio a troca de componentes da mensagem original por outros componentes previamente escolhidos, ou seja, uma

tabela com valores pré-determinados para cada elemento é montada ou o valor é calculado de forma dinâmica, aumentado o grau de confusão do texto. Como exemplo, um sistema de substituição simples é mostrado. A mensagem a ser criptografada será “texto de exemplo”. As letras que formam essa mensagem são *t, e, x, o, d, m, p, l*. Atribuindo símbolos a cada uma dessas letras formaremos a Tabela 3.4.

Tabela 3.4 – Tabela utilizada no exemplo de substituição.

Letra	Símbolo
t	*B2*
e	C4
x	E55
o	F16
d	J9
m	H10
p	Z1
l	M11

Substituindo cada letra da mensagem por seu respectivo símbolo formaremos a seguinte mensagem cifrada:
“B2C4E55B2F16 J9C4 C4E55C4H10Z1M11F16”. Para reordenar a mensagem basta substituir cada símbolo por sua letra correspondente.

Quanto à manipulação dos bits, os sistemas podem ser classificados como sistemas de cifragem de blocos e sistemas de cifragem de fluxo. Sistemas de cifragem de blocos manipulam e codificam os bits em conjuntos ou blocos que podem variar de acordo com a implementação (64,128,256... bits). Sistemas de cifras de fluxo manipulam e codificam cada byte do texto claro individualmente.

Os sistemas criptográficos podem ainda ser classificados quanto às suas chaves de encriptação e decriptação como sendo simétricos ou assimétricos. Sistemas simétricos são aqueles em que a chave de codificação e a chave de decifragem são as mesmas. Sistemas assimétricos ou de chave pública são os

que possuem uma chave para criptografar e outra chave diferente para decifrar a mensagem.

3.1.1 – Panorama histórico da criptografia

A criptografia é utilizada á bastante tempo , sempre com o principal intuito de tornar ininteligível uma mensagem. Para o bom entendimento sobre a lógica dos métodos criptográficos atuais e sobre a causas que levaram a adoção de certas metodologias com relação a outras, se faz necessário uma observação sobre o panorama histórico da criptografia, ilustrado na **Tabela 3.5 .**

Tabela 3.5 – Panorama histórico da criptografia

(Adaptado de: [WEB11])

ANO / PERÍODO	ACONTECIMENTO HISTÓRICO	LOCAL
Por volta de 1900a.C.	Primeiro registro de uso da criptografia.	Egito
50 a.C.	Julio César utiliza sua cifra de substituição.	Império Romano
718 d.C	Al-Khalil escreve “O livro das mensagens criptográficas” onde propõe um método de criptoanálise conhecido como método da palavra provável.	Mundo Árabe
1586 d.C.	Blaise de Vigenère publica um livro onde discute vários métodos de cifragem.	Europa
1854 d. C.	Charles Babbage, quebra a cifra de Vigenère.	Europa
1927 a 1933	Uso de mensagens criptografadas para comunicação entre navios com contrabando de run.	Estados Unidos
1960	O Dr. Horst Feistel desenvolve a cifra Lucifer . Vários algoritmos se baseiam nesta cifra que mais tarde foi chamada de cifra de Feistel.	Estados Unidos
1974	É divulgado o algoritmo DES.	Estados Unidos

Tabela 3.5 – Panorama histórico da criptografia

(continuação)

ANO / PERÍODO	ACONTECIMENTO HISTÓRICO	LOCAL
1977	É divulgado o algoritmo RSA.	Estados Unidos
1987	Divulgação do algoritmo RC4, desenvolvido por Ron Rivest.	Estados Unidos
2001	É proposto o algoritmo AES, vencedor de um concurso com finalidade de escolher um novo padrão criptográfico para os Estados Unidos.	Estados Unidos

Os fatos históricos citados tem o objetivo de ilustrar a evolução da criptografia durante os anos e servem para justificar a utilização do uso de chaves na implementação do algoritmo proposto para este projeto. Vale lembrar que várias técnicas criptográficas desenvolvidas ao longo dos séculos foram perdidas ou não foram divulgadas seja por opção dos desenvolvedores para mantê-las mais seguras, seja pela não repercussão de tais métodos nos meios em que foram apresentados.

3.1.2 – Algoritmos criptográficos

Na análise dos acontecimentos históricos acima mencionados, podemos notar que a criptografia evoluiu de sistemas que se baseavam apenas na montagem da lógica de codificação das mensagens claras para sistemas os quais, além de possuírem uma boa metodologia de composição de cifra, utilizam também o conceito de chaves. As chaves são utilizadas como elementos norteadores para os algoritmos criptográficos, sendo que o resultado do processo de cifragem será intimamente dependente da chave de entrada utilizada no sistema.

A evolução de sistemas baseados na lógica da composição de cifra para sistemas baseados no uso de chaves veio da necessidade de se ter um sistema

mais flexível a ataques e da necessidade de se poupar esforços durante a comunicação com vários usuários. Vejamos uma situação hipotética:

Suponhamos que o indivíduo A deseja enviar mensagens a cinco indivíduos ao mesmo tempo, B,C,D,E e F, de forma que um indivíduo não pode ler a mensagem referente a outro. Porém, quando o indivíduo A envia uma mensagem para qualquer um dos cinco indivíduos, esta mensagem também chega aos outros quatro.

- Para resolução do problema usando a abordagem de sistemas baseados na composição de cifras, o indivíduo A deveria utilizar para enviar suas mensagens, 5 algoritmos criptográficos diferentes, cada um referente a um indivíduo receptor. Cada indivíduo receptor deveria possuir um algoritmo criptográfico próprio para decifrar as mensagens destinadas a ele, de forma que o algoritmo pertencente ao indivíduo B não decifraria as mensagens destinadas aos indivíduos C,D,E ou F e assim por diante.
- Utilizando um sistema baseado no uso de chaves, o problema seria resolvido da seguinte forma: o indivíduo A possuiria apenas um algoritmo criptográfico e utilizaria 5 chaves diferentes para criptografar as mensagens, sendo que cada chave estaria relacionada a um receptor. Cada indivíduo receptor possuiria uma chave própria para poder decifrar suas mensagens, de forma que, a chave do indivíduo C não decifraria as mensagens destinadas aos indivíduos B,D,E e F e assim por diante.

Na situação hipotética mostrada é notória a diferença de utilização de recursos computacionais existente entre os sistemas baseados na cifragem e os sistemas baseados no uso de chaves, visto que os primeiros requerem muito mais espaço para armazenamento, já que A deveria armazenar 5 algoritmos diferentes. Imaginemos uma situação onde A se comunicasse com 1000 usuários!

A grande maioria dos algoritmos de criptografia atuais utilizam a lógica de sistema de chaves para construírem a codificação.

3.1.2.1 - ONE-TIME-PAD

O algoritmo de one-time-pad ou algoritmo de cifragem de Vernan, é o algoritmo criptográfico considerado mais seguro entre todos os algoritmos criptográficos existentes e é praticamente inquebrável [WEB15]. Ele se baseia na utilização de uma chave formada por bits gerados aleatoriamente. Essa chave é então aplicada à mensagem original por meio de uma operação de “OU Exclusivo” ou XOR. O único problema desse método é que a chave gerada deve ter o mesmo tamanho da mensagem que será criptografada. Esse pequeno detalhe, torna a utilização desse algoritmo praticamente inviável.[WEB15]

O one-time-pad é um exemplo de um algoritmo que utiliza cifragem de fluxo, pois os bits são codificados individualmente. A implementação do algoritmo proposto neste projeto, também possui uma cifragem de fluxo, pois, em uma de suas etapas, as amostras do sinal são codificadas individualmente.

3.1.2.2 – RC4

Alguns algoritmos criptográficos atuais tentam reproduzir o princípio de funcionamento do one-time-pad, tendo em vista a grande segurança que esse sistema proporciona. Para isso, são utilizadas funções para geração de números com aspecto aleatório, que são chamados números pseudo-aleatórios. Um exemplo de algoritmo que utiliza as funções geradoras de números pseudo-aleatórios é o RC4.

O algoritmo RC4 foi desenvolvido por Ron Rivest em 1987, para ser usado pela empresa RSA. Segundo [WEB18], este algoritmo utiliza a seguinte técnica:

- Uma chave de entrada **K** é gerada. Sua notação é feita como a de um vetor;
- Um vetor **S** é preenchido com valores de 0 a 255;
- É feita a soma entre a variável auxiliar *j*, o valor de **S**[i] (**S** na posição *i*) e o valor da chave **K**[i] (**K** na posição *i*). Essa soma é armazenada na variável *j.* (j = j + **S**[i] + **K**[i]);

- É feita a troca entre os valores de **S**[i] (**S** na posição *i*) e **S**[j] (**S** na posição *j)* ;
- A função geradora de números pseudo-aleatórios incrementa em uma unidade a variável *i;*
- É feita a soma entre **S** [i] (**S** na posição *i*) e *j* e o resultado é armazenado em *j.* (j = j + **S**[i]);
- É realizada novamente a troca entre os valores de **S**[i] (**S** na posição *i*) e **S**[j] (**S** na posição *j*);
- É calculada a saída aplicando-se a operação de XOR (OU-Exclusivo) entre o valor **S**{**S**[i] + **S**[j] } (**S** na posição **S**[i] + **S**[j]) e o valor do texto original na posição apontada pela variável auxiliar *k*;

Após esses passos é obtido como resultado um caractere cifrado da mensagem original. Essa operação é repetida para todos os elementos da mensagem.

O algoritmo proposto neste projeto, também faz uso da técnica de geração de números pseudo-aleatórios no processo de cifragem dos sinais de áudio.

3.1.2.3 – DES E TRIPLE – DES

Por muitos anos, o DES (Data Encryption Standard) foi o padrão de criptografia nos Estados Unidos e era usado em todo o mundo [Buchmann,2002]. Esse criptossistema é um dos sistemas da família das cifras de Feistel, pois utiliza a lógica de Feistel com algumas modificações.

As cifras de Feistel utilizam a lógica das cifras em bloco, tendo como base o Plaintext, bloco de dados da mensagem original de tamanho *2w*. Este bloco inicial é então subdividido em dois sub-blocos com o mesmo tamanho w, sendo o primeiro formado com os bits mais significativos a esquerda (será chamado de "L_0" de *Left*) e o segundo formado com os bits menos significativos do bloco (será chamado de "R_0" de *Right*).

Para a formação do bloco codificado , é feita a permutação entre L_0 e R_0, sendo que R_0 assume a posição L_1 no próximo bloco formado (L_1=R_0). O sub-

bloco R_1, será formado a partir de uma operação ***F*** entre L_0 , R_0 e uma chave criptográfica K_1. Tanto a operação ***F*** quanto a chave K_1, variam de acordo com a lógica do algoritmo implementado. A Figura 3.1 ilustra essas operações.

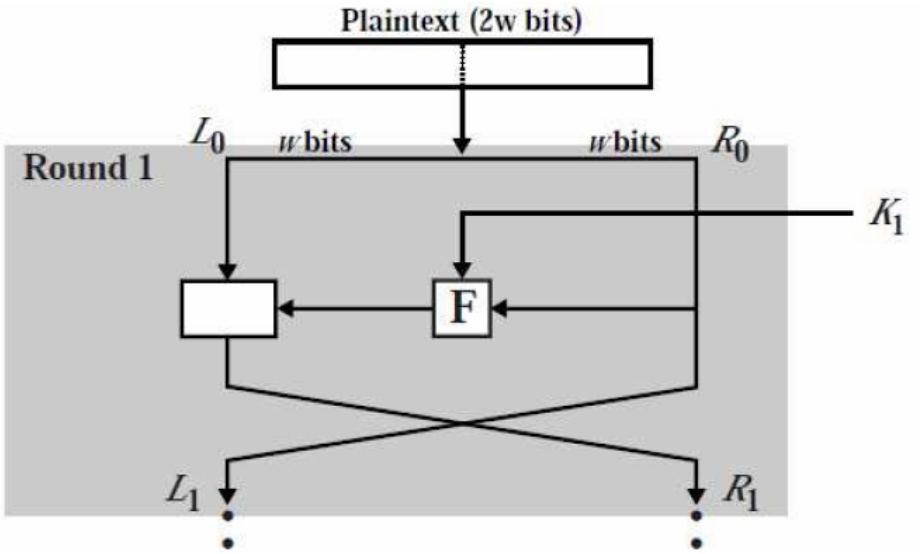


Figura 3.1 – Lógica de montagem dos blocos de Feistel.
(Fonte: [WEB13])

O DES funciona da seguinte maneira: primeiramente a mensagem original é particionada em blocos de 64 bits cada. Cada bloco sofre uma permutação inicial nos seus elementos. Em seguida , o algoritmo utiliza uma chave de entrada de 64 bits. Destes 64 bits somente 56 serão usados efetivamente no processo de codificação, pois a cada conjunto de 8 bits da chave, um bit é usado para controle de paridade, par ou ímpar dependendo da implementação.

Paridade é um método usado para saber se o número de bits “1” existentes em um bloco é par ou ímpar. Se o número de bits “1” no bloco for ímpar, o bit de paridade recebe valor “1”.Caso contrário, seu valor será “0”. como exemplo, pegaremos a seqüência de 7 bits “ 0 1 0 1 0 0 1 ”. O número de bits “1” dessa seqüência é ímpar (3 bits). Logo, o bit de paridade para essa seqüência receberá “1” como valor : “ 0 1 0 1 0 0 1 1”. Para o conjunto “ 0 0 0 1 1 0 0 ”, o bit de

paridade assumiria o valor “0”, pois o número de bits “1” da mensagem é par (2 bits), gerando a seguinte seqüência : “ 0 0 0 1 1 0 0 0 ”. Esse tipo de paridade é denominado paridade par. Outro tipo de paridade é a paridade ímpar, na qual o bit de paridade recebe valor “1” quando o número de uns do conjunto avaliado é par.

Voltando para a explicação do DES, a partir da chave de 64 bits (na verdade apenas 56 serão utilizados) são geradas 16 novas chaves de 48 bits cada.

Depois da geração das 16 chaves, por meio de 16 iterações, o algoritmo aplica as chaves geradas ao bloco de mensagem permutado. O resultado da aplicação das chaves à mensagem sofre ainda uma última permutação , formando a mensagem codificada final. Essa operação pode ser vista na Figura 3.2.

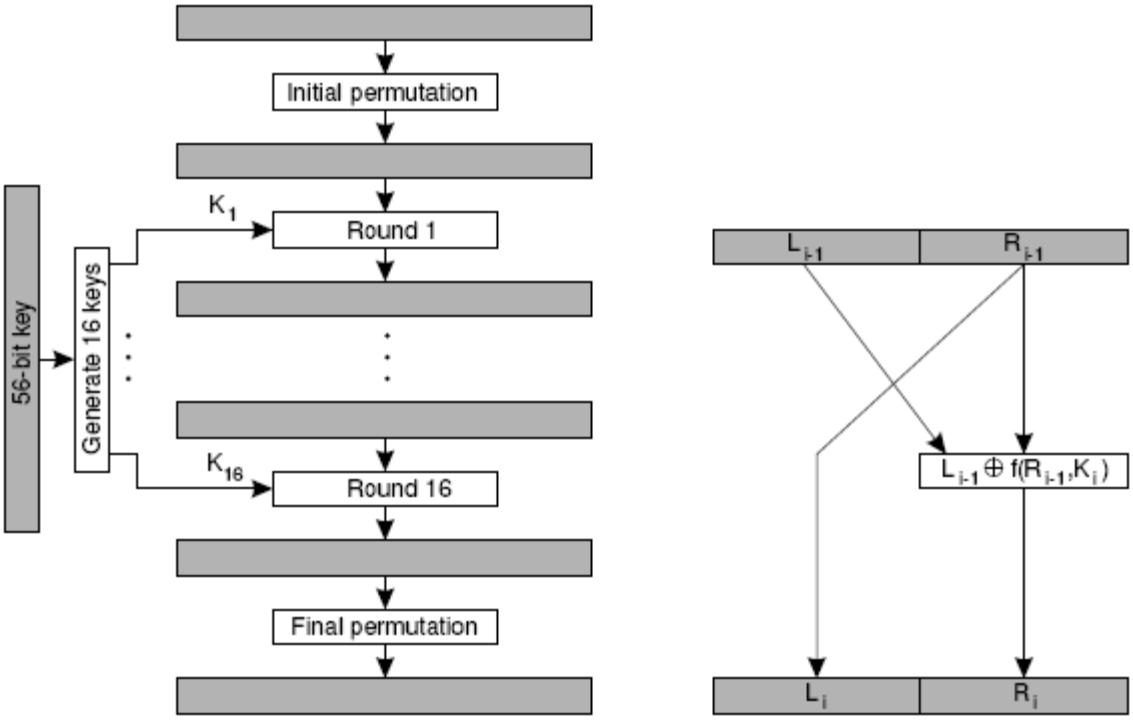


Figura 3.2 – Esquemático de funcionamento do DES.
(Fonte: [WEB13])

O DES é um algoritmo de chave simétrica, ou seja, a chave usada para decifrar a mensagem é a mesma que foi usada na entrada do sistema para criptografá-la.

Uma das vantagens do DES é que qualquer mudança mínima na entrada do sistema provoca uma grande mudança na mensagem criptografada. Isso é possível devido à lógica robusta usada no processo de cifragem da mensagem.

Em um método de ataque de “força bruta”, onde todas as chaves possíveis chaves para o sistema são geradas e testadas, seriam geradas

2^{56} = 72.057.594.037.927.936 chaves.

O algoritmo DES foi quebrado em 1997 utilizando um computador produzido especialmente para este fim.

Para um melhor entendimento sobre a lógica do algoritmo de DES e sobre o mecanismo de funcionamento de suas funções internas vide [WEB11] e [Buchmann,2002].

O Triple-Des utiliza o mesmo mecanismo do Des só que utilizando 2 ou 3 chaves. Primeiramente o texto claro é criptografado como no Des original. Em seguida, é feita a decifragem da mensagem, porém usando uma segunda chave, diferente da que foi utilizada inicialmente, o que provoca um embaralhamento maior na mensagem. No último passo, aplica-se novamente o processo de codificação inicial, usando como texto claro o resultado obtido no passo anterior. Para o último passo, utiliza-se a primeira chave ou uma terceira chave diferente das duas anteriores.

O algoritmo proposto neste projeto executa uma permutação inicial em blocos de amostras do sinal de áudio, de maneira similar a ocorrida no algoritmo DES.

3.1.2.4 – AES

O AES (Advanced Encryption Standard) surgiu de um concurso promovido pelo NIST (National Institute of Standards and Technology)dos E. U.A, com início em 1997, com o objetivo de ser o novo padrão de criptografia usado nos Estados Unidos, já que o DES havia sido quebrado. No final do concurso em 2001, o

algoritmo escolhido foi o proposto por Vincent Rijmen e Joan Daemen. A esse algoritmo foi dado o nome de algoritmo de Rijndael em homenagem aos autores.

O AES, assim como o DES, é um algoritmo simétrico de cifragem em blocos. No entanto, ele se diferencia do anterior por ser mais flexível , podendo utilizar blocos de dados com tamanho de 128 bits e chaves que podem ser de 128, 192 ou 256 bits.

O algoritmo AES segue os seguintes passos:

- A partir da chave de entrada do processo, são geradas outras chaves dependendo do tamanho da primeira. Para uma chave de entrada de 128 bits, 9 novas chaves são geradas; para chaves de entrada de 192 bits, 11 são geradas e para chaves de entrada de 256 bits 13 outras são geradas. O número de iterações do algoritmo depende do número de chaves total usadas no processo, que por sua vez depende do tamanho da chave de entrada utilizada. Cada chave gerada no algoritmo será usada para uma nova repetição do processo. Para chaves de 128 bits na entrada, 10 iterações serão realizadas (Uma chave de entrada + 9 chaves geradas); para chaves de 192 bits na entrada, 12 iterações; para chaves de 256 bits na entrada, 14 iterações.
- É feita uma operação de XOR (OU-Exclusivo) entre o bloco de mensagem e a chave de entrada;
- Os blocos de mensagem de 128 bits passam por um processo de substituição, onde cada byte é substituído por seu valor equivalente dentro da S-Box. As S-Box são tabelas que armazenam os valores usados na substituição. Elas são formadas a partir de uma lógica específica, que depende do algoritmo criptográfico implementado;
- O bloco resultante da ação anterior, passa por um processo de transposição, no qual os bytes que formam o bloco são permutados entre si.
- No bloco gerado, são formados conjuntos de bytes com tamanho dependente do tamanho da chave de entrada, e é aplicada uma operação de XOR entre esses bytes e uma constante pré-definida no algoritmo;
- O processo é repetido para todas as chaves geradas.

O AES é projetado para ser eficazmente implementado em software e hardware, sendo essa característica uma das suas principais vantagens. Até os dias atuais, nenhum método de criptoanálise real conhecido conseguiu quebrar este algoritmo. [WEB13]

Assim como o algoritmo AES, o algoritmo de cifragem de áudio implementado neste projeto, também faz uso de operações de XOR durante os processos de cifragem e decifragem.

3.1.2.5 – BLOWFISH

O Blowfish foi proposto pela primeira vez em 1993 por Bruce Schneier, como uma alternativa aos algoritmos criptográficos existentes. Desde o seu surgimento, o Blowfish vem sendo analisado por uma série de estudiosos em criptografia, entretanto nenhum método de criptoanálise conhecido, obteve sucesso na tentativa de quebrar esse algoritmo.

O Blowfish utiliza cifragem em blocos de 64 bits, como no DES, e chaves que variam de 32 a 448 bits [WEB14].Ele também se baseia nas cifras de Feistel, como o DES, e utiliza S-box variável, dependente da chave de entrada do processo.

3.1.2.6 – RSA

O RSA é um dos algoritmos criptográficos de chave pública mais populares e eficientes, visto que, também é um algoritmo que ainda não foi quebrado por nenhum método de criptoanálise conhecido (2006). Seu nome é uma homenagem aos seus criadores, Ron **R**ivest, Adi **S**hamir e Len **A**dleman.

O RSA trabalha com blocos e chaves de tamanho variável, como o AES , mas diferentemente dos algoritmos anteriores, utiliza o conceito de chave pública ou assimétrica, ou seja, a chave que é utilizada para desfazer a criptografia é diferente da que foi usada para criptografar a mensagem.

A lógica de segurança utilizada no RSA provém da geração de números primos grandes (100 dígitos) e da dificuldade de fatoração desses números.

3.2 – Scrambler

A denominação Scrambler ou Misturador é geralmente empregada para discriminar os algoritmos utilizados no processo de cifragem de sinais de áudio, sendo portanto um tipo especial de criptografia. Em outras palavras, o Scrambler pode ser entendido como o conjunto de técnicas criptográficas aplicadas aos sinais sonoros. Em alguns contextos, a terminologia Scrambler faz referência ao uso da criptografia aplicada a outros tipos de sinais, tais como sinais luminosos (fibra ótica), sinais televisivos, sinais de rádio.

As técnicas adotadas na implementação de Scramblers são similares as técnicas usualmente implantadas na criptografia convencional. Porém, algumas particularidades são inerentes aos misturadores, pois advém da própria atividade de manipulação dos sinais de áudio.

Por exemplo, as técnicas de substituição e transposição utilizadas nos Scramblers seguem o mesmo princípio das utilizadas na criptografia tradicional, mas existem alguns pontos de peculiaridade.

Na transposição em misturadores, o sinal pode ser misturado no domínio da freqüência ou no domínio do tempo; o tamanho do bloco de dados a ser misturado deve ser cuidadosamente escolhido, sob pena de não causar efeito de dispersão significativo ao sinal claro [WEB16].

Na substituição em sinais de áudio, não tem efeito para a cifragem alguns tipos de alterações lineares de amplitude ou de modulação [WEB16].

Outras técnicas de Scrambler se baseiam no fenômeno da interferência sonora, que ocorre quando há o encontro de duas ou mais ondas [WEB19]. O

resultado da interferência sonora é a formação de uma nova onda sonora completamente diferente das ondas anteriores.

O método da interferência sonora é o método utilizado na implementação do algoritmo proposto para este projeto, no qual são acrescidos sinais de ruído ao sinal originalmente claro, de forma a torná-lo ininteligível. A formação das ondas de ruído é feita por meio da geração de seqüências de números pseudo-aleatórios. Os ruídos gerados são acrescidos ao sinal original através da aplicação das operações de XOR. Uma abordagem detalhada sobre a geração de números pseudo-aleatórios e sobre as operações de XOR utilizadas na implementação é feita a seguir.

3.2.1 – Geração de números aleatórios

Os números aleatórios são de grande importância na computação, tendo sua utilização aplicada em simulações, testes, jogos, análises populacionais, na criptografia e em muitos scramblers.

Como exemplo de fontes geradoras de números aleatórios , podem ser citadas a escolha de um envelope numerado em uma urna, o lançamento de um dado, o uso de uma roleta, o nível de ruído medido em um resistor em funcionamento entre outras. [Rosa,Junior,2002]

Em algumas simulações, testes e também em alguns algoritmos criptográficos, os números aleatórios podem ocasionar um problema, pois uma seqüência numérica anteriormente gerada, não pode ser recuperada (a não ser pelo acaso), ou seja, uma situação criada pela geração de números aleatórios, não pode ser remontada em um momento posterior de forma controlada. No entanto, existem funções matemáticas que geram seqüências numéricas similares às seqüências de números aleatórios, sendo chamadas de funções geradoras de números pseudo-aleatórios. [Rosa,Junior,2002]

As funções geradoras de números pseudo-aleatórios concebem suas seqüências numéricas a partir de uma semente. A semente é um número inicial passado a função. Portanto, se houver a necessidade de se reproduzir uma

seqüência numérica gerada por estas funções, basta aplicar a mesma semente utilizada para sua geração e o conjunto de números será recuperado.

Existe uma infinidade de funções destinadas a geração de números pseudo-aleatórios. No algoritmo implementado, esses números são gerados por meio do método linear congruente.

O método linear congruente é dado pela seguinte relação recursiva:

$$x_{n+1} = (a * x_n + c) \bmod m \; ; \; n \geq 0 \qquad \textbf{(3.1)}$$

Nesta relação, $\mathbf{x_{n+1}}$ é o valor a ser gerado, **a** é o multiplicador, $\mathbf{x_n}$ é o valor anterior a $\mathbf{x_{n+1}}$, **c** é o incremento e **m** é o módulo. A função z = y **mod** x, retorna a z o valor do resto da divisão entre y e x, onde y é o numerador e x o denominador.

O valor inicial x_0 é a semente da função. A escolha dos valores de **a**, **c** e **m** é o que afeta drasticamente a periodicidade da seqüência gerada, ou seja, dependendo dos valores assumidos por estes parâmetros a seqüência pode se repetir muito rapidamente ou não. O período máximo para uma função linear congruente é limitado pelo valor de **m**.

Em [Rosa,Junior,2002] é mostrado um teorema que nos indica qual a melhor escolha para **a**, **c** e **m**, de forma a fazer com que a função linear congruente tenha período **m.**

Teorema A - O método linear congruente tem um período **m** se e somente se:

i) **c** e **m** são primos entre si.

ii) b = **a** - 1 é um múltiplo de p, para todo p primo divisor de **m**

iii) b é um múltiplo de 4, se **m** é um múltiplo de 4.

Seguindo os preceitos apontados no Teorema A para a escolha dos parâmetros **a**, **c** e **m**, podem ser construídas funções lineares congruentes que possibilitem a formação de seqüências aleatórias com períodos escolhidos de acordo com a necessidade exigida pela aplicação.

3.2.2 – Operações de XOR (OU-Exclusivo)

As operações de XOR (OU-Exclusivo) são operações bastante utilizadas para a codificação e decodificação das mensagens nos processos de criptografia e Scrambler, seja em cifras de blocos como o AES, seja em cifras de fluxo como o RC4 ou em algoritmos de Scrambler baseados na interferência sonora. A vasta utilização destas operações é explicada pela característica peculiar na maneira como esta operação é desfeita.

A operação de XOR trabalha a nível de bits e seu símbolo é representado por $\oplus$. Sua tabela verdade é mostrada na Tabela 3.6.

Tabela 3.6 – Tabela verdade para as operações de XOR.

Primeira parcela	0	0	1	1
Símbolo da operação de XOR	$\oplus$	$\oplus$	$\oplus$	$\oplus$
Segunda parcela	0	1	0	1
Resultado	**0**	**1**	**1**	**0**

Analisando a Tabela 3.6, vemos que o bit resultante da operação de OU-Exclusivo somente receberá valor 1, se os bits avaliados forem diferentes.

A característica dessa operação que a torna tão popular, é a seguinte: se fazemos a operação de XOR entre dois números **a** e **b** resultando em **c**, a operação de XOR feita entre **c** e **b** resulta em **a**, e feita entre **c** e **a** resulta em **b**, como é mostrado na Equação 3.2a.

$$\mathbf{a} \oplus \mathbf{b} = \mathbf{c} \Rightarrow \mathbf{c} \oplus \mathbf{a} = \mathbf{b} \quad \text{e} \quad \mathbf{c} \oplus \mathbf{b} = \mathbf{a} \qquad \textbf{(3.2a)}$$

Por exemplo, se tomamos os números 4 (100 em binário) e 6 (110 em binário) e fazemos a operação de XOR entre eles temos: 100 $\oplus$ 110 = 010 (2 em

decimal). Pegando o resultado da operação e aplicando XOR entre ele e um dos elementos envolvidos anteriormente, encontramos o outro elemento, ou seja:

$$4 \oplus 2 = 100 \oplus 010 = 110 = 6; \quad \textbf{(3.2b)}$$

$$6 \oplus 2 = 110 \oplus 010 = 100 = 4; \quad \textbf{(3.2c)}$$

Essa característica é importante para a criptografia, pois se uma operação de XOR é aplicada entre uma seqüência numérica clara (**Sc**) e uma seqüência numérica aleatória (**Sa**) resultando em **R**, para que seja recuperada a seqüência clara, basta apenas aplicar a operação de XOR entre **R** e **Sa**.

3.3 – Esteganografia

A esteganografia (do grego “steganós”, oculto, escondido - “grafia”, escrita) é a arte de ocultar mensagens ou arquivos dentro de outras mensagens ou arquivos. Por conseguinte, os algoritmos esteganográficos têm por principal objetivo embutir uma mensagem clara em uma outra mensagem ou arquivo hospedeiro, e fazer com que ela não seja percebida por um observador intruso, caso ele consiga capturar a mensagem hospedeira e analisá-la.

Várias são as técnicas usadas pelos algoritmos esteganográficos para atingirem suas metas de inserção oculta de mensagens em arquivos, tais como manipulação de caracteres, LSB ou “inserção no bit menos significativo”, inserção de blocos não significativos (utilizada em arquivos de grande extensão como filmes), filtragem e mascaramento (que podem utilizar filtros passa baixa e bits mais significativos) e uso de transformadas (de Fourier ou transformada Z).

4- IMPLEMENTAÇÃO DO ALGORÍTMO

Com o objetivo de aplicar a técnica de Scrambler à sinais de áudio, abordando tanto a transposição quanto a substituição, foi implementado um algoritmo cujo funcionamento é detalhado nas seções seguintes deste capítulo.

4.1 – Características

O algoritmo desenvolvido foi implementado utilizando-se o software Matlab da MathWorks, em sua versão 7.0.1.2747, Release 14, Service Pack 1.

O Matlab é uma ferramenta de programação que tem seu direcionamento voltado para o desenvolvimento de aplicações técnicas, e para a montagem de testes e ambientes de simulação, sendo por isso apreciado por estudantes das mais variadas áreas. Na Figura 4.1 é mostrada a tela inicial do Matlab 7.0.1.

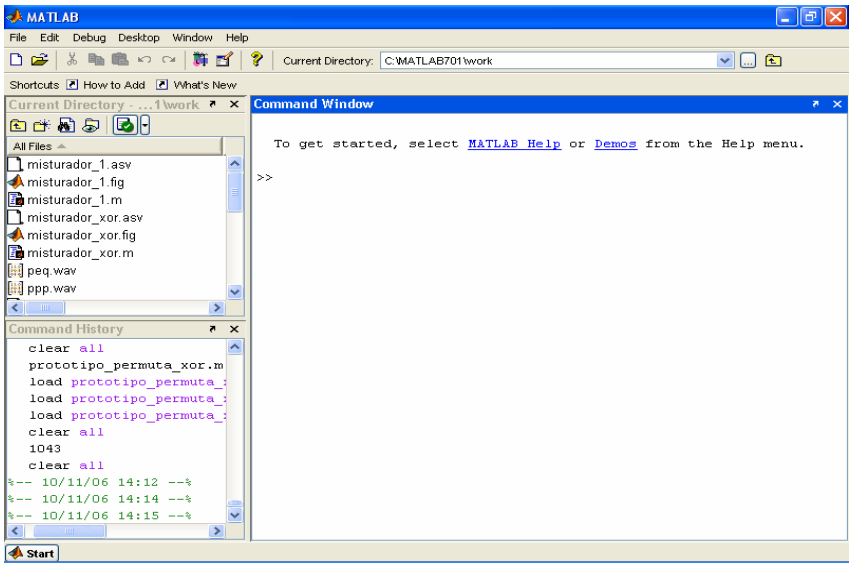


Figura 4.1 – Tela inicial do Matlab 7.0.1.

O algoritmo utiliza os arquivos no formato Wave para ler e armazenar os sinais de áudio. Os arquivos Wave utilizados devem estar no formato “PCM”, utilizando um canal (mono) e 8 ou 16 bits de codificação. O valor da taxa de amostragem é indiferente na implementação da cifração. O detalhamento sobre a estrutura dos arquivos Wave é abordado na seção 2.3.5. O esquemático do projeto pode ser visto no Apêndice B.

4.2 – Visão Geral do Algoritmo

Uma visão geral sobre o funcionamento do algoritmo implementado é ilustrada nos fluxogramas da Figura 4.2 e 4.3 que representam, respectivamente, o processo de codificação e o processo de decodificação.

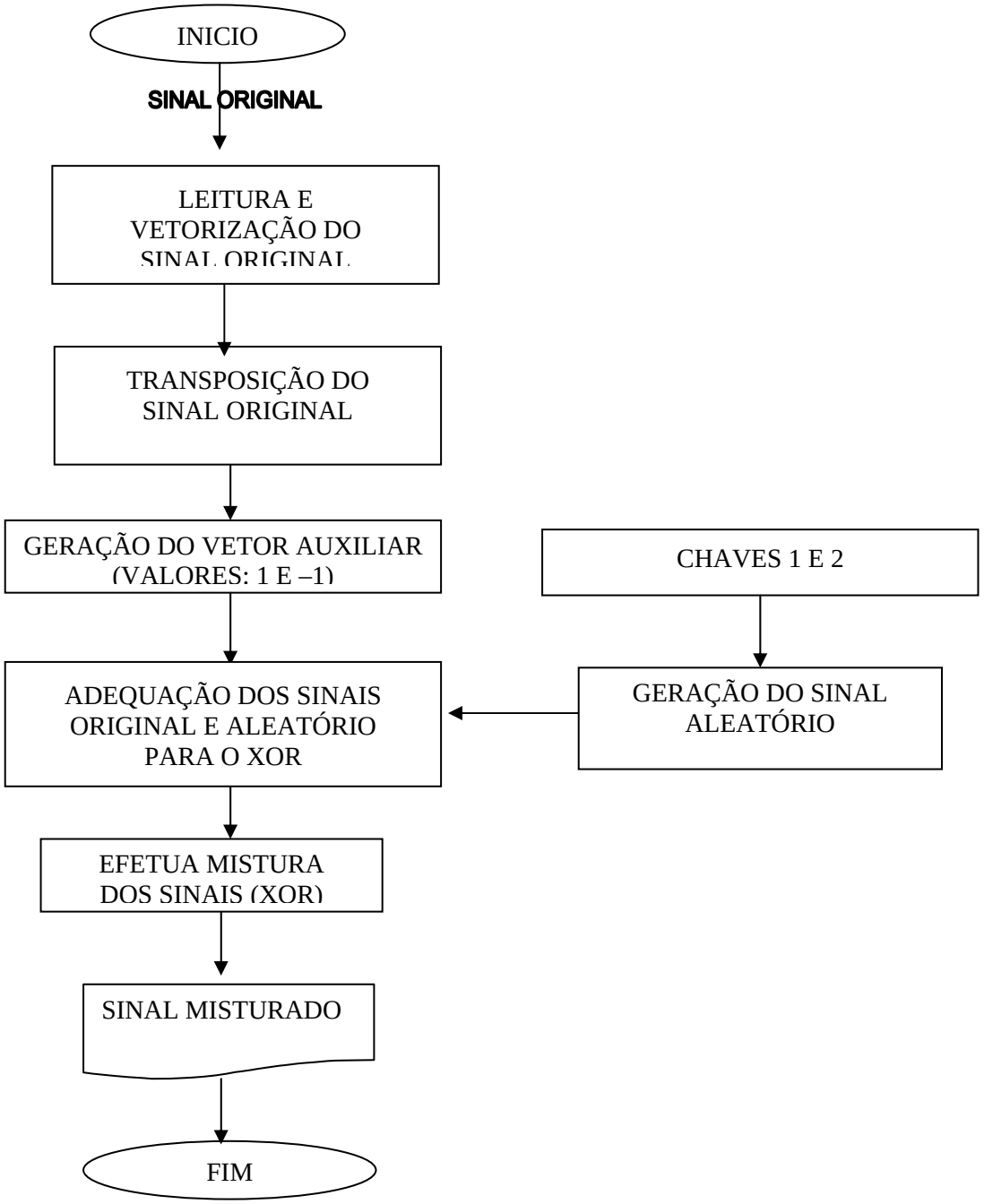


Figura 4.2 – Fluxograma do processo de codificação do algoritmo.

O processo de codificação implementado segue os seguintes passos:

- O sinal original é lido e representado por um vetor de amostras (vetorizado). Nesse vetor, as amostras são representadas por valores na faixa entre -1 e 1, podendo assumir até 4 casas decimais;
- É aplicada a operação de transposição no sinal vetorizado, resultando em um vetor permutado. Esse vetor contém as mesmas amostras contidas no sinal vetorizado, porém elas estão reordenadas;
- È montado um vetor auxiliar de mesmo tamanho do vetor permutado, porém formado pelos valores **1** e **–1**. O vetor auxiliar recebe o valor **1** na posição ***i*** se e somente se a amostra do sinal permutado na posição ***i*** for positiva. Caso o valor da amostra na posição ***i*** seja negativa, o valor assumido pelo vetor auxiliar nessa mesma posição será **–1**.
- O sinal permutado tem os valores de suas amostras multiplicados por 10000 e transformados em valores positivos. Essa etapa é necessária para a adequação do sinal às operações de XOR no Matlab, pois estas são realizadas apenas entre números inteiros positivos. Outra adequação feita é a de manipulação das amostras do vetor permutado com base no vetor auxiliar de 1`s e -1`s formado anteriormente, de modo que: caso o valor existente na posição *i* do vetor auxiliar seja **1**, implica que o valor contido na posição *i* do vetor permutado será múltiplo de 2. Se o valor da posição *i* do vetor auxiliar for **–1**, o valor na respectiva posição no vetor permutado não será um número múltiplo de 2. Após este passo, o vetor de amostras permutado estará pronto para as operações de XOR;
- Paralelamente aos passos anteriormente mencionados, o processo de criação dos sinais aleatórios ocorre. Neste processo, primeiramente, são geradas duas chaves por meio da função rand(), que é uma das funções de geração de números randômicos no Matlab. A chave 1 é formada por 17 algarismos, utilizando 140 bits para sua representação. A chave 2 é formada por 16 algarismos, utilizando 64 bits para sua representação.
- As células de aleatoriedade apontadas pela chave 2 produzem 16 seqüências de números pseudo-aleatórios. Essas seqüências são formadas usando como

semente os números contidos nas 16 primeiras posições da chave 1. Os valores dos elementos das seqüências pseudo-aleatórias geradas estarão na faixa entre 0 e 1.;

- É feita a adequação dos sinais aleatórios gerados para as operações de XOR. Para isto, basta multiplicar as amostras dos sinais aleatórios por 10000, transformando-as em números positivos inteiros . Essa multiplicação é necessária, pois as operações de XOR são realizadas somente entre números inteiros positivos e o sinal aleatório é representado por números decimais positivos com até 4 casas de precisão;
- Depois da adequação dos sinais, operações sucessivas de XOR são aplicadas entre o sinal permutado modificado e as seqüências numéricas geradas. Como esta é uma operação que ocorre a nível de bits, significa dizer que as seqüências aleatórias e o sinal permutado foram traduzidos em bits antes da operação;
- O sinal resultante tem o valor de suas amostras dividido por 10000. Isso é necessário pois, para a manipulação dos arquivos Wave, o Matlab trabalha com valores de amostras na faixa entre –1 e 1.

- O sinal resultante pode ser ouvido e armazenado em um arquivo. As chaves resultantes devem ser gravadas para o processo de decodificação;

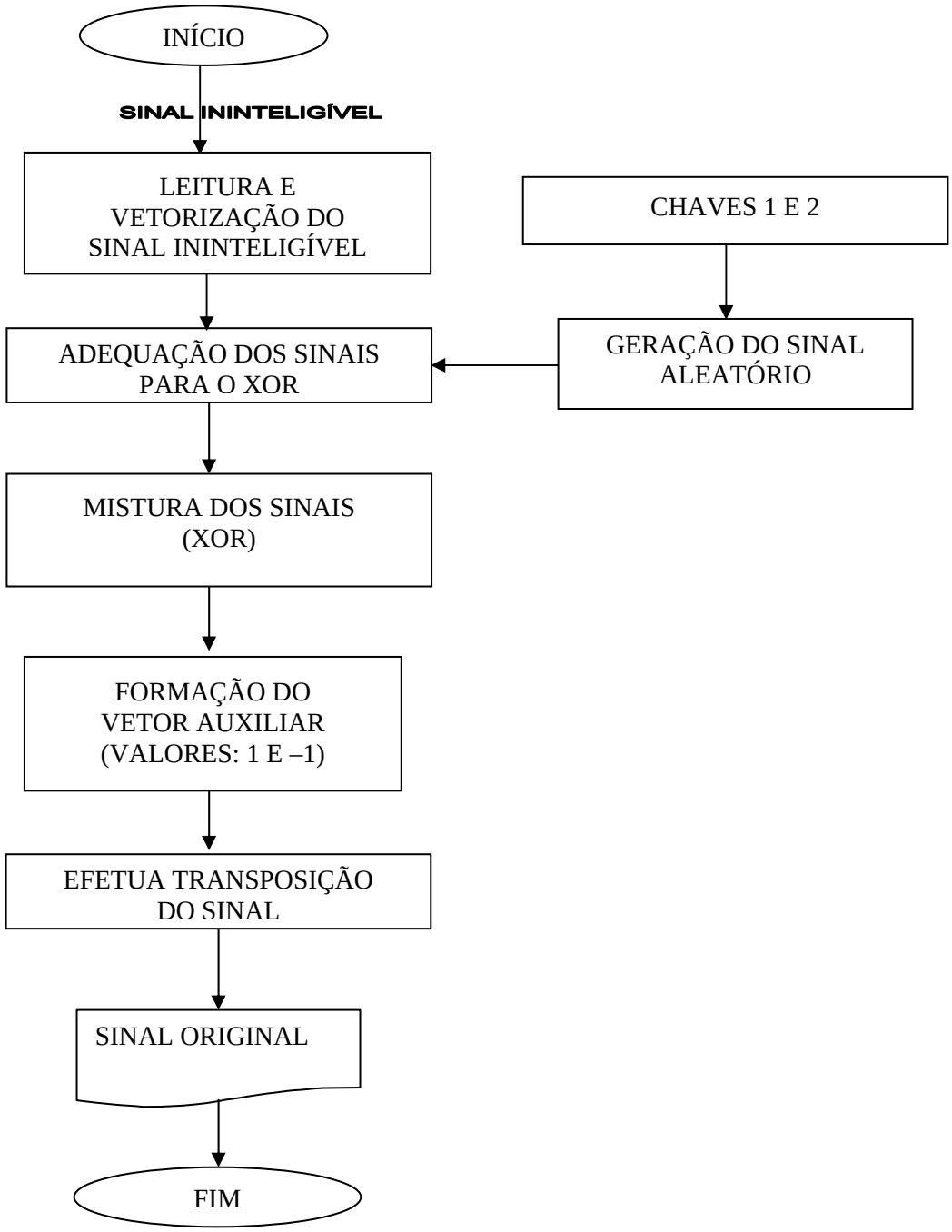


Figura 4.3 – Fluxograma do processo de decodificação do áudio.

O processo de decodificação implementado perfaz o caminho inverso da codificação, da seguinte forma:

- O sinal misturado pela codificação é lido e representado por meio de um vetor de amostras (vetorizado);
- As chaves geradas no processo de codificação devem ser inseridas pelo usuário;
- É feita a geração das seqüências de números pseudo-aleatórios, baseando-se nos valores contidos nas chaves inseridas pelo usuário;
- O sinal misturado e o sinal aleatório são multiplicados por 10.000. Essa multiplicação é necessária para a aplicação das operações de XOR, que, no Matlab, são realizadas com números inteiros.
- São aplicadas sucessivas operações de XOR entre o sinal misturado e as seqüências numéricas geradas, obtendo-se o sinal permutado;
- As amostras do vetor oriundo das operações de XOR (sinal transposto) recebem sinal negativo ou positivo conforme a seguinte abordagem: se o valor da amostra for um número múltiplo de 2, ela recebe sinal positivo. Caso contrário, recebe sinal negativo;
- O sinal resultante é dividido por 10.000. Isso é feito para adequar os valores de suas amostras aos valores manipuláveis pelo Matlab, para o tipo de arquivo em questão, Esses valores devem estar entre 1 e -1;
- É aplicada a transposição inversa ao sinal permutado, deixando-o em sua forma original;
- O sinal recuperado pode ser ouvido e gravado em um arquivo Wave.

Os códigos implementados para os processos de cifração e decifração dos sinais de áudio são apresentados no Apêndice A.

4.3 – Operações e Funções

Para compor a lógica implementada no algoritmo, as operações de transposição ou permutação e as operações de XOR (OU-Exclusivo) foram utilizadas, além da função linear congruente de geração de números pseudo-aleatórios. A maneira como cada uma delas é implementada é abordada a seguir.

4.3.1 – Transposição na implementação

No algoritmo, para o processo de codificação, a etapa de transposição perfaz os seguintes passos :

- O sinal original é dividido em **G** grupos de tamanhos iguais. Cada grupo G é subdividido em 22 subgrupos com tamanho **h**. O tamanho **h** de cada subgrupo é calculado segundo a função mostrada abaixo :

h = inteiro(Na/22);
Enquanto h>3000
h = inteiro(h/1.5);
Fim-enquanto;

Nesse algoritmo, **h** representa o tamanho do subgrupo e **Na** representa o número total de amostras do arquivo Wave. A função **z = inteiro(x/y)**, significa que **z** assumirá o valor inteiro do resultado da divisão entre **x** e **y**.

Por exemplo, em um arquivo wave de 150.000 amostras o tamanho **h** será,

h = inteiro(150000/22); => h=6818

Enquanto h>3000
h = inteiro(h/1.5); => 1ª iteração: h= inteiro(6818/1.5) = 4545 > 3000
=> 2ª iteração: h= inteiro(4545/1.5) = 3030 > 3000

=> 3ª iteração: h=inteiro(3030/1.5) = 2020 < 3000

Fim-enquanto;

h = 2020

Logo, para um arquivo com tamanho total de 150000 amostras, teremos cada subgrupo com tamanho **h** igual a 2020 amostras. Em um arquivo Wave utilizando 22000 Hz de taxa de amostragem, esse tamanho é equivalente a 6.81 segundos.

- Depois de calculado o tamanho **h** dos subgrupos, o número total de amostras(**Na**) do sinal deve ser ajustado de forma a obedecer a seguinte relação :

  Na =n * (22 * h) , onde **n** é um número inteiro. **(4.1a)**

Isto significa dizer que **Na** deve ser múltiplo do produto **"22 * h".** Esta operação garante que todas as amostras possam ser permutadas. Para isso, é acrescentado um número **q** de amostras ao sinal original, calculado segundo a Equação 4.2a**.**

q = 22 * h - resto(Na/(22*h)) , onde **(4.2a)**

q = número de amostras a serem acrescidas ao sinal;
Na = número de amostras total do sinal;
h = tamanho do subgrupo;
resto(x/y) retorna o resto da divisão entre x e y.

Pegaremos como exemplo, o mesmo sinal de 150.000 amostras tomado no exemplo anterior (**Na =**150000). Como calculado no exemplo anterior, **h=**2020 para este sinal. O número de amostras a serem acrescentadas ao sinal será:

q = 22 * 2020 – resto(150000/(22*2020)) = **27760** **(4.2b)**

Portanto o valor de amostras a serem acrescentadas ao sinal é 27760. O novo valor de amostras **(Na)** fica então:

$\mathbf{Na_{novo}} = \mathbf{Na_{antigo}} + \mathbf{q;} \Rightarrow \mathbf{Na_{novo}} = 150000 + 27760 = 177760$ **(4.3)**

Logo o novo número de amostras do sinal será 177760 que é um número múltiplo de "**22*h"**. Isto pode ser provado pela substituição dos valores na Equação **4.1a :**

177760 =n * (22 * 2020) => n = 4

As amostras adicionadas ao sinal terão valor igual ao valor da última amostra do sinal original. Isto é, se a última amostra do sinal original for 0,51, as amostras acrescentadas vão também assumir o valor 0,51.

- O passo seguinte é o que realmente caracteriza a transposição, pois o que foi feito até agora apenas serviu para adequar o sinal original, de forma a garantir maior eficiência na aplicação desta operação. Neste estágio, cada grupo **G** é tratado separadamente e é feita a troca das posições entre os seus subgrupos. As novas posições assumidas pelos subgrupos seguem a Tabela 4.1**.**

Tabela 4.1 – Tabela de permutação usada no processo de codificação.

22	6	19	13	10	12	3	17	5	15	4	21	16	8	2	20	14	11	7	1	18	9

Vamos analisar a tabela de permutação. Cada valor da tabela representa um subgrupo dentro do grupo G. Como o grupo G possui 22 subgrupos, cada subgrupo recebe um número entre 1 e 22 como é mostrado na Tabela 4.2.

Tabela 4.2- Estrutura de um grupo G.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Traçando um paralelo entre a estrutura dos grupos G's mostrada na Tabela 4.2 e a tabela de permutação representada na Tabela 4.1, podemos notar que no sinal resultante da permutação: o subgrupo 22 vai assumir a posição 1 dentro do seu grupo G; o subgrupo 6 vai assumir a posição 2 dentro do seu grupo G; o subgrupo 19 vai assumir a posição 3 dentro do seu grupo G e assim por diante até chegarmos ao subgrupo 9 que assumirá a posição 22 no seu grupo G.

É importante observar que a ordem dos grupos G's no sinal original e no sinal permutado é a mesma. O que muda é a ordem dos subgrupos dentro de cada grupo. O resultado desta etapa é o sinal permutado.

Na decodificação, ocorre a permutação direta, sem processos de preparação. Para esta etapa utiliza-se a tabela de permutação ilustrada na Tabela 4.3.

Tabela 4.3 – Tabela de permutação usada na decodificação.

20	15	7	11	9	2	19	14	22	5	18	6	4	17	10	13	8	21	3	16	12	1

Aplicando a permutação ao sinal codificado conforme a Tabela 4.3, obtém-se o sinal original, pois esta tabela desfaz a permutação feita no processo de codificação e com isso o processo de transposição é finalizado.

O procedimento de permutação adotado no protótipo pode ser comparado ao adotado no algoritmo DES. No entanto, no DES os "grupos G's" são formados por 64 subgrupos e estes subgrupos tem tamanho fixo igual a 1 elemento.

4.3.2 – Geração de Números Pseudo-aleatórios na implementação

Foram elaboradas 16 funções lineares congruentes, seguindo as condições propostas pelo Teorema A, abordado na seção 3.2.1. Essas funções recebem o nome de células de aleatoriedade, pois servem para geração de seqüências de números pseudo-aleatórios. Os valores dos parâmetros **a**, **c** e **m** para cada função são mostrados na Tabela 4.4.

Tabela 4.4- Funções lineares congruentes.

Numeração	a	c	m	Função gerada
1	1032	101	1062961	$x_{i+1}=(1032 * x_i + 101) \bmod 1062961$
2	1320	613	1739761	$x_{i+1}=(1320 * x_i + 613) \bmod 1739761$
3	1278	907	1630729	$x_{i+1}=(1278 * x_i + 907) \bmod 1630729$
4	2018	191	4068289	$x_{i+1}=(2018 * x_i + 191) \bmod 4068289$
5	2820	277	7946761	$x_{i+1}=(2820 * x_i + 277) \bmod 7946761$
6	1671	307	2785561	$x_{i+1}=(1671* x_i + 307) \bmod 2785561$
7	1154	479	1329409	$x_{i+1}=(1154 * x_i + 479) \bmod 1329409$
8	1440	811	2070721	$x_{i+1}=(1440 * x_i + 811) \bmod 2070721$
9	2244	440	5031049	$x_{i+1}=(2244 * x_i + 440) \bmod 5031049$
10	1874	167	3508129	$x_{i+1}=(1874 * x_i + 167) \bmod 3508129$
11	2610	433	6806881	$x_{i+1}=(2610 * x_i + 433) \bmod 6806881$
12	1778	673	3157729	$x_{i+1}=(1778 * x_i + 673) \bmod 3157729$
13	2142	739	4583881	$x_{i+1}=(2142 * x_i + 739) \bmod 4583881$
14	1232	523	1515361	$x_{i+1}=(1232 * x_i + 523) \bmod 1515361$
15	1584	769	2505889	$x_{i+1}=(1584 * x_i + 769) \bmod 2505889$
16	2390	293	5707321	$x_{i+1}=(2390* x_i + 293) \bmod 5707321$

Os números que servirão como sementes para as funções acima, são obtidos por meio da função rand() durante o processo de codificação. A função rand() é uma função de geração de números randômicos e faz parte do pacote de

funções do Matlab 7.0.1. As sementes geradas formarão a chave 1 usada no algoritmo para codificação e decodificação, juntamente com um número adicional que representa o tamanho **h** dos subgrupos. A formação dos subgrupos **h** é abordada na seção 4.2.1.

No processo de decodificação, a chave 1 deve ser digitada pelo usuário para que o sinal possa ser recuperado.

Portanto, a chave 1 será formada por 17 números, dos quais os 16 primeiros se encontram no intervalo entre 1 e 256, ou seja, 8 bits de representação para cada número totalizando 128 bits. O último número será representado por 12 bits, totalizando uma chave de tamanho igual a 140 bits.

O protótipo conta ainda com uma segunda chave que também é gerada durante o processo de codificação. Ela é formada por 16 números. Esses números estão no intervalo entre 1 e 16 (incluindo os extremos), ou seja, 4 bits para representar cada um, totalizando uma chave de 68 bits. Os 16 números dessa chave, serão usados para estabelecer quais células de aleatoriedade serão utilizadas. Como um exemplo da relação entre as chaves, será analisada a situação hipotética a seguir. Se os primeiros 16 números da chave 1 são 1, 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 , 16 ; e os 16 números da chave 2 são 2 , 4 , 6 , 9 , 3 , 7 , 5 , 1 , 10 , 8 , 12 , 6 , 13 , 15 , 16 , 14; significa que o número 1 (primeiro elemento da chave 1) vai ser usado como semente pela função 2 (primeiro elemento da chave 2); o número 2 (segundo elemento da chave 1) vai ser usado com semente pela função 4 (segundo elemento da chave 2) ; o número 3 (terceiro elemento da chave 1) vai ser usado como semente pela função 6 (terceiro elemento da chave 2) e assim por diante.

Outra observação importante relacionada as chaves é que uma célula de aleatoriedade pode ser acionada mais de uma vez, utilizando como semente o número correspondente na chave 1. Na situação hipotética mencionada acima, a função 6 se repete nas posições 3 e 12 da chave 2. Essa será utilizada duas vezes, com as sementes 3 e 12 que são respectivamente a posição 3 e a posição 12 na chave 1.

Essa técnica garante um nível de segurança razoável para as chaves pois, pelo método da força bruta, devem ser geradas $2^{140} * 2^{67} \cong 2 * 10^{62}$ combinações diferentes.

4.3.3 – Operações de XOR (OU-Exclusivo) na implementação

No algoritmo implementado, as operações de XOR são aplicadas entre o sinal resultante da transposição inicial e as seqüências numéricas pseudo-aleatórias geradas pelas funções lineares congruentes. São realizadas 16 operações de XOR, uma para cada seqüência gerada. O resultado de todas estas operações é o sinal codificado final.

4.4 – Telas do Algoritmo

A tela inicial implementada é ilustrada na Figura 4.4. Nessa tela, o usuário pode optar por abrir um arquivo Wave para efetuar a cifragem, ou abrir um arquivo Wave cifrado para que seja recuperado o sinal original. Os arquivos Wave abertos devem seguir as especificações apresentadas na seção 4.1, sob pena de não serem lidos satisfatoriamente, ou de gerarem anomalias durante os processos de cifragem e decifragem dos dados.

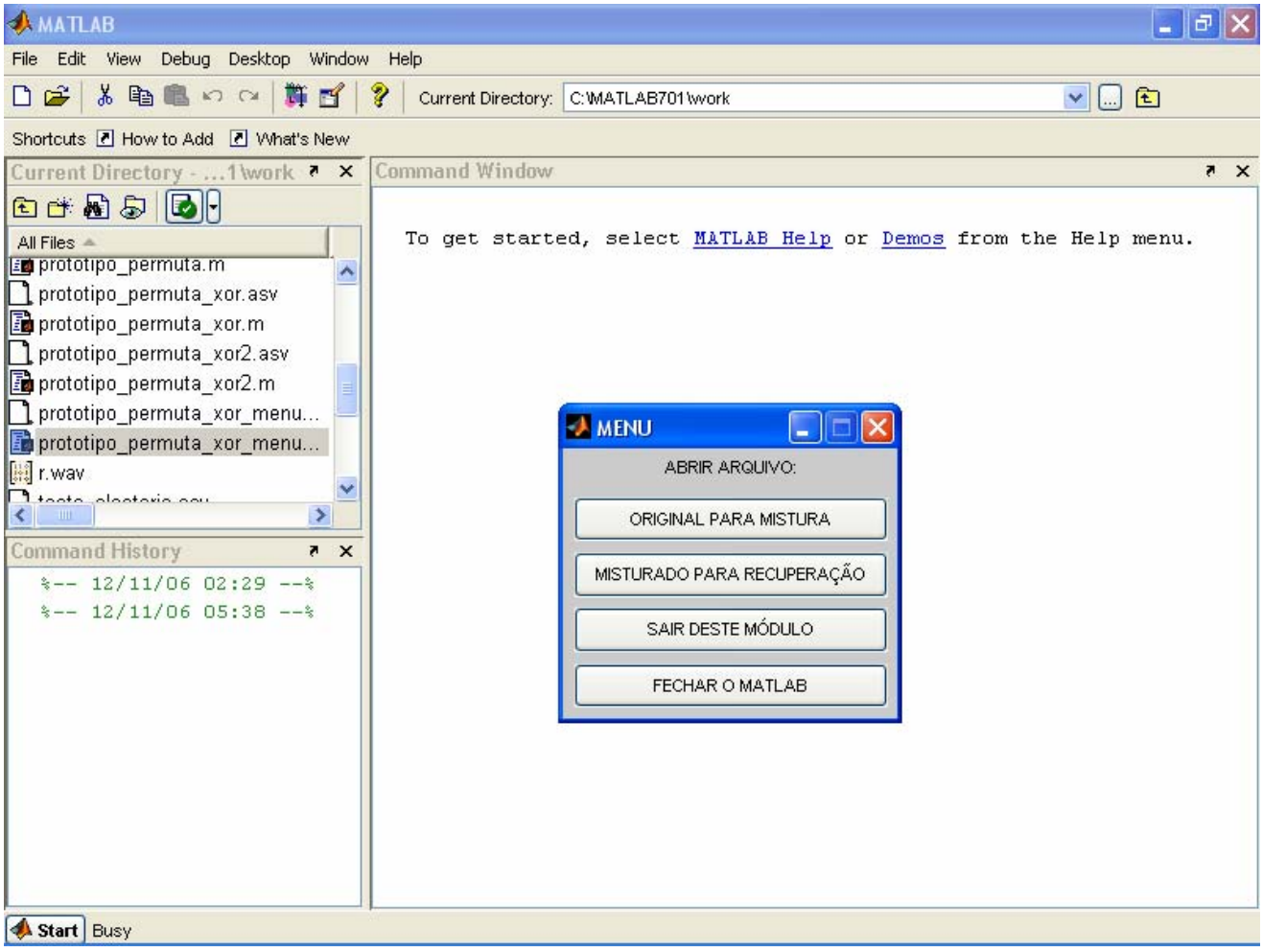


Figura 4.4 – Tela inicial do algoritmo.

Se optar por abrir um arquivo Wave original para que seja feita a codificação, ou se optar por abrir um arquivo Wave misturado para recuperação, uma tela de abertura de arquivo será mostrada, na qual poderá ser escolhido o arquivo Wave a ser lido.

Após a abertura do arquivo Wave original para mistura, uma tela como a mostrada na Figura 4.5 é aberta. Nela, o usuário tem a opção de ouvir o arquivo aberto ou efetuar a mistura no arquivo.

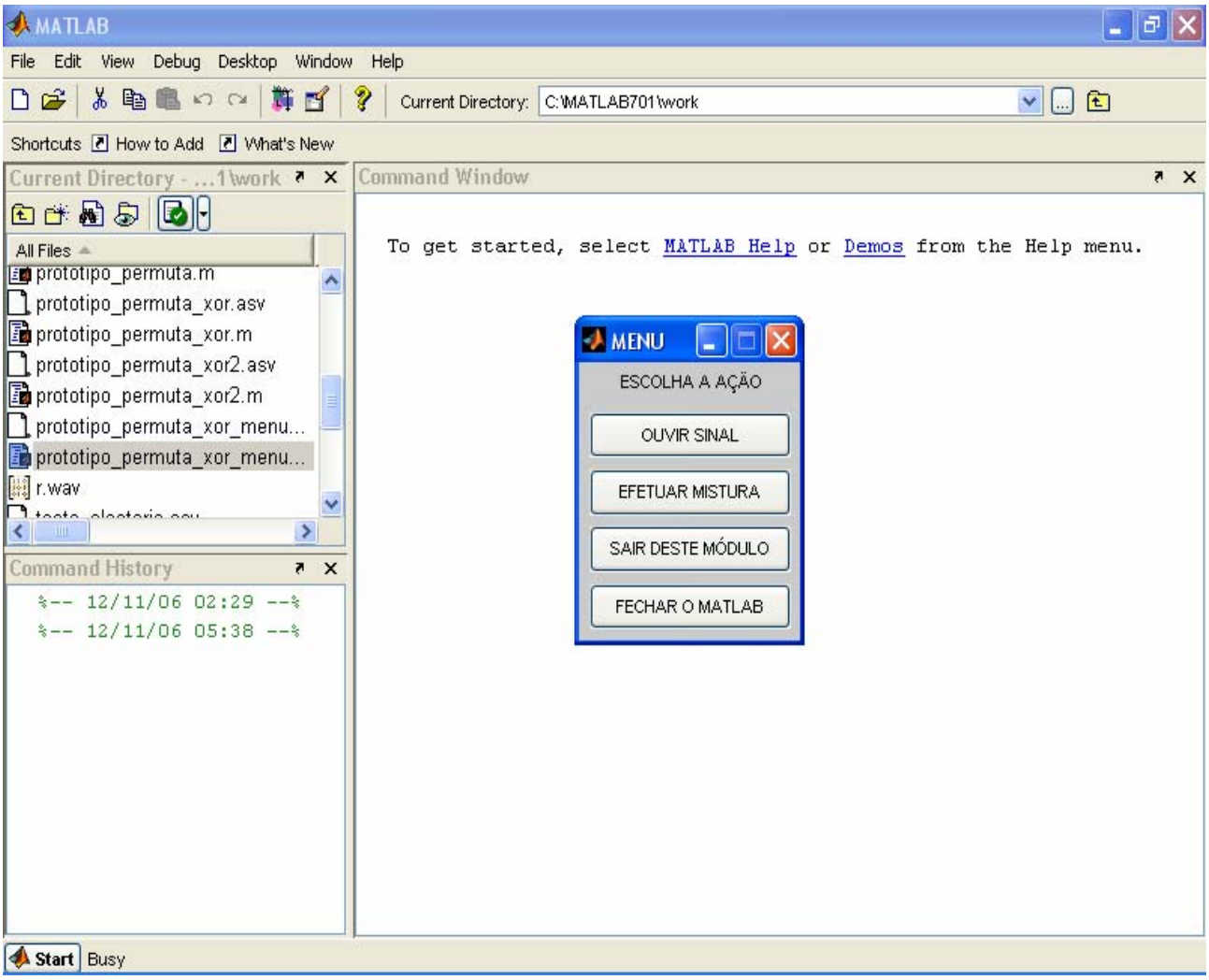


Figura 4.5 – Tela intermediária do algoritmo.

Ao clicar na opção “EFETUAR MISTURA” aparecerá, após a finalização do processo de mistura, a tela ilustrada na Figura 4.6, na qual o usuário poderá ver as chaves geradas, ouvir o resultado da mistura e salvar o sinal resultante em um arquivo Wave. Além disso, poderá ouvir o sinal permutado originado da operação

de transposição, bem como gerar gráficos tanto deste sinal quanto do sinal misturado, produto final do processo.

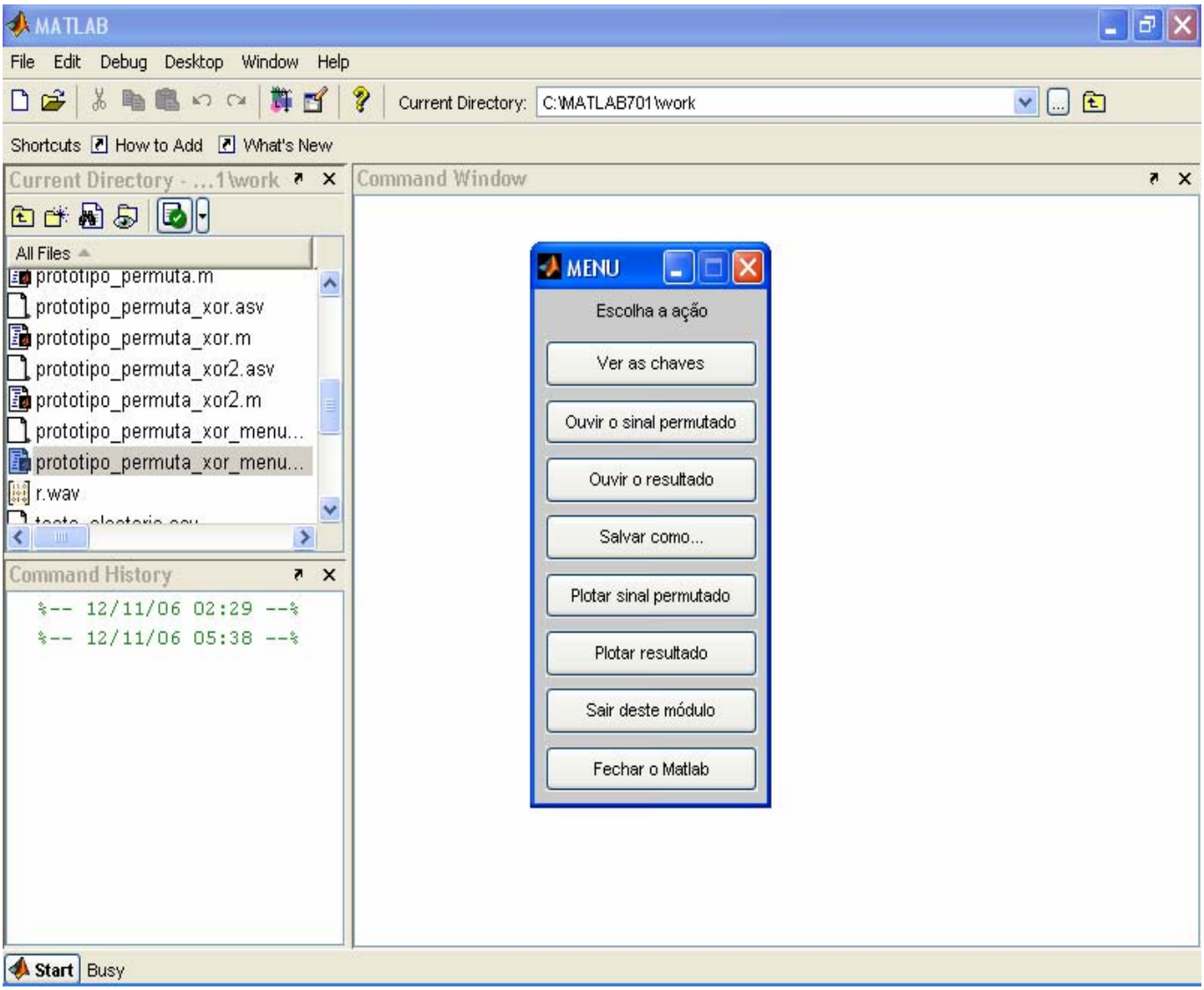


Figura 4.6 – Tela intermediária do algoritmo.

Se for aberto um arquivo Wave misturado para recuperação, surgirá a tela apresentada na Figura 4.7, na qual o usuário poderá escolher a opção de ouvir o sinal misturado aberto, recuperar o sinal misturado, sair do módulo e fechar o Matlab.

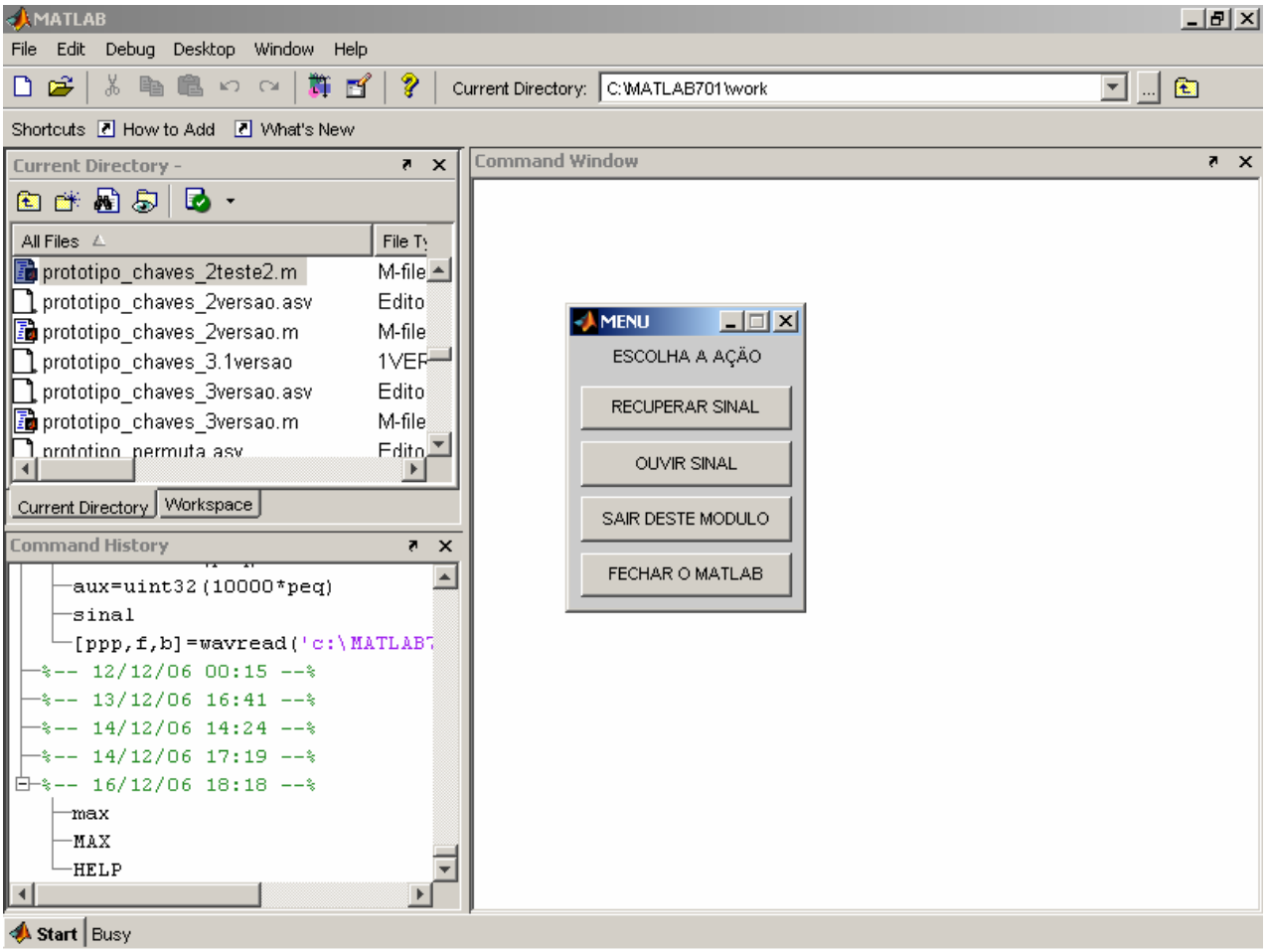


Figura 4.7 – Tela intermediária do algoritmo.

Se a opção “RECUPERAR SINAL” for escolhida, é solicitado ao usuário que digite as duas chaves geradas no processo de codificação. Após a digitação das chaves é iniciado o processo de recuperação e ao final desta etapa, a tela mostrada na Figura 4.8 será aberta.

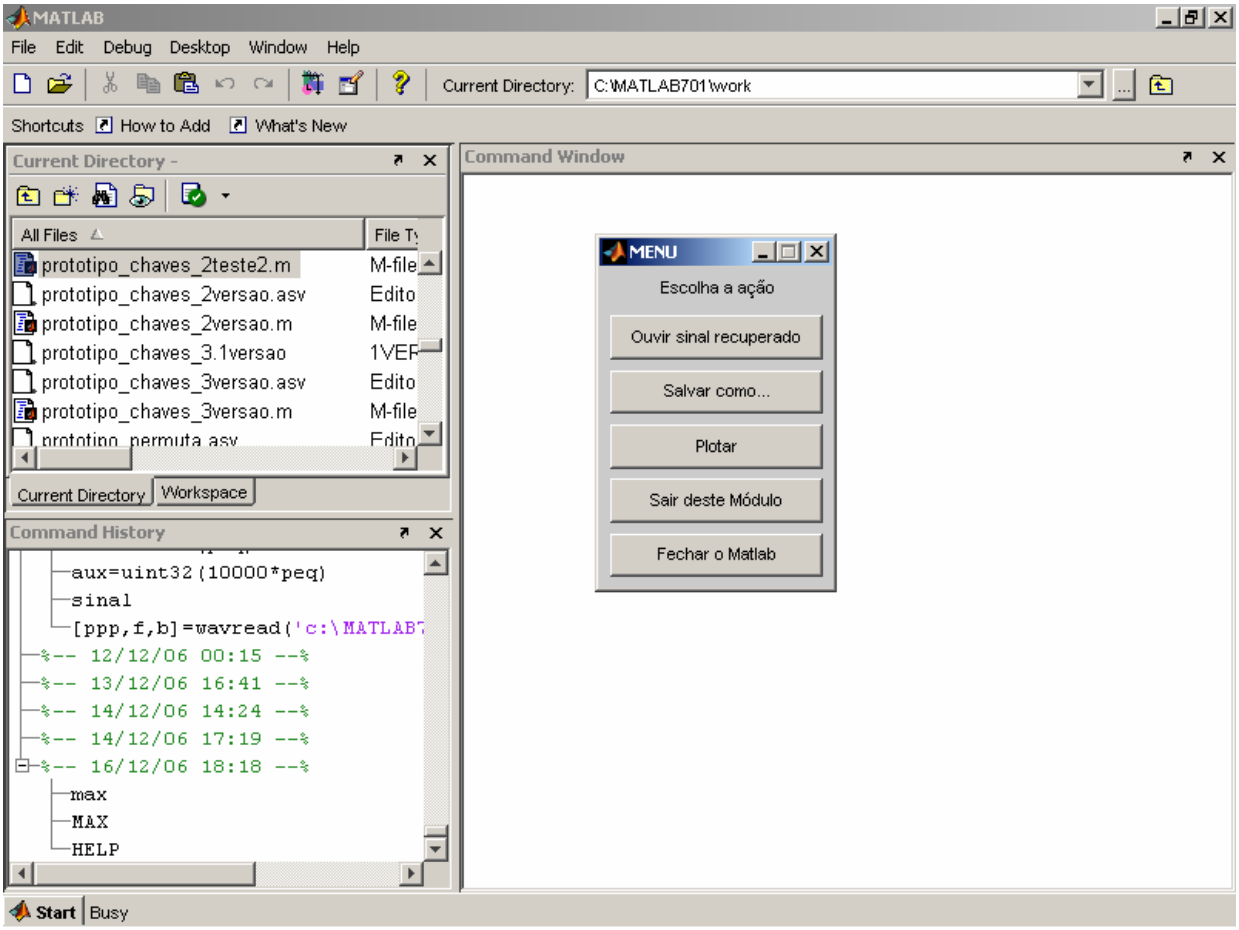


Figura 4.8 – Tela final do processo de recuperação.

Nesta tela, o usuário poderá optar por ouvir o sinal recuperado, salvar este sinal em um arquivo Wave ou plotar o seu gráfico.

4.5 – Resultados Obtidos

A seguir são mostrados os gráficos de alguns dos sinais testados nas simulações feitas.

A Figura 4.9 ilustra o gráfico Amplitude x Tempo Amostral de um sinal que representa a mensagem falada “teste de comunicação”, com tempo de 1,58 segundos. Este sinal foi gravado em um arquivo Wave com taxa de amostragem

de 22khz, codificação de 16 bits, utilizando 1 canal (mono) e no formato PCM. Ele será referenciado como “**sinal de teste 1**”.

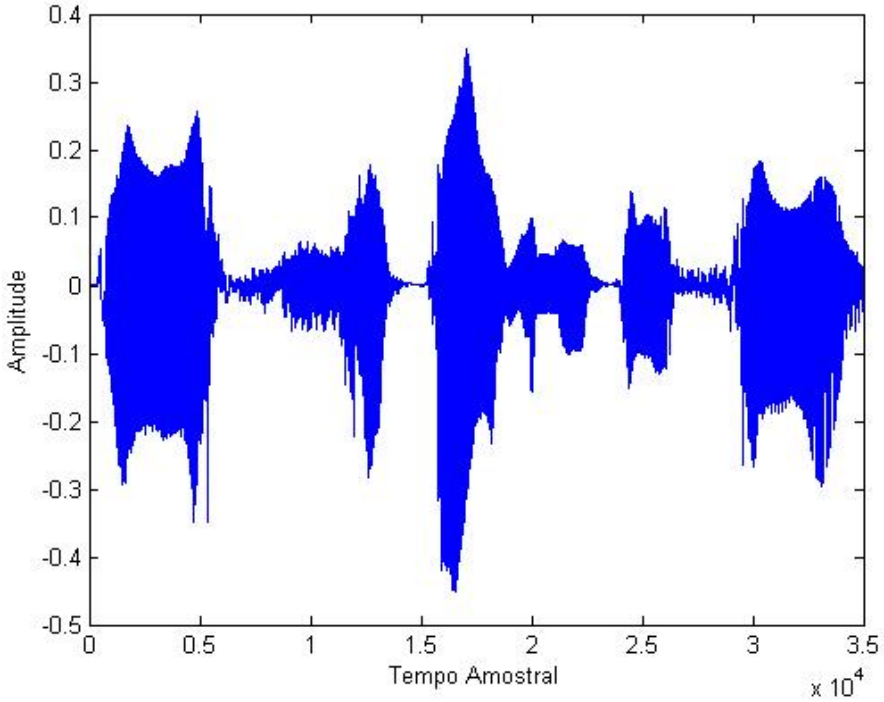


Figura 4.9 – Gráfico do **sinal de teste 1** original.

Após a aplicação do algoritmo de cifragem implementado sobre este sinal, o gráfico do **sinal de teste 1** transposto, resultante da operação de transposição inicial sobre o **sinal de teste 1**, pode ser visto na Figura 4.10.

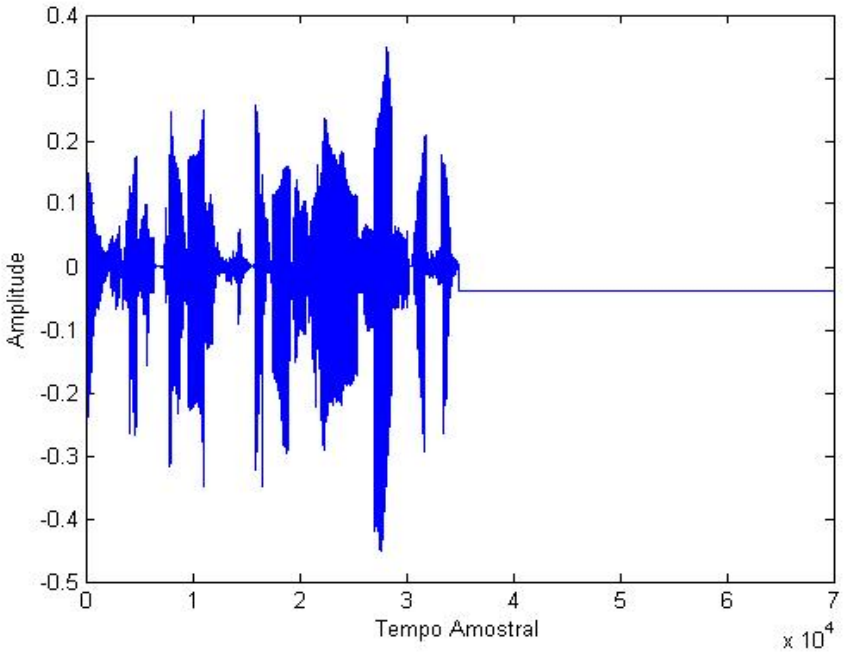


Figura 4.10 – Gráfico do **sinal de teste 1** transposto.

Por meio da observação comparativa entre a Figura 4.9 e a Figura 4.10, pode-se perceber que, na resultante da transposição do **sinal de teste 1**, o número de amostras é maior que o existente no **sinal de teste 1** original. Isso ocorreu devido a necessidade de acréscimo de amostras para o processo de formação dos grupos **G**. A formação dos grupos **G** é abordada na seção 4.3.1. Outra observação importante é que as amostras do **sinal de teste 1** original não têm seus valores alterados no processo de transposição, apenas sofrem mudanças em suas posições.

O gráfico referente ao **sinal de teste 1** resultante, obtido após as operações de XOR entre o **sinal de teste 1** e os sinais aleatórios, é mostrado na Figura 4.11.

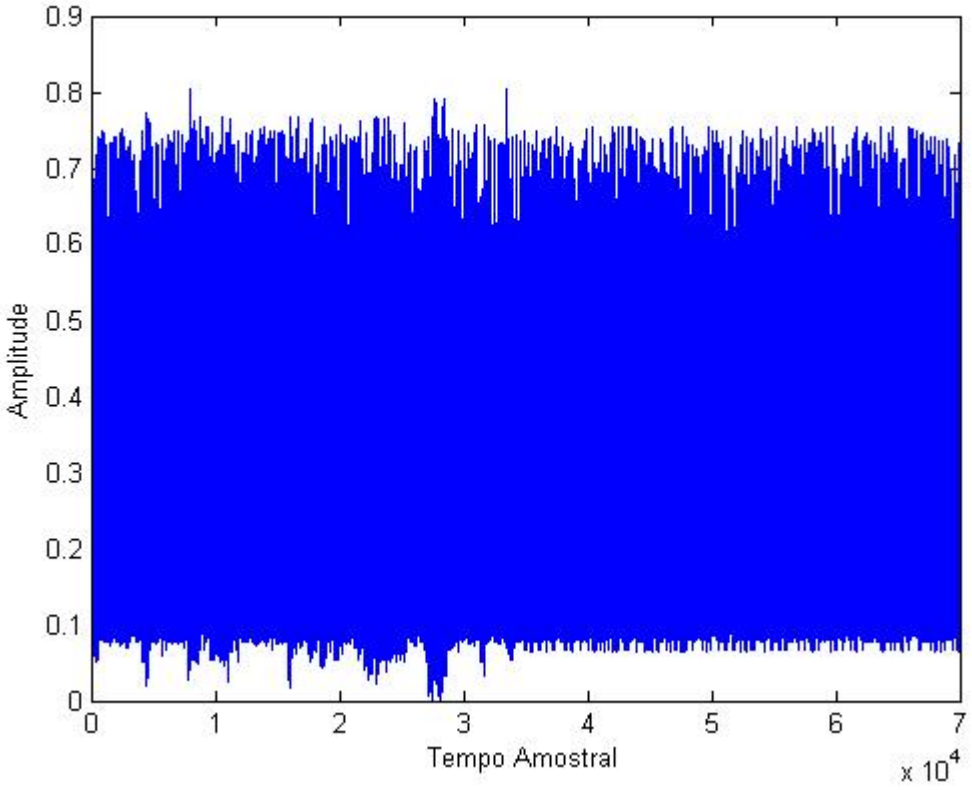


Figura 4.11 – Gráfico do **sinal de teste 1** resultante.

Uma observação importante referente a Figura 4.11 é que, nessa etapa do algoritmo, os valores das amostras são alterados, pois ocorre a inserção do ruído aleatório ao sinal original da mensagem, o que provoca a ininteligibilidade do mesmo.

O gráfico referente ao **sinal de teste 1** recuperado, resultante da aplicação da decifragem sobre o **sinal de teste 1** resultante, é apresentado na Figura 4.12

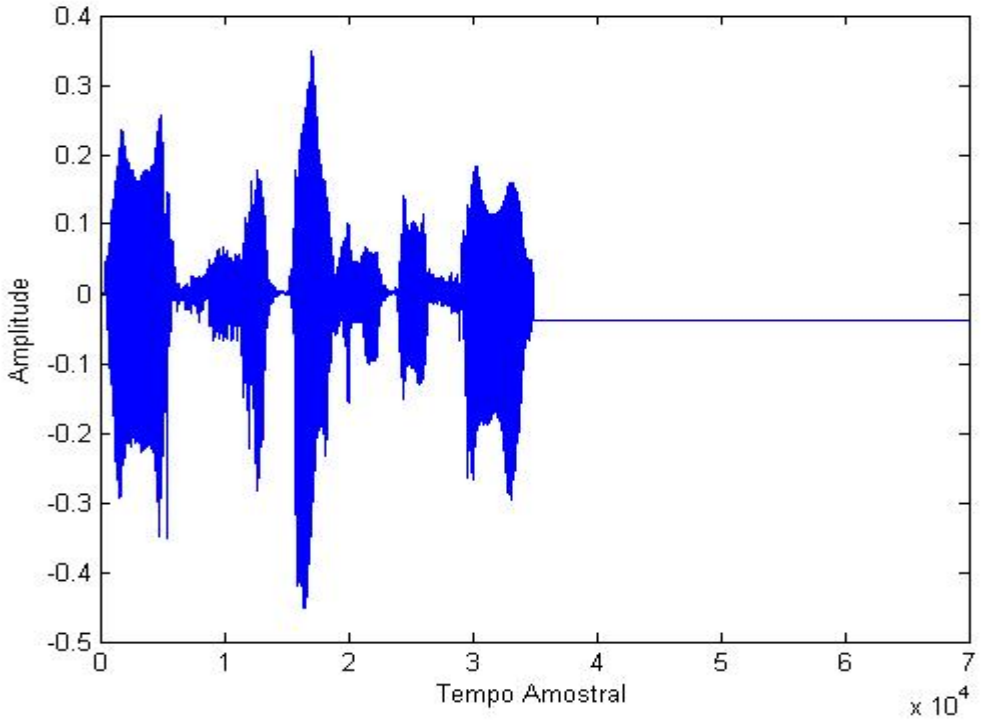


Figura 4.12 – Gráfico do **sinal de teste 1** recuperado.

Pode ser observado na Figura 4.12 que o **sinal de teste 1** recuperado possui características semelhantes ao **sinal de teste 1** original representado na Figura 4.9. A diferença entre eles está no número de amostras, pois no sinal recuperado existem mais amostras que no sinal original. Isto se deve ao fato de que, durante o processo de transposição, foram acrescidas amostras ao sinal original. Esse acréscimo foi necessário para a correta formação dos grupos **G**, sendo que estas amostras acrescidas permanecem no sinal recuperado, formando a parte residual do processo.

Na Figura 4.13 é mostrado o gráfico do sinal referente a mensagem de áudio “Esta é uma mensagem teste.”, com tempo de 2,25 segundos, sendo armazenada em um arquivo Wave com taxa de amostragem 48khz, 16 bits de

codificação, utilizando 1 canal (mono) e no formato PCM. Este sinal receberá o nome de **sinal de teste 2**.

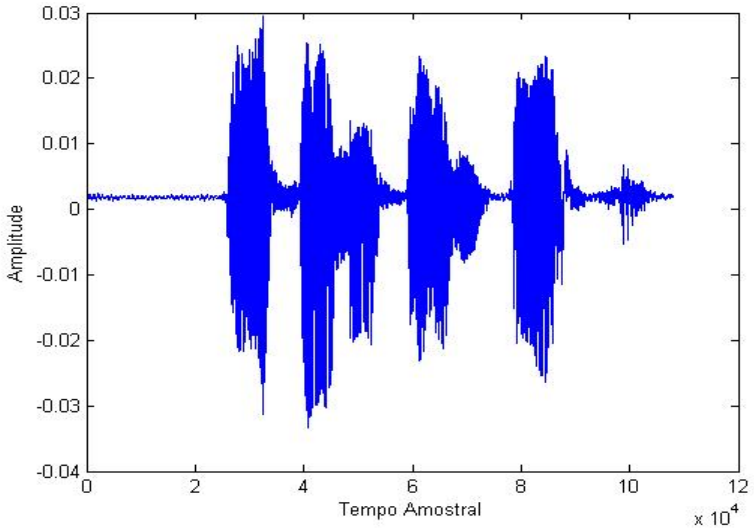


Figura 4.13 – Gráfico do **sinal de teste 2** original.

Depois que o sinal original passa pela etapa de transposição, o **sinal de teste 2** transposto resultante pode ser visto na Figura 4.14.

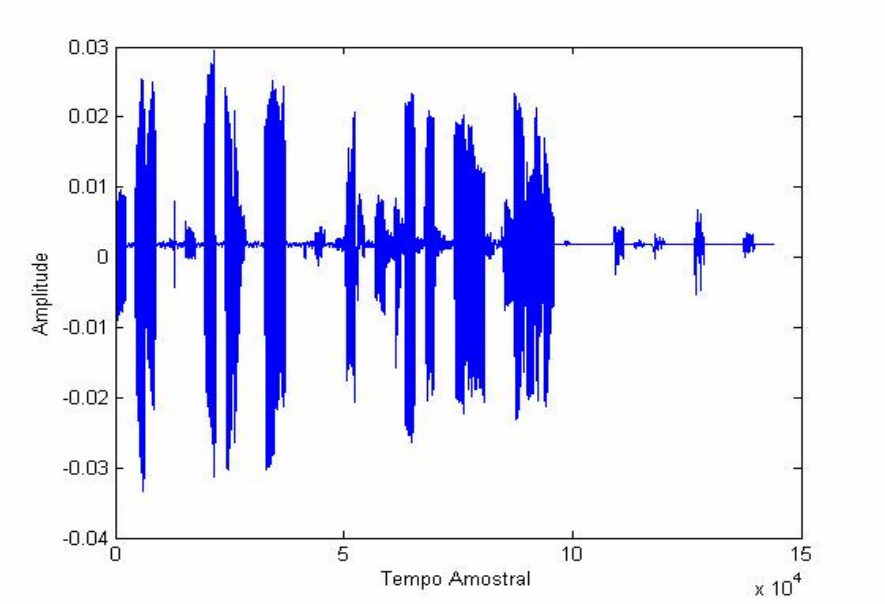


Figura 4.14 – Gráfico do **sinal de teste 2** transposto.

O sinal transposto mostrado na Figura 4.14 possui as mesmas amostras contidas no sinal original, porém elas estão ordenadas de forma diferente. Além das amostras do sinal original, foram acrescidas amostras extras para a formação dos grupos **G** (vide seção 4.3.1).

O **sinal de teste 2** resultante é obtido após serem efetuadas as operações de XOR entre o **sinal de teste 2** original e os sinais aleatórios gerados, e o seu gráfico pode ser visualizado na Figura 4.15. Nesse gráfico, as amostras do sinal tem seus valores modificados (resultado da operação de XOR).

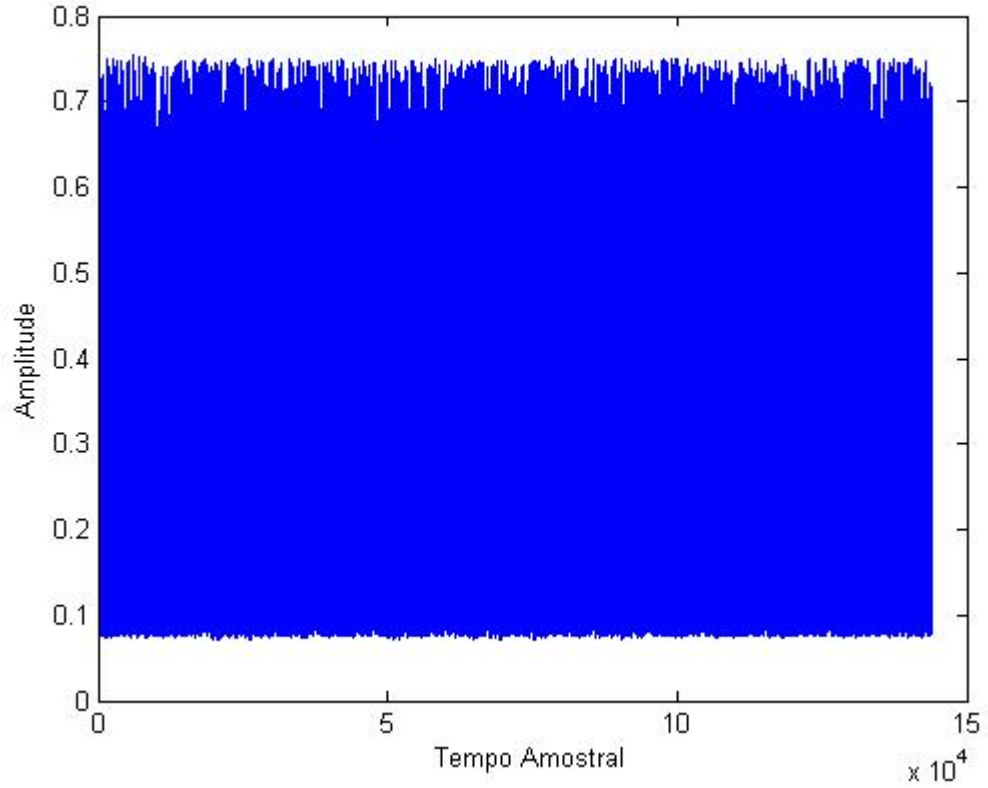


Figura 4.15 – Gráfico do **sinal de teste 2** resultante.

O gráfico do **sinal de teste 2** recuperado a partir do sinal resultante é visto na Figura 4.16.

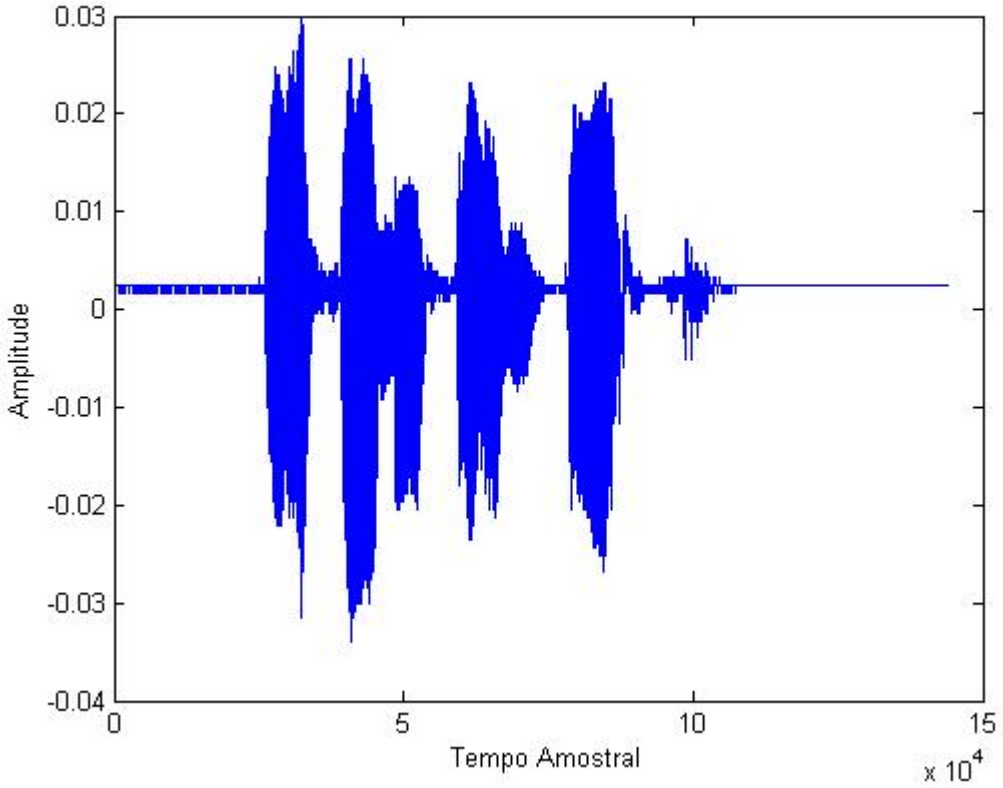


Figura 4.16 – Gráfico do **sinal de teste 2** recuperado.

Na observação da Figura 4.16, pode ser visto que as amostras acrescidas durante o processo de permutação permanecem nesse sinal, sendo consideradas os resíduos do algoritmo, assim como aconteceu com o **sinal de teste 1** recuperado, observado anteriormente.

5- CONCLUSÃO

No desenvolvimento deste projeto, conhecimentos foram ganhos sobre o processo de scrambler, bem como sobre os processos de manipulação de sinais de áudio e geração de números pseudo-aleatórios entre outros.

Um dos principais aprendizados diz respeito a manipulação dos sinais sonoros, que se mostrou não muito trivial, o que exigiu vários esforços e muita pesquisa durante o processo de implementação do algoritmo, pois a técnica pensada inicialmente - realizar a cifragem do sinal por meio do somatório algébrico simples entre o sinal original e os sinais aleatórios - se mostrou pouco eficiente na prática.

Outra importante conclusão, diz respeito ao Matlab, que se mostrou bastante eficiente e fácil de ser usado. Porém, algumas poucas dificuldades foram encontradas com relação à forma com que a manipulação dos dados ocorre, o que provocou problemas inexplicáveis na fase de implementação, como o travamento da máquina e o aparecimento de mensagens de erro ilógicas. Tais problemas foram solucionados mais tarde com a pesquisa mais detalhada sobre esta ferramenta.

A técnica de scrambler proporcionou bons resultados com relação ao objetivo estabelecido no início do projeto, de transformar um sinal de áudio claro em um sinal de áudio ininteligível. No entanto, quanto maior o grau de segurança aplicado, maior será o processamento envolvido durante a transformação do sinal claro em sinal misturado[Buchmann,2002].

Quanto a geração de números pseudo-aleatórios, o método linear congruente, apesar de não ser o método mais robusto para geração deste tipo de número [Buchmann,2002]/[Rosa,Junior,2002], mostrou-se eficiente com a criação das funções lineares congruentes sendo regidas pelas relações propostas no Teorema A apresentado na seção 3.2.1.

Com relação a implementação, os resultados esperados foram alcançados. Tal afirmação pode ser feita pois, ao final do processo de cifragem, não houveram resquícios de inteligibilidade no sinal codificado resultante. Além disso, as

mensagens de áudio recuperadas por meio do processo de decifragem, puderam ser entendidas claramente.

Apesar da boa pesquisa proporcionada pelo planejamento e execução deste projeto, o mesmo pode ser aperfeiçoado no intuito de se tornar uma ferramenta viável comercialmente. Para tanto, algumas sugestões de continuidade dos estudos aqui apresentados são citadas com o objetivo de servirem como base para projetos futuros. São elas:

- Implementação do algoritmo apresentado utilizando linguagens comerciais, tais como C++, Java, Pascal, Delphi, Visual Basic;
- Implementação de um protótipo desenvolvido em uma estrutura de hardware;
- Utilização de outros métodos para geração de números pseudo-aleatórios, como por exemplo o Blum Blum Shub, e de outras técnicas para manipulação dos sinais sonoros;
- Desenvolvimento de um estudo criptoanalítico sobre o algoritmo apresentado, apontando possíveis pontos vulneráveis e alternativas de melhorias;
- Implementação de um projeto para aplicação do algoritmo apresentado em comunicações VoIP.

BIBLIOGRAFIA

[Hanselman,2003] – **MATLAB 6 – Curso Completo** - Duane Hanselman, Bruce Littlefield; Pearson Education , São Paulo, 2003

[Haykin,Veen,2001] – **Sinais e Sistemas** - Simon Haykin, Barry Van Veen; Bookman , Porto Alegre, 2001

[Tipler,2000] – **Física Para Cientistas e Engenheiros – Volume 1**, Paul A. Tipler; Editora LTC, 4ª edição, 2000.

[Buchmann,2002] – **Introdução à criptografia***,* Johannes A. Buchmann; Editora Berkeley, São Paulo, 2002.

[Matsumoto,2001] – **MATLAB 6 – Fundamentos de Programação**, Élia Yathie Matsumoto; Editora Érica, São Paulo, 2001.

[Rosa,Junior,2002] – **Gerando números aleatórios**, Fernando Henrique F. P. da Rosa, Vagner Aparecido P. Junior. Acessível em: http://www.feferraz.net/files/lista/random_numbers.pdf. Último acesso: 20 de novembro de 2006.

[Soares,Lemos,Colcher,1995] – **Redes de computadores**, Luiz Fernando Gomes Lemos, Guido Lemos, Sérgio Colcher; Editora Campus, Rio de Janeiro, 1995.

[WEB1] – **Som**, Wikipédia. Acessível em: http://pt.wikipedia.org/wiki/%C3%81udio – Último acesso em: 20 de novembro de 2006.

[WEB2] – **Remoção de Ruídos** – Cristiano Scabello. Acessível em: http://estudiolivre.org/tiki-index.php?page=ruidos&bl – Último acesso em: 20 de novembro de 2006.

[WEB3] – **Acústica**, Alunos da Escola Estadual Prof.º Ascendino Reis. Acessível em: http://ww2.unime.it/weblab/awardarchivio/ondulatoria/acustica.htm – Último acesso em: 20 de novembro de 2006

[WEB4] – **Física con ordenador, Curso Interactivo de Física en Internet**, Angel Franco Garcia. Acessível em: http://www.sc.ehu.es/sbweb/fisica/default.htm – Último acesso em: 20 de novembro de 2006

[WEB5] – **O ouvido humano**, C.A. Bertulani. Acessível em: http://www.if.ufrj.br/teaching/fis2/ondas2/ouvido/ouvido.html – Último acesso em: 20 de novembro de 2006.

[WEB6] – **Ondas Sonoras**, C.A. Bertulani. Acesssível em: http://www.if.ufrj.br/teaching/fis2/ondas2/ondas2.html – Último acesso em: 20 de novembro de 2006.

[WEB7] –**Tutoriais de Áudio e Acústica**, Fernando Iazzetta. Acessível em: http://www.eca.usp.br/prof/iazzetta/tutor/index.html – Último acesso em: 20 de novembro de 2006.

[WEB8] – **Digitalização de um sinal analógico**, Roland. Acessível em: http://paginas.terra.com.br/lazer/py4zbz/teoria/digitaliz.htm – Último acesso em: 20 de novembro de 2006.

[WEB9] – **Som e placas de som**, Taisy Silva Weber. Acessível em: http://www.inf.ufrgs.br/~taisy/disciplinas/slides/Arq11som.pdf – Último acesso em: 20 de novembro de 2006.

[WEB10] – **Áudio Digital**, Fernando Iazzetta. Acessível em: http://www.eca.usp.br/prof/iazzetta/tutor/audio/a_digital/a_digital.html – Último acesso em: 20 de novembro de 2006.

[WEB11] – **Criptografia NumaBoa**, Vicktoria Tkotz. Acessível em: http://www.numaboa.com/content/section/11/57 – Último acesso em: 20 de novembro de 2006.

[WEB12] – **AES Proposal: Rijndael**, Joan Daeman, Vincent Rijmen. Acessível em: http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf – Último acesso em: 20 de novembro de 2006.

[WEB13] – **Criptografia simétrica**, Pedro Ferreira do Souto. Acessível em: http://paginas.fe.up.pt/~pfs/aulas/ssr2006/at/symcrypto2.pdf – Último acesso em: 20 de novembro de 2006.

[WEB14] – **Criptografia – BlowFish**, Rafael T. de Sousa Jr., Ricardo S. Puttini. Acessível em: http://www.redes.unb.br/security/criptografia/blowfish/default.html – Último acesso em: 20 de novembro de 2006.

[WEB15] – **Análise de "Stream Ciphers"**, Daniel Massaki Yamamoto, Tania Nunes Rabello. Acessível em: http://www.bibl.ita.br/ixencita/artigos/FundDanielMassaki.pdf – Último acesso em: 20 de novembro de 2006.

[WEB16] – **Criptoanálise de Sinais de Voz Cifrados por Permutação de Segmentos Temporias**, José Antônio Apolinário Júnior. Acessível em**:** http://aquarius.ime.eb.br/~apolin/papers/MScUnB.pdf – Último acesso em: 20 de novembro de 2006.

[WEB17] – **Tutorial para Matlab – Curso de Métodos Matemáticos em Finanças I**, Jorge P. Zubelli, Moacyr Silva, Dayse Haime Pastore. Acessível em:

http://w3.impa.br/~zubelli/tutorial/index.html – Últimо acesso em: 20 de novembro de 2006.

[WEB18] – **RC4,** Wikipédia. Acessível em: http://pt.wikipedia.org/wiki/RC4– Último acesso em: 24 de novembro de 2006.

[WEB19] – **Criptografía em audio**, Laboratório de criptografia. Acessível em http://castlab.cl/spn/curiosidades/audio/criptografia.html

APÊNDICE A – CÓDIGO DO ALGORITMO PROPOSTO

```
%inicialização de variáveis
j=0;
k=0;
pasta=' ';
arqwave=' ';

%menu principal "Abrir arquivo"
while k==0
        k = menu('ABRIR ARQUIVO:','ORIGINAL PARA MISTURA','MISTURADO PARA
RECUPERAÇÃO','SAIR DESTE MÓDULO','FECHAR O MATLAB');


        if k==1 %menu principal - opção: ORIGINAL PARA MISTURA
            [arqwave,pasta]=uigetfile('*.wav','Abrir sinal original');
            if isstr(arqwave)== 1
                str=strcat(pasta,arqwave);
                [sinal,f,b]=wavread(str);
            else

                k=0;

            end

             choose=k;

               %Submenu "ESCOLHA A AÇÃO"
             while choose==1
                choose = menu('ESCOLHA A AÇÃO','OUVIR SINAL','EFETUAR
MISTURA','PLOTAR SINAL','SAIR DESTE MÓDULO','FECHAR O MATLAB');



                if choose==2%Submenu - opção: EFETUAR MISTURA

                    %módulo de permutacao
                    p=[22 6  19 13 10 12 3 17 5 15 4 21 16 8  2 20 14 11
7 1 18 9];
                    iteracoes=22;
                    d=1;
                    dz=0;
                    i=1;
                    u=1;
                    q=0;
                    yp=sinal;
                    x=size(sinal);
                    x=x(1);
                    pcomp=0;
                    base=uint32(x/iteracoes);

                    %calculo dos subgrupos
                    while base>3000
                        base=uint32(base/1.5);
```

```
end
r=mod(uint32(x),iteracoes*base);
if r > 0
  pcomp=iteracoes*base-r;
    h=1;
   l=1;
   h=waitbar(0,'Adicionando amostras ao sinal');
   aux=double([1:1:pcomp]);
   while(l<=pcomp)
        aux(l)=sinal(x);
        waitbar(l/pcomp);
        l=l+1;
   end
    close(h);
    sinal=[sinal;aux'];
    yp=sinal;
end
x=x+pcomp;



%início da transposição
h=waitbar(0,'Efetuando a permutação...');
while i <= x
    waitbar(i/x);
    t=(p(d)-1)*base + 1 + dz*iteracoes*base;
    q=p(d)*base+ dz*iteracoes*base;


    if q>x
        t=q+1;
        i=x;
    end
    while t <= q
        yp(u)=sinal(t);
        u=u+1;
        t=t+1;
    end

    if d==iteracoes
        d=0;
        dz=dz + 1;
    end
    i=i+base;
    d=d+1;
end
close(h);
permutado=yp;
% fim do módulo de transposição

%vetor auxiliar para armazenamento do sinal das
%amostras.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
posit_negat=double(yp./abs(yp));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
                    %fim da montagem
                     %Tornando todo o sinal positivo
                    yp=abs(yp);


                    %modulo do xor
                      %montagem do sinal aleatorio
                    %q - vetor de números primos para referência na
montagem
                    %dos parâmetros 'm'

q=[1031;1319;1277;2017;2819;1669;1153;1439;2243;1873;2609;1777;2141;1231;
1583;2389];

c=[101;613;907;191;277;307;479;811;440;167;433;673;739;523;769;293];
                    %para n=1  em a=n*q+1.

a=[1032;1320;1278;2018;2820;1671;1154;1440;2244;1874;2610;1778;2142;1232;
1584;2390];
                    %m=q^2

m=[1062961;1739761;1630729;4068289;7946761;2785561;1329409;2070721;503104
9;3508129;
                       6806881;3157729;4583881;1515361;2505889;5707321];

                    x=size(yp);%tamanho do arquivo
                    x=x(1);
                    p2=yp;
                    i=1;
                    j=1;

                    %geracao das chaves
                    %geracao da key1
                    i=1;
                    key1=[1:1:17];
                    key2=[1:1:16];
                    key1=key1';
                    key2=key2';
                    h=waitbar(0,'Geração das chaves...');
                        while i<=16
                            waitbar(i/32);
                            aux=1000*rand(1,1);
                            key1(i)=uint32( mod(aux,256));
                            if key1(i)==0
                                i=i-1;
                            else
                                if key1(i)==256
                                    i=i-1;
                                end
                            end
                            i=i+1;
                        end
                    key1(17)= base;
                    %geracao da key2

                        i=1;
```

```
    while i<=16
        waitbar((i+16)/32);
        aux=1000*rand(1,1);
        key2(i)=uint32( mod(aux, 17));
        if key2(i)==0
            i=i-1;

        else

            if key2(i)==17
                i=i-1;
            end
        end
        i=i+1;
    end
    close(h);
  %utilizacao da key1 como semente para as funções
  %aleatórias apontadas pela key2
j=1;
w1=0;
w2=0;
valor=0;
anterior=0;
h=waitbar(0,'Gerando as seqüências aleatórias...');
while(j<=16)
    waitbar(j/16);
    valor=valor-anterior;
    i=1;
    anterior=valor;
    valor=valor + key1(j);
    w1=valor;
    w2=key2(j);
    while(i<=x)%x é o numero total de amostras
        d=a(w2,1)* w1 + c(w2,1);
        p2(i,j)= mod(d , m(w2,1));
        w1=p2(i,j);
        i=i+1;
    end
    j=j+1;
end
close(h);

   %final da montagem

p2=uint32(p2');
tam=[1:1:x];
r=uint32(p2(1,tam));
j=2;
h=waitbar(0,'Efetuando as operações de XOR');

%xor entre as seqüências aleatórias
while(j<=16)
    waitbar(j/16);
    r=uint32(bitxor(r,p2(j,tam)));
    j=j+1;
end
```

```
close(h);

i=1;
r=double(r);
    while(i<=x)
        if(r(1,i)>1)
            r(1,i)=double(r(1,i)/10);
        else
            r(1,i);
            i=i+1;
        end
    end

%AJUSTANDO VARIAVEIS
mult=10000;
r=r*3/4;
r2=r;
yp=yp/4;
yp=uint32(mult*yp);
i=1;
%Ajustando amostras positivas e negativas

 h=waitbar(0,'Ajustando as amostra positivas e
negativas...');
while(i<=x)%x é o número total de amostras
    if posit_negat(i)>=0 %Valor da amostra positivo?
        if mod(yp(i),2)==0 %Amostra divisível por 2
?;

        else
            yp(i)=yp(i)+1;%caso não seja,tornar-se-
a.;

        end
    else% Se o valor da amostra não for positivo...
        if mod(yp(i),2)~=0%ele é divisível por 2 ?
        else
            yp(i)=yp(i)+1;%se for, deixará de ser.
        end
    end
    i=i+1;
    waitbar(i/x);
end
close(h);



%Operação final de xor entre o sinal original e o
%sinal aleatório gerado.

r2=uint32(mult*r);
r2=r2';
r2=uint32(bitxor(r2,yp));
r2=double(r2);
r=double(r2);
r=double(r/mult);
```

```
                    %módulo de submenu: 'Escolha a ação'
                    escolha=1;
                    while escolha == 1
                        escolha = menu('Escolha a ação','Ver as
chaves','Ouvir o sinal permutado','Ouvir o resultado','Salvar
como...','Plotar sinal permutado','Plotar resultado','Sair deste
módulo','Fechar o Matlab');
                        if escolha == 1 %ver as chaves

                            i=1;
                           chave1='';
                           chave2='';
                           %preparando a chave 1
                            while i<=17
                                chave1=strcat(chave1,num2str(key1(i)));
                                chave1=strcat(chave1,'-');
                                i=i+1;
                            end

                            %preparando a chave 2
                            i=1;
                            while i<=16
                                chave2=strcat(chave2,num2str(key2(i)));
                                chave2=strcat(chave2,'-');
                                i=i+1;
                            end
                            %imprimindo na tela
                            inputdlg({'CHAVE 1:' 'CHAVE
2:'},'CHAVES',2,{chave1 chave2});

                        else
                            if escolha ==2 %submenu - opção: Ouvir o
sinal permutado
                                sound(permutado,f,b);
                                escolha=1;
                            else
                                if escolha == 3%submenu - opção: Ouvir o
resultado
                                    sound(r,f,b);
                                    escolha=1;
                                else
                                    if escolha == 4 %submenu - opção:
Salvar como

[arqwave,pasta]=uiputfile('*.wav','Salvar Mistura');
                                        arq=strcat(pasta,arqwave);
                                        wavwrite(r,f,b,arq);
                                        escolha=1;
                                    else
                                        if escolha == 5 %submenu - opção:
Plotar sinal permutado
                                            plot(permutado);
                                            escolha=1;
                                        else
                                            if escolha == 6 %submenu -
opção: Plotar resultado
```

```
                                                plot(r);
                                                escolha=1;
                                            else
                                                if escolha == 7 %submenu
- opção:Sair deste módulo
                                                    close;
                                                else
                                                    if escolha==8
%submenu - opção: Fechar o matlab
                                                        exit;
                                                    end
                                                end
                                            end
                                        end
                                    end
                                end
                            end

                        else
                            if choose==3 %submenu - opção: PLOTAR SINAL
                                plot(sinal);
                                choose=1;
                            else

                                if choose==4 %submenu - opção: Sair deste módulo
                                    close;
                                else
                                    if choose==5 %submenu - opção: Fechar o
MATLAB
                                        exit;
                                    else
                                        if choose==1 %submenu - opção: OUVIR
SINAL
                                            sound(sinal,f,b);
                                        end
                                    end
                                end
                            end
                        end

                    end




                else


                    if k==2 %MISTURADO PARA RECUPERAÇÃO
                    [arqwave,pasta]=uigetfile('*.wav','Abrir sinal misturado');
                        if  isstr(arqwave)== 1
                            str=strcat(pasta,arqwave);
```

```
                    [sinal,f,b]=wavread(str);
                else
                    k=0;
                end

            choose=k;
            %SUBMENU ESCOLHA A AÇÃO
            while choose==2
                choose = menu('ESCOLHA A AÇÃO','RECUPERAR SINAL','OUVIR
SINAL','SAIR DESTE MODULO','FECHAR O MATLAB');

                 if choose==1 %submenu - opção: RECUPERAR SINAL


                    %montagem do sinal aleatório

q=[1031;1319;1277;2017;2819;1669;1153;1439;2243;1873;2609;1777;2141;1231;
1583;2389];

c=[101;613;907;191;277;307;479;811;440;167;433;673;739;523;769;293];
                    %para n=1 igual a 1 em a=n*q+1.

a=[1032;1320;1278;2018;2820;1671;1154;1440;2244;1874;2610;1778;2142;1232;
1584;2390];
                    %m=q^2

m=[1062961;1739761;1630729;4068289;7946761;2785561;1329409;2070721;503104
9;3508129;
                       6806881;3157729;4583881;1515361;2505889;5707321];

                    x=size(sinal);%tamanho do arquivo
                    x=x(1);
                    p2=sinal;
                    i=1;
                    j=1;
                    msg='Digite o numero -';
                    msg2=('º- da chave 1');

                    %obtendo as chaves
                    while(i<=17)
                        aux=num2str(i);
                        msgr=strcat(msg,aux);
                        msgr=strcat(msgr,msg2);
                        aux2=inputdlg(msgr,'OBTER CHAVES',1);
                        aux2=cell2mat(aux2);
                        key1(i)=str2num(aux2);
                        i=i+1;
                    end

                    msgr=msg;
                    msg2=('º- da chave 2');
                    i=1;

                    while(i<=16)
                        aux=num2str(i);
```

```
                    msgr=strcat(msg,aux);
                    msgr=strcat(msgr,msg2);
                    aux2=inputdlg(msgr,'OBTER CHAVES',1);
                    aux2=cell2mat(aux2);
                    key2(i)=str2num(aux2);
                    i=i+1;
                end

                j=1;
                w1=0;
                valor=0;
                anterior=0;
                h=waitbar(0,'Processando o sinal aleatório a partir
das chaves...');

                while(j<=16)
                    waitbar(j/16);
                    valor=valor-anterior;
                    i=1;
                    anterior=valor;
                    valor=valor + key1(j);
                    w1=valor;
                    w2=key2(j);
                    while(i<=x)%x é o numero total de amostras
                        d=a(w2,1)* w1 + c(w2,1);
                        p2(i,j)= mod(d , m(w2,1));
                        w1=p2(i,j);
                        i=i+1;
                    end
                    j=j+1;
                end
                close(h);
                   %final da montagem

                i=1;
                j=1;
                p2=uint32(p2');
                tam=[1:1:x];
                r=uint32(p2(1,tam));
                j=2;
                h=waitbar(0,'Efetuando as operações de XOR');
                %xor entre as seqüências aleatórias
                while(j<=16)
                    waitbar(j/16);
                    r=uint32(bitxor(r,p2(j,tam)));
                    j=j+1;
                end
                close(h);

                %while(j<=16)
                r=double(r);
                i=1;
                    while(i<=x)
                        if(r(1,i)>1)
                            r(1,i)=r(1,i)/10;
                        else
```

```
                i=i+1;
            end
        end

    %Ajustando o valor do sinal para um número inteiro
    mult=10000;
    sinal2=uint32(mult*sinal);

    %operação de XOR final entre o sinal original e o
    %aleatório

    r=r*3/4;

    r2=uint32(mult*r);
    r2=r2';
    sinal=double(bitxor(sinal2,r2));

     %Avaliando o sinal de cada amostra...

     i=1;
    posit_negat=[1:1:x];
    posit_negat=posit_negat';%vetor linha
    h=waitbar(0,'Montando o vetor auxiliar de sinais das
amostras...');
    while(i<=x)
        if mod(sinal(i),2)==0
            posit_negat(i)=1;
        else
            posit_negat(i)=-1;
        end
        waitbar(i/x);
        i=i+1;

    end
    close(h);
    % fim da montagem
    sinal=4*sinal;
    sinal=double(sinal/mult);
    sinal=sinal .* posit_negat;

    %desmisturar
    p=[20 15  7 11 9 2 19 14 22 5 18 6 4 17 10 13 8 21 3
16 12 1];
    yp=sinal;
    x=size(sinal);
    x=x(1);
    iteracoes=22;
    base=key1(17);
    d=1;
    dz=0;
    i=1;
    u=1;
    q=0;
    h=waitbar(0,'Permutação inversa do sinal...');
    while i <= x
```

```
                    t=(p(d)-1)*base + 1 + dz*iteracoes*base;
                    q=p(d)*base+ dz*iteracoes*base;


                    if q>x
                        t=q+1;
                        i=x;
                    end
                    while t <= q
                        yp(u)=sinal(t);
                        u=u+1;
                        t=t+1;
                    end

                    if d==iteracoes
                        d=0;
                        dz=dz + 1;
                    end
                    waitbar(i/x);
                    i=i+base;
                    d=d+1;
                end
                close(h);
                %fim da permutação inversa

                recuperado=double(yp);

                escolha2=1;

                while escolha2 == 1
                    escolha2=menu('Escolha a ação','Ouvir sinal
recuperado','Salvar como...','Plotar','Sair deste Módulo','Fechar o
Matlab');
                    if escolha2==2%Salvar como...
                        [arqwave,pasta]=uiputfile('*.wav','Salvar
Sinal Recuperado');
                        arq=strcat(pasta,arqwave);
                        wavwrite(recuperado,f,b,arq);
                        escolha2=1;
                    else
                        if escolha2==3%Plotar
                            plot(recuperado);
                            escolha2=1;
                        else
                            if escolha2==4%Sair deste módulo
                                close;
                            else
                                if escolha2==5%Fechar o MATLAB
                                    exit;
                                else
                                    if escolha2==1%Ouvir sinal
recuperado

                                       sound(recuperado,f,b);
                                    end
```

```
                                end
                            end
                        end
                    end

                end
                choose=2;
            else

                if choose==3 %submenu - opção: Sair deste módulo
                    close;
                else
                    if choose==4 %submenu - opção: Fechar o MATLAB
                        exit;
                    else
                        if choose==2 %submenu - opção: OUVIR SINAL
                          sound(sinal,f,b);
                        end

                    end
                end
            end
        end
        else
            if k==3%menu principal - opção SAIR DESTE MÓDULO
                close;
            else
              if k==4% menu principal - opção: "FECHAR O MATLAB"
                  exit;
              end
            end
        end
    end


end
```

APÊNDICE B – ESQUEMÁTICO DO PROJETO

O esquemático do projeto é apresentado na Figura B.1

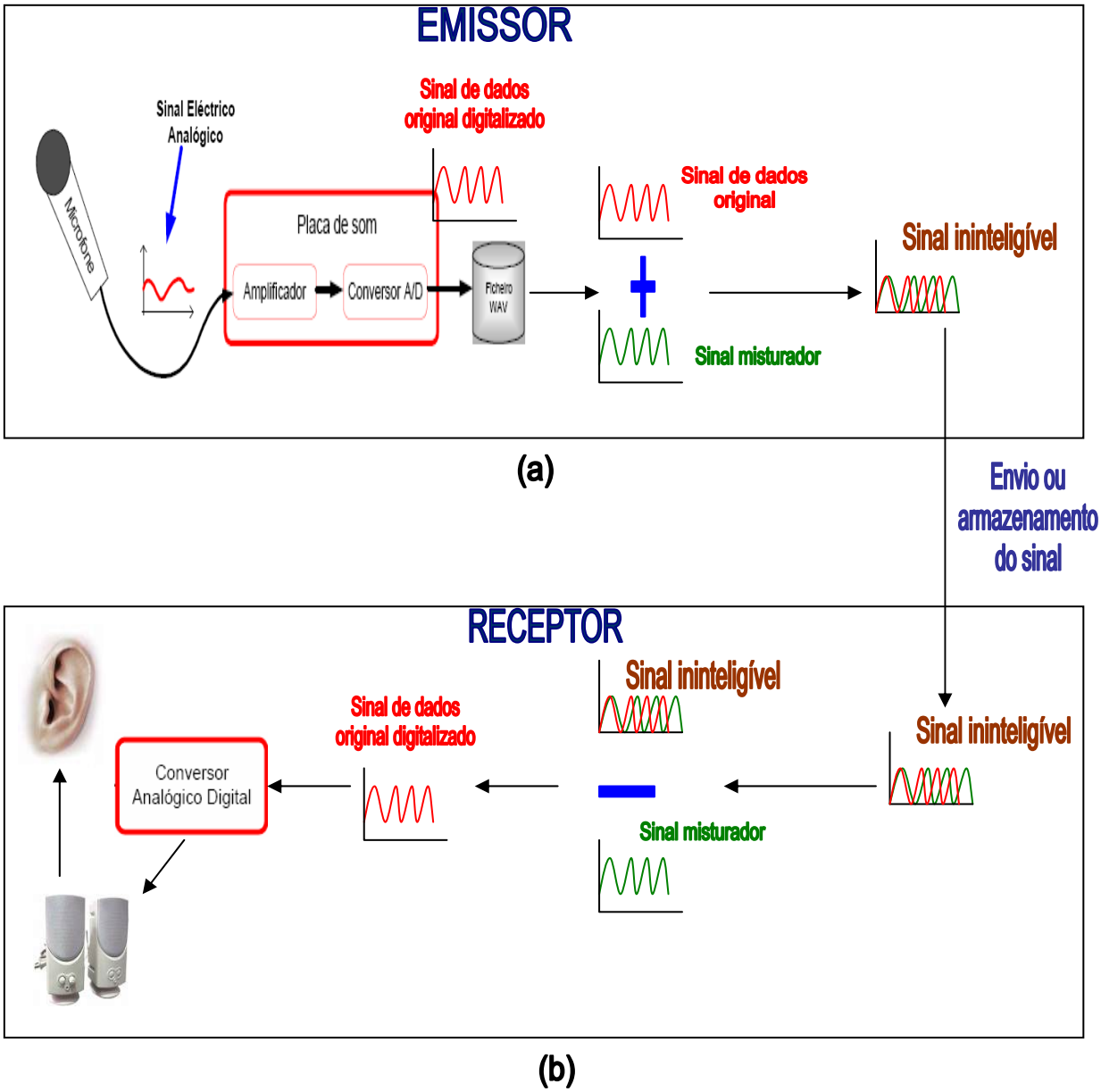


Figura B.1 – (a) Esquemático do projeto na parte do emissor
(b) Esquemático do projeto na parte do receptor

Pode-se observar na Figura B.1a que, no módulo de cifragem (Emissor) acontecem as seguintes tarefas: primeiramente, o sinal de áudio é captado por um microfone e transformado em pulsos elétricos analógicos. Em seguida, esse sinal elétrico vai para a placa de som, onde ele é amplificado e convertido do formato analógico para o digital. A representação digital do sinal é então armazenada em um arquivo Wave e é a partir deste ponto que começa a manipulação do sinal pelo algoritmo implementado. No passo seguinte, o algoritmo implementado faz a leitura e vetorização do arquivo Wave. No último passo, é feita a interposição do sinal misturador ao sinal de dados original resultando em um sinal ininteligível. Após este passo, o módulo de codificação é concluído e o sinal resultante pode ser armazenado em máquina local ou enviado a outra máquina (por e-mail, por exemplo).

No módulo de decifragem (Receptor), ilustrado na Figura B.1b os seguintes procedimentos ocorrem: é feita a leitura e vetorização do arquivo Wave que contém o sinal ininteligível. Em seguida, é feita a retirada dos sinais misturadores acrescentados ao sinal de dados original na etapa de codificação. Após este passo, obtém-se o sinal de dados original. No próximo passo, o sinal de dados original pode ser armazenado em um arquivo Wave ou enviado a placa de som. Se for enviado para a placa de som, o sinal será convertido do formato digital para o formato analógico e repassado às caixas de som. Com isso, o usuário poderá escutar o sinal de dados original recuperado.