Universidade Federal de Uberlândia

Apresentação de Senhas em Máquinas Hostis

Autor: Karla Aparecida Perine Lagares

Orientador: Prof. Dr. João Nunes de Souza

Dissertação de Mestrado apresentada à Faculdade de Computação como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Banca Examinadora

João Nunes de Souza, Dr. - FACOM/UFU
Jeroen Antonius Maria van de Graaf, Dr. - LCC/UFMG
Cícero Fernandes de Carvalho, Dr. - FAMAT/UFU

Uberlândia, MG

Setembro/2005

FICHA CATALOGRÁFICA

Elaborada pelo Sistema de Bibliotecas da UFU / Setor de Catalogação e Classificação

L173a	Lagares, Karla Aparecida Perine, 1978- Apresentação de senhas em máquinas hostis / Karla Aparecida Perine Lagares. - Uberlândia, 2005. 89f. : il. Orientador: João Nunes de Souza. Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação. Inclui bibliografia. 1. Computadores - Medidas de segurança - Teses. 2. Criptografia -Teses. 3. Ciência da Computação - Teses. I. Souza, João Nunes de. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título. CDU: 681.3-78 (043.3)

Resumo

Esta dissertação propõe um sistema de apresentação de senhas em máquinas hostis, baseado em Teoria dos Códigos, mais especificamente no conceito de arranjo. Seu objetivo é permitir que o usuário informe sua senha ao servidor sem ter que digitá-la ou informá-la de maneira explícita. O sistema visa garantir a segurança do usuário mantendo sua senha protegida em todo o processo de comunicação, considerando o fato de que a máquina do usuário é hostil.

Palavras-chave: Senha , Criptografia , Segurança , Protocolo.

Abstract

This dissertation proposes a password presentation system in hostile machine, based on Coding Theory or, more specifically, the standard array concept. The objective is to permit the user to inform his password to a server with no key in or to inform in a explicit manner. The system aim is to guarantee the user's security keeping his password protected in the whole communication process, considering the fact that the user's machine is hostile.

Keywords: Password, Cryptography, Security, Protocol.

Ao meu esposo

Agradecimentos

À Deus, por ter me dado a vida e pelas bênçãos que Ele me concede em todos os seus dias.

Aos meus pais, pela educação e formação moral.

À minha irmã, pela nossa amizade e carinho.

Ao meu esposo, pelo companheirismo, pela força, pela paciência e pelo carinho despendidos para que eu pudesse chegar até aqui.

E a todos que de alguma forma colaboraram para esta conquista, o meu muito obrigado.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

1.1 INTRODUÇÃO

A era da computação eletrônica digital, iniciada durante a Segunda Guerra Mundial, trouxe enormes avanços em nossa capacidade de “processamento de informações”. Os benefícios destes avanços são claros, e hoje, passados mais de 50 anos daqueles primórdios, a tecnologia de processamento e comunicação de dados permeia toda a sociedade.

Um problema milenar, e que se agravou com a automação do processamento da informação, é o problema do sigilo das informações. Pode-se dizer que com o surgimento da escrita, surgiu também o problema de se manter documentos protegidos do conhecimento daqueles para o qual os mesmos não fossem destinados [6].

A autenticação do usuário é uma das formas mais comuns para se prover uma acesso individual seguro e confidencial à informações pessoais ou serviços. Esta autenticação normalmente é feita através da utilização de senhas [13].

Porém, a segurança é um grande problema para usuários [20] e sistemas que utilizam serviços de senhas como bancos, comércio eletrônico e sistemas corporativos [16]. Estas senhas são utilizadas para diversos fins, dentre eles: autenticação, acesso e transações bancárias. Autenticação de senhas, por exemplo, é uma das operações mais comuns e elementares para proteger recursos de acessos não autorizados [19]. Porém, ter uma senha [13] e não utilizar mecanismos para manter o seu sigilo e autenticidade tornam os sistemas que se utilizam deste recurso bastante vulneráveis [12]. Senhas

podem ser capturadas, observadas, quebradas [8] e facilmente descobertas [11] [21].

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

O objetivo deste trabalho é utilizar a Teoria dos Códigos visando desenvolver um sistema de apresentação de senhas seguro. Neste caso, a Teoria dos Códigos [3] [2] não é utilizada para detectar e corrigir erros de transmissão [9] e sim para desenvolver um sistema de apresentação de senhas de modo a prover um nível tal de segurança, que o usuário prova conhecer a sua senha, sem ter que digitá-la ou apontá-la durante o processo de comunicação. Neste contexto, o resultado da apresentação da senha é igual à senha mais um ruído. O vetor resultante é analisado no contexto da Teoria dos Códigos e o ruído é eliminado. Para isso, o protocolo proposto é basicamente desenvolvido a partir do conceito de arranjos [2].

1.2.2 OBJETIVOS ESPECÍFICOS

- Apresentar uma visão geral de criptografia, demonstrando a sua importância na proteção da informação;
- Apresentar alguns itens relativos às estruturas algébricas utilizadas em teoria dos códigos;
- Apresentar os principais conceitos de teoria dos códigos;
- Definir um sistema de apresentação de senhas utilizando teoria dos códigos.

1.3 MOTIVAÇÃO

Após um estudo sobre manipulação de senhas e seus aspectos de segurança, foram detectadas algumas vulnerabilidades relacionadas à informação de senhas. A necessidade por propostas rela-

cionadas à apresentação de senhas de forma segura foi a motivação para esta pesquisa.

Esta proposta de dissertação tem como objetivo propor um sistema de apresentação de senhas que visa melhorar a segurança no momento em que o usuário informa sua senha, para que a mesma não seja capturada.

1.4 ARTIGO SUBMETIDO

Título: Apresentação de Senhas em Máquinas Hostis

Evento: Sétimo Simpósio Brasileiro de Segurança, a ser realizado no período de 08 a 11 de novembro de 2005.

Local: São José dos Campos - SP

1.5 ESTRUTURAÇÃO DESTE TRABALHO

Este trabalho está estruturado da seguinte forma: o Capítulo 2 apresenta uma visão geral sobre criptografia, suas principais definições e as formas de criptografia existentes. O Capítulo 3 apresenta uma abordagem sobre estruturas algébricas. O Capítulo 4 apresenta um panorama sobre teoria dos códigos e suas principais definições. O Capítulo 5 apresenta uma proposta de apresentação de senhas em máquinas hostis. É apresentado um novo sistema de apresentação de senhas que permite que o usuário informe a sua senha ao servidor sem ter que digitá-la ou informá-la de maneira explícita. O Capítulo 6 apresenta as considerações e conclusões finais deste trabalho. As referências utilizadas no decorrer do texto dizem respeito aos livros textos e artigos utilizados durante o desenvolvimento do trabalho.

Capítulo 2

CRIPTOGRAFIA

2.1 INTRODUÇÃO

A criptografia consiste num processo de codificação (cifragem) e decodificação (decifragem) de dados, que dificulta a interceptação desautorizada da informação. Este capítulo tem como objetivo apresentar os fundamentos da criptografia, sua nomenclatura, os principais sistemas criptográficos e suas características.

O capítulo está estruturado da seguinte forma: a seção 2.2 mostra a definição e os principais serviços oferecidos pela criptografia; na seção 2.3 é apresentado o conceito de criptoanálise; a seção 2.4 mostra os tipos de criptografia existentes; a seção 2.5 descreve a assinatura digital; e na seção 2.6 é apresentado o conceito da função Hash.

2.2 CRIPTOGRAFIA

Criptografia é uma palavra de origem grega: *"kriptós"* = escondido e *"grápho"* = grafia. É a arte ou ciência de escrever em cifra ou em códigos. Em outras palavras, é um conjunto de técnicas que permitem tornar uma mensagem originalmente escrita com clareza, chamada texto original (plaintext), em uma mensagem codificada, chamada texto cifrado (ciphertext), de forma a permitir

que normalmente apenas o destinatário a decifre e compreenda, com a utilização de uma chave.

Para que a mensagem seja codificada, é preciso cifrá-la. **Cifrar** é o ato de transformar dados em alguma forma ilegível. Seu propósito é o de garantir a privacidade, mantendo a informação oculta a qualquer pessoa não autorizada, mesmo que esta consiga visualizar os dados cifrados.

Para que o destinatário possa compreender a mensagem, é necessário decifrá-la. **Decifrar** é o processo inverso da cifragem, ou seja, transforma os dados cifrados na sua forma original, inteligível.

Para cifrar (**C**) ou decifrar (**D**) uma mensagem (**M**), são necessárias informações secretas chamadas chaves (**K**) ou senhas. A mesma chave pode ser utilizada tanto para cifrar como para decifrar mensagens. Entretanto, há mecanismos que utilizam chaves diferentes.

O processo de cifragem é dado por:

$$MC = \mathbf{C}_K(\mathbf{M})$$

O processo de decifragem é dado por:

$$\mathbf{M} = \mathbf{D}_K(MC)$$

A criptografia já estava presente no sistema de escrita hieroglífica dos egípcios. Desde então vem sendo muito utilizada, principalmente para fins militares e diplomáticos. A criptografia no âmbito da computação é importante para que se possa estabelecer segurança em todo o ambiente computacional. Ela é usada para garantir as informações ou dados (armazenados ou em trânsito) [11].

Alguns conceitos correntes em criptografia são [18]:

- **Confidencialidade:** garante o sigilo da informação. Somente usuários autorizados têm acesso à informação. Caso um indivíduo não autorizado consiga acessar a informação, ele não conseguirá interpretá-la ;
- **Autenticação:** processo que permite ao sistema verificar se o processo ou a pessoa com quem está se comunicando é de fato quem alega ser;
- **Integridade:** garante ao usuário que a informação correta (original) não foi alterada nem intencionalmente nem acidentalmente;
- **Não-repudiação:** evita que um emissor negue o envio de uma mensagem.

2.3 CRIPTOANÁLISE

Criptoanálise é uma palavra de origem grega: *"kryptós"* = escondido e *"análisys"* = decomposição. É a arte ou ciência de determinar a chave ou decifrar mensagens sem conhecer a chave. Uma tentativa de criptoanálise é chamada ataque. Em outras palavras, a criptoanálise trata dos ataques aos sistemas criptográficos (criptossistemas).

Segundo [1], a criptoanálise moderna parte do pressuposto de que um atacante (criptoanalista) conhece qual o criptossistema está sendo usado. É pressuposto que apenas a chave e o texto original são secretos. O atacante tenta recuperar os textos originais a partir dos textos cifrados ou tenta descobrir quais chaves estão sendo usadas.

Estes são alguns tipos de ataques [1]:

- **Ataque somente ao texto cifrado:** O atacante conhece os textos cifrados e tenta recuperar os textos originais correspondentes ou a chave;
- **Ataque com texto original conhecido:** O atacante conhece um texto original e o texto cifrado correspondente ou diversos de tais pares. Ele tenta descobrir a chave usada ou decifrar outros textos cifrados;
- **Ataque com texto original escolhido:** O atacante é capaz de cifrar textos originais sem conhecer a chave. Ele tenta descobrir a chave usada ou decifrar outros textos cifrados;
- **Ataque com texto original adaptável escolhido:** O atacante é capaz de cifrar textos originais. Ele é capaz de escolher novos textos originais como uma função dos textos cifrados obtidos, mas não conhece a chave. Ele tenta descobrir a chave ou decifrar outros textos cifrados;
- **Ataque com texto cifrado escolhido:** O atacante pode decifrar textos escolhidos, mas não conhece a chave. Ele tenta descobrir a chave.

Existem várias maneiras de montar esses ataques. Um ataque simples, somente ao texto cifrado, consiste na sua decifragem usando todas as chaves possíveis. Esse ataque é chamado *busca exaustiva*. O texto original correto está entre os poucos textos que fazem sentido que o atacante obtenha. Dada a velocidade dos computadores modernos, esse ataque é bem-sucedido sobre muitos criptossistemas.

2.4 TIPOS DE CRIPTOGRAFIA

A criptografia pode ser classificada, de acordo com o tipo de chave utilizada, em duas categorias: **simétrica** e **assimétrica**. Os criptossistemas simétricos utilizam a **criptografia convencional**. Já os criptossistemas assimétricos utilizam a **criptografia de chave pública**.

2.4.1 CRIPTOGRAFIA SIMÉTRICA

A criptografia é simétrica se a chave usada na cifragem é igual à chave usada na decifragem. Para tanto, é necessário estabelecer um acordo entre as partes que se comunicam para que a mesma chave seja utilizada, antes do início do processo de envio e recebimento de mensagens cifradas. Veja o exemplo a seguir:

Exemplo 2.1: Alice (**A**) deseja enviar uma mensagem para Bob (**B**). Então, eles devem escolher em comum uma chave secreta (k). Após a escolha da chave, **A** cifra a mensagem original **M** com K e envia a mensagem cifrada para **B**:

$$MC = \mathbf{C}_k(\mathbf{M})$$

Após receber a mensagem cifrada MC, **B** pode decifrar a mensagem utilizando a chave secreta K:

$$\mathbf{M} = \mathbf{D}_k(MC)$$

Este tipo de criptografia pode ser representado como mostra a *Figura 2.1*: **A** cifra a mensagem original com a chave *K* e envia o texto cifrado **B**. Ao receber a mensagem cifrada, **B** utiliza a mesma chave *K*para decifrar a mensagem.

A utilização da criptografia simétrica garante a confidencialidade na comunicação e possui uma implementação simples. Porém, apesar de sua simplicidade, existem alguns problemas [19]:

· Cada par necessita de uma chave secreta para se comunicar de forma segura. Portanto, estas devem ser trocadas entre as partes e armazenada de forma segura, o que nem sempre é possível de garantir;

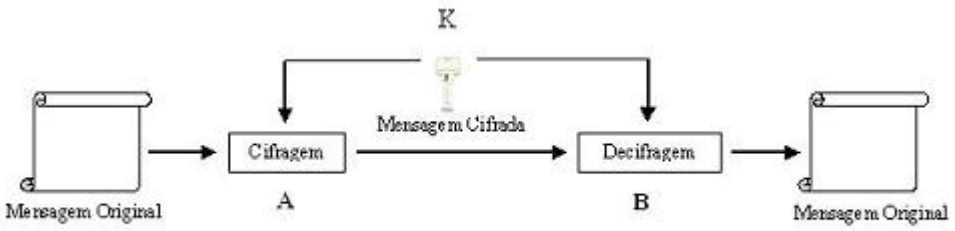

Figura 2.1: Sistema de Criptografia Simétrica

· Não garante a identidade de quem enviou ou recebeu a mensagem;

· Sua segurança está diretamente ligada ao tamanho da chave. Quanto maior for a chave, mais seguro é o criptossitema;

· A quantidade de usuários em uma rede pode dificultar o gerenciamento das chaves, devido ao número de chaves necessárias para que os mesmos se comuniquem. A quantidade de chaves para a comunicação entre n usuários é dada por [22]:

$$Q_{chaves} = \frac{n(n-1)}{2}$$

Um dos exemplos mais difundidos de algoritmo criptográfico simétrico é o Data Encryption Standard (DES) [11] . Por muitos anos, esse criptossistema foi o padrão de criptografia nos Estados Unidos e era usado em todo mundo. Hoje, o DES não é mais seguro. Em outubro de 2000, o secretário de comércio dos Estados Unidos anunciou o novo Padrão Avançado de Codificação Criptográfica (Advanced Encryption Standard) proposto à nação. Ele declarou vencedora a fórmula Rijndael de criptografia de dados (desenvolvida por Joan Daemen e Vincent Rijmen) escolhida em um concurso que levou três anos. Entretanto, existem muitas variantes seguras do DES e a maioria dos sucessores sugeridos para o DES são similares a ele. Portanto, o DES ainda é um criptossistema importante [1].

Outros exemplos de algoritmos simétricos são: 3DES, IDEA, RC5 e RC6.

2.4.2 CRIPTOGRAFIA ASSIMÉTRICA

A criptografia é assimétrica se utiliza um par de chaves: uma pública, que cifra a mensagem, e uma privada, que decifra a mensagem; ou vice-versa. A chave privada deve ser mantida em segredo, enquanto a chave pública deve ser publicada.

Este tipo de criptografia foi inventado em 1976 por Whitfield Diffie e Martin Hellman [19]. Veja o exemplo a seguir:

Exemplo 2.2: Alice (**A**) deseja enviar uma mensagem para Bob (**B**). **B** possui duas chaves, uma chave pública KU_B que é conhecida por **A**, e uma chave privada KR_B que somente ele conhece. Para cifrar a mensagem original **M** a ser enviada para **B**, **A** utiliza a chave pública de **B**:

$$MC = \mathbf{C}_{KU_B}(\mathbf{M})$$

Ao receber a mensagem cifrada MC, **B** utiliza sua chave privada para obter a mensagem decifrada **M**:

$$\mathbf{M} = \mathbf{D}_{KR_B}(MC)$$

Este tipo de criptografia pode ser representado como mostra a *Figura 2.2*: Para cifrar a mensagem original, **A** utiliza a chave pública de **B**. Apenas **B** poderá decifrar a mensagem cifrada utilizando sua chave privada.

Nos sistemas de chave pública, a gerência de chaves é muito mais simples do que nos sistemas simétricos. Eles não só simplificam a gerencia de chaves, mas podem ser usados para gerar assinaturas digitais [1].

Os algoritmos de chave pública e privada exploram propriedades específicas dos números primos e, principalmente, a dificuldade de fatorá-los, mesmo em computadores rápidos. Outros problemas matemáticos em que os algoritmos podem se basear são o logaritmo discreto e as curvas elípticas [1].

Um dos principais exemplos de algoritmo assimétrico é o RSA, batizado com os nomes de seus inventores Ron Rivest, Adi Shamir e Len Adleman. O algoritmo foi publicado pela primeira vez

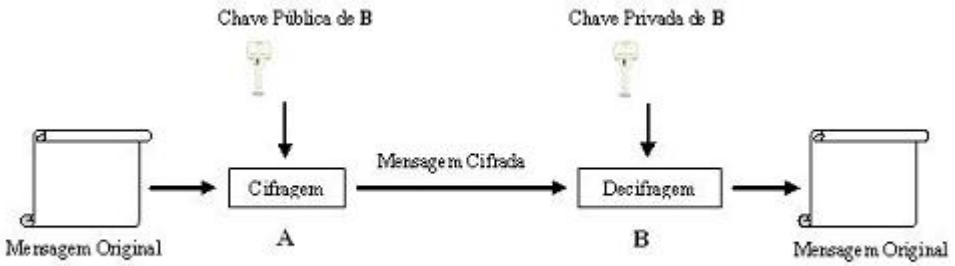

Figura 2.2: Sistema de Criptografia Simétrica

em 1978 [14]. Foi o primeiro criptossistema de chave pública e ainda é um dos mais importantes. Pode ser usado para esquemas de assinaturas digitais, autenticações e em mecanismos de segurança de e-mail [22].

Sua segurança está intimamente relacionada à dificuldade de encontrar a fatoração de um número inteiro positivo múltiplo, que é o produto de dois números primos grandes [1].

O RSA foi patenteado em 1983 e até hoje não revelou qualquer falha importante. É possível que um dia alguém descubra uma forma de quebrar o RSA sem usar a força bruta. Tal achado revolucionaria a atual Teoria dos Números e garantiria fama e fortuna para o descobridor [19].

Outros exemplos de algoritmos assimétricos são: Rabin, ElGamal, Curvas Elípticas, McEliece (que utiliza Teoria dos Códigos), entre outros.

2.4.3 CRIPTOGRAFIA SIMÉTRICA X CRIPTOGRAFIA ASSIMÉTRICA

Atualmente, inúmeros sistemas criptográficos utilizam como modelo de criptografia um modelo denominado híbrido. Nestes modelos são utilizados conjuntamente os dois tipos de criptografia, fazendo uso das boas características de cada uma. Um exemplo é aquele em que os sistemas de chave pública são utilizados para a distribuição de chaves secretas. Em seguida, estas chaves secretas são utilizadas por sistemas simétricos na cifragem dos dados a serem transmitidos.

Os modelos híbridos utilizam as principais características dos sistemas criptográficos simétricos e assimétricos. A comparação entre as duas categorias de criptografia é apresentada na *Tabela 2.1*.

Tabela 2.1: Comparação entre criptografia simétrica e assimétrica

	Simétrica	**Assimétrica**
Cifragem/Decifragem	Rápida	Lenta
Gerência e distribuição das chaves	Complexa	Simples
Número de Chaves	1(secreta)	2(privada e pública)
Vulnerabilidades	Maior	Menor
Assinatura Digital	Não oferece	Oferece

Os algoritmos criptográficos podem ser combinados para a implementação dos três mecanismos criptográficos básicos: o ciframento, o decifrамento e a assinatura. Estes mecanismos são, por exemplo, componentes dos protocolos criptográficos, embutidos na arquitetura de segurança dos produtos destinados ao comércio eletrônico [18]. Estes protocolos criptográficos, portanto, provêm os serviços associados à criptografia que viabilizam o comércio eletrônico: disponibilidade, sigilo, controle de acesso, autenticidade, integridade e não-repúdio.

2.5 ASSINATURA DIGITAL

A assinatura digital é definida utilizando criptografia com chave pública e permite garantir a autenticidade de quem envia a mensagem. As assinaturas digitais são utilizadas para assinar documentos eletrônicos como, por exemplo, contratos eletrônicos, transações bancárias e correspondências eletrônicas que envolvam responsabilidade civil [1].

Uma assinatura digital pode ser conseguida quando se utiliza a chave privada (KR) de quem assina para cifrar uma mensagem. A verificação da assinatura é feita com a chave pública (KU) correspondente. A assinatura é considerada autêntica se o resultado da decifragem corresponde à mensagem original. A garantia está no fato de que apenas o detentor da chave privada (correspondente à chave pública utilizada) poderia ter gerado aquela mensagem cifrada. Veja o exemplo a seguir:

Exemplo 2.3: Alice (**A**) deseja mandar uma mensagem assinada para Bob (**B**). A possui duas chaves, uma chave pública KU_A que é conhecida por **B** e uma chave privada KR_A que somente ela conhece. Para gerar uma assinatura da mensagem original **M** a ser enviada para **B**, **A** utiliza a sua

chave privada e calcula MC:

$$MC = \mathbf{C}_{KR_A}\ (\mathbf{M})$$

O valor MC é a assinatura de **M**. Ao receber a assinatura MC, **B** utiliza a chave pública de **A** para obter a mensagem decifrada **M**. A assinatura somente é autëntica quando a igualdade a seguir é verificada.

$$\mathbf{M} = \mathbf{D}_{KU_A}\ (MC)$$

Este processo pode ser representado como mostra a *Figura 2.3*.

Na *Figura 2.3*, para assinar a mensagem original **M**, A utiliza sua própria chave privada. **B** poderá decifrar a assinatura MC, utilizando a chave pública de **A**.

A assinatura digital empregada como foi mostrada acima não garante a confidencialidade da mensagem. Dado que a chave pública é de conhecimento público, qualquer um pode decifrar a assinatura. No entanto, tem-se certeza de que a cifragem foi realizada pelo detentor da chave privada correspondente. O algoritmo usado para gerar a assinatura precisa garantir que, se a chave privada não for conhecida, não seja possível criar uma assinatura válida.

Existem vários métodos para gerar e verificar assinaturas digitais como Schnorr, ElGamal e Fiat-Shamir [1].

2.6 FUNÇÃO HASH

Função Hash ou resumo de mensagem é uma técnica que, ao ser aplicada, gera um resumo criptografado de tamanho fixo e pequeno de uma mensagem de qualquer tamanho. Ela funciona como uma impressão digital de uma mensagem, pois gera, a partir de uma entrada de tamanho variável, um valor fixo pequeno: o *digest* ou *valor hash*. Este valor está para o conteúdo da mensagem assim como o dígito verificador de uma conta-corrente está para o número da conta [19].

A função hash oferece agilidade nas assinaturas digitais e garante a integridade do documento recebido. Ela pode ser representada por:

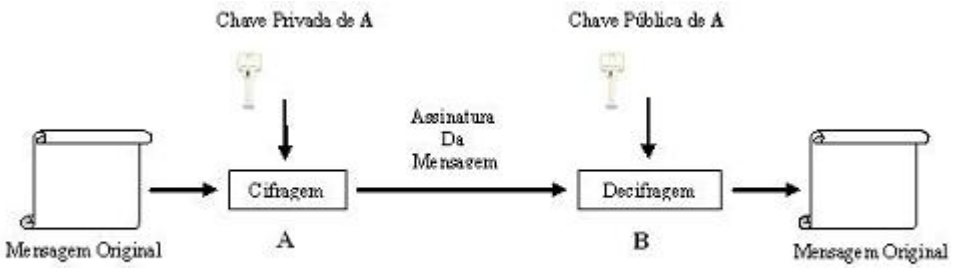

Figura 2.3: Sistema de Assinatura Digital

$$h = \mathbf{H}(\mathbf{M})$$

onde **M** é uma mensagem de tamanho variável e **H(M)** é o valor hash de tamanho fixo. Suas propriedades básicas são [19]:

· difícil de fazer a operação reversa (função unidirecional), ou seja, dado um resumo é muito difícil obter a mensagem original;

· uma pequena modificação na mensagem (**M**) para **M'** faz com que **H(M)** seja diferente de **H(M')**. Assim, após o resumo de uma mensagem ter sido calculado através do emprego de uma função hash, qualquer modificação em seu conteúdo - mesmo em apenas um bit da mensagem - será detectada, pois um novo cálculo do resumo sobre o conteúdo modificado resultará em um resumo bastante distinto.

O exemplo a seguir mostra uma utilização da função hash:

Exemplo 2.4: Suponha que Alice (**A**) envia um resumo (h) e a mensagem (**M**), ambos cifrados com sua chave privada KR_A para Bob (**B**). **B** decifra tudo com a chave pública de **A**. Em seguida, **B** calcula um novo resumo h' com base na mensagem recebida (**M**) e depois compara os dois valores. Se estes valores forem iguais ($h = h'$), significa que a mensagem não sofreu alterações, o que garante a integridade da mensagem. Observe que neste exemplo, Alice assina a mensagem **M** juntamente com o resumo h.

Este processo pode ser representado como mostra a *Figura 2.4*: Na **A** calcula o resumo da mensagem **M** utilizando a função **H**. A mensagem original **M** é concatenada com o resumo h. O resultado

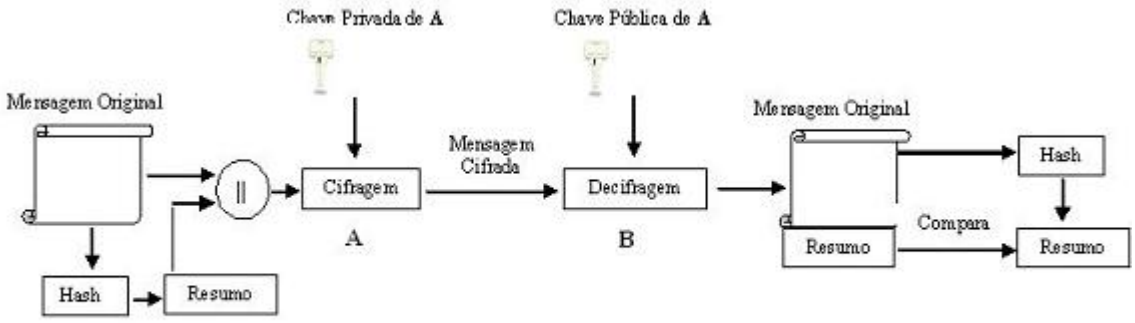

Figura 2.4: Assinatura Digital utilizando Função Hash

é cifrado com a chave privada de **A**, KR_A e enviado para **B**. Este, por sua vez, decifra o pacote recebido com a chave pública de **A**, KU_A. Em seguida, calcula o resumo h' da mensagem **M** recebida e compara os dois resultados. Se eles forem iguais ($h = h'$), **B** tem certeza que a mensagem não foi alterada.

Muitas funções hash podem ser obtidas gratuitamente. As mais famosas são as da família MD (Message Digest), de Ron Rivest. São elas: MD5, MD4 (precursor do MD5) e MD2 [19]. Pode-se citar também o SHA-1 (Secure Hash Algorithm) [1]. A seguir serão apresentadas algumas informações a respeito de cada um dos algoritmos citados acima:

· MD2: É uma função de dispersão unidirecional simplificada e produz um resumo de 128 bits. Sua segurança é dependente de uma permutação aleatória de bytes. É a mais lenta e menos segura da funções hash citadas;

· MD4: Não é mais utilizado. Após terem sido descobertas algumas fraquezas, Ron Rivest escreveu o MD5;

· MD5: É uma função de dispersão unidirecional que produz um resumo de 128 bits, para uma mensagem de qualquer tamanho.

· SHA-1: É uma função de dispersão unidirecional que gera um resumo de 160 bits a partir de uma mensagem de tamanho qualquer.

2.7 CONCLUSÃO

Neste capítulo foram apresentados os principais termos utilizados na criptografia, a definição e classificação de alguns métodos criptográficos. O próximo capítulo apresenta alguns conceitos de Estruturas Algébricas, necessárias ao entendimento de alguns conceitos relativos à Teoria dos Códigos.

Capítulo 3

ESTRUTURAS ALGÉBRICAS

3.1 INTRODUÇÃO

As estruturas algébricas são um ramo da Matemática que estuda corpos, grupos, conjuntos, aritmética modular, vetores, espaços vetoriais, transformações lineares, entre outros.

A Teoria dos Códigos que é apresentada no capítulo 4 utiliza uma série de conceitos de estruturas algébricas. Este capítulo tem como objetivo apresentar alguns de seus fundamentos, necessárias ao estudo da teoria dos códigos.

O capítulo está estruturado da seguinte forma: a seção 3.2 mostra a definição de conjunto; na seção 3.3 é apresentado o conceito de grupo; a seção 3.4 mostra o que é um anel; a seção 3.5 define corpo; na seção 3.6 é apresentado o conceito de números primos; na seção 3.7 define-se aritmética modular e suas propriedades; a seção 3.8 mostra a definição de vetor; a seção 3.9 mostra o que é matriz e algumas operações relacionadas; a seção 3.10 descreve espaço vetorial; a seção 3.11 define espaço dual; e, finalmente, na seção 3.12 mostra o que é uma transformação linear.

3.2 CONJUNTO

Um **conjunto S** é definido como sendo uma coleção de elementos $\{a_1, a_2, ..., a_n\}$. Dependendo do número de elementos contidos em S este é dito ser um conjunto *finito* ou *infinito*. Se um elemento a_i pertence ao conjunto S, representa-se isso como $a_i \in S$ ou $a_i \in \{a_1, a_2, ..., a_n\}$ [6].

Seja S um conjunto. Uma operação binária "$\circ$" sobre elementos de S é uma regra que associa 2 elementos a e b de S a um terceiro elemento c, único, definido por $c = a \circ b$ e $c \in S$. Neste caso, a operação $\circ$ é dita ser fechada em S.

Além disso, uma operação binária $\circ$ é dita ser associativa se, e somente se, para quaisquer elementos a, b e c do conjunto S, pode-se expressar a relação

$$(a \circ b) \circ c = a \circ (b \circ c).$$

Se para todo $a, b \in S$, tem-se $a \circ b = b \circ a$, a operação é dita ser *comutativa*.

3.3 GRUPO

Um conjunto não-vazio G associado a uma operação binária $\circ$ é chamado grupo $G = \{G, \circ\}$ se os seguintes axiomas forem satisfeitos [6]:

i. A operação binária $\circ$ é fechada;

ii. A operação binária $\circ$ é associativa;

iii. G contém o elemento identidade e, único, tal que para qualquer elemento $a \in G$, temos $a \circ e = e \circ a = a$;

iv. Para qualquer elemento $a \in G$, existe o elemento inverso, único, $a^{-1} \in G$, tal que $a \circ a^{-1} = a^{-1} \circ a = e$.

Um grupo G é chamado **Abeliano** se a operação binária $\circ$ for comutativa em G. Geralmente, se a operação em G for "adição", G é dito ser um grupo *aditivo*, e se a operação for "multiplicação", G é dito um grupo *multiplicativo*. Além disso, em um grupo aditivo, geralmente representamos o elemento identidade pelo símbolo "0" (zero) e o elemento inverso de um elemento "a" por "$-a$",

enquanto que num grupo multiplicativo usamos "1" (um) como elemento identidade e "a^{-1}" como o inverso de "a".

Se A e B são subconjuntos de um grupo G, então escreve-se

$AB = \{ab : a \in A, b \in B\}$, ou $A + B = \{a + b : a \in A, b \in B\}$

Também escreve-se a em vez de $\{a\}$.

Exemplo 3.1: O conjunto $G = \{0, 1, 2, 3\}$ com a operação "soma módulo-4" forma a estrutura de um Grupo Abeliano Aditivo, como apresentado na *Tabela 3.1*.

3.3.1 SUBGRUPO

Um subconjunto H de um grupo G é chamado **subgrupo** de G, se o próprio H forma um grupo com a operação induzida de G. Se H é subgrupo de G e $a \in G$, então Ha é chamado **classe lateral à direita de H** e o conjunto aH é chamado **classe lateral à esquerda de H**.

Um subgrupo H de G é chamado subgrupo normal, se $a^{-1}Ha \subset H$ para todo $a \in G$. Equivalentemente, H é normal, se $aH = Ha$ para todo $a \in G$, isto é, se as classes laterais à direita e à esquerda de H coincidem.

Exemplo 3.2:[23] As permutações de n símbolos formam um grupo sob composição de aplicações; é chamado grupo simétrico de grau n e é denotado por S_n. Os elementos de S_3 são:

$$\xi = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$$

$$\tau_2 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

$$\phi_1 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$$

Tabela 3.1: Exemplo de um Grupo Abeliano de 4 elementos

+	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$$\tau_1 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}$$

$$\tau_3 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}$$

$$\phi_2 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

Aqui, $\begin{pmatrix} 1 & 2 & 3 \\ i & j & k \end{pmatrix}$ é a permutação que transforma $1 \longmapsto i, 2 \longmapsto j, 3 \longmapsto k$. A tabela de multiplicação de S_3 é mostrada na *Tabela 3.2*.

(O elemento na a-ésima linha e b-ésima coluna é ab.) O conjunto $H = \{\xi, \tau_1\}$ é um subgrupo de S_3; suas classes laterais à direita e à esquerda são:

Classes laterais à direita	Classes laterais à esquerda
$H = \{\xi, \tau_1\}$	$H = \{\xi, \tau_1\}$
$H\phi_1 = \{\phi_1, \tau_2\}$	$\phi_1 H = \{\phi_1, \tau_3\}$
$H\phi_2 = \{\phi_2, \tau_3\}$	$\phi_2 H = \{\phi_2, \tau_2\}$

Observe que as classes laterais à direita e as classes laterais à esquerda são distintas; portanto, H não é um subgrupo normal de S_3.

Tabela 3.2: Tabela de Multiplicação de S_3

	ξ	τ_1	τ_2	τ_3	ϕ_1	ϕ_2
ξ	ξ	τ_1	τ_2	τ_3	ϕ_1	ϕ_2
τ_1	τ_1	ξ	ϕ_1	ϕ_2	τ_2	τ_3
τ_2	τ_2	ϕ_2	ξ	ϕ_1	τ_3	τ_1
τ_3	τ_3	ϕ_1	ϕ_2	ξ	τ_1	τ_2
ϕ_1	ϕ_1	τ_3	τ_1	τ_2	ϕ_2	ξ
ϕ_2	ϕ_2	τ_2	τ_3	τ_1	ξ	ϕ_1

3.4 ANEL

Um anel é formado por um conjunto R com duas operações binárias, '$+$' e '.', soma e multiplicação respectivamente, sobre os elementos de R. Além disso, deve satisfazer os seguintes axiomas [6]:

$i.$ $\{R, +\}$ forma um grupo Abeliano;

$ii.$ A operação de multiplicação é fechada em R;

$iii.$ A multiplicação é associativa;

$iv.$ A multiplicação é distributiva sobre a soma, de tal forma que

$a.(b + c) = (a.b) + (a.c), a, b, c \in R.$

Se a operação de multiplicação for também comutativa, então R é dito ser um Anel Comutativo.

Exemplo 3.3: O conjunto $\{0, 1, 2, 3\}$ com as operações de soma e multiplicação módulo-4, forma um anel comutativo, como mostra a *Tabela 3.3*.

3.5 NÚMERO PRIMO

Seja p um inteiro tal que $p \neq 1$ e $p \neq -1$. Diz-se que p é **primo** se os seus únicos divisores são $1, -1, p$ e $-p$. Por exemplo, $2, 3$ e -11 são primos, enquanto que 35 não é primo pois 5 é divisor de 35 $(5|35)$. Um número que não é primo é dito **composto**. Assim 35 é um número composto. Os números 1 e -1 não são primos nem são compostos.

Propriedade fundamental dos números primos [5]: Seja p um número primo e a e b inteiros

positivos. Se p é divisor de $(a.b)$ (em símbolos $p|a.b$), então p é divisor de a ou p é divisor de b.

3.6 ARITMÉTICA MODULAR

Seja um número inteiro positivo n e qualquer inteiro a. Se a for dividido por n, obtém-se um quociente q e um resto r, tal que $a = qn + r$, $0 \leq r < n$; $q = \lfloor a/n \rfloor$.

Dados a e n conforme indicado acima, sempre é possível encontrar q e r que satisfaçam a relação acima.

Se a é um inteiro e n é um inteiro positivo, define-se $a\ mod\ n$ como o resto obtido da divisão de a por n. Assim, para qualquer inteiro a, pode-se escrever que $a = \lfloor a/n \rfloor\ .n$ + (a mod n).

O operador módulo possui as seguintes propriedades:

- $a \equiv$ b mod n se n|(a-b);
- Se a mod n = b mod n então $a \equiv$ b mod n ;
- Se $a \equiv$ b mod n então b $\equiv$ a mod n ;
- Se ($a \equiv$ b mod n) e (b $\equiv$ c mod n)$ento$ a $\equiv$ c mod n.

3.6.1 OPERAÇÕES DA ARITMÉTICA MODULAR

O operador $(mod\, n)$, por definição, mapeia todos os inteiros em um conjunto de inteiros $\{0, 1, ..., (n-1)\}$. A aritmética modular possui as seguintes propriedades:

- $\lfloor$(a mod n) + (b mod n) mod $n = (a + b)$ mod n
- $\lfloor$ (a mod n) - (b mod n) mod $n = (a - b)$ mod n
- $\lfloor$ (a mod n) . (b mod n) mod n = (a.b) mod n

3.7 CORPO

Um conjunto não vazio K é um corpo se em K pode-se definir duas operações, denotadas por $+$ (adição) e $.$ (multiplicação), satisfazendo as seguintes propriedades [10]:

Tabela 3.3: Exemplo de um Anel Comutativo de 4 elementos.

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

a) Da adição:

- $a + b = b + a, a, b \in K$ (propriedade comutativa).
- $a + (b + c) = (a + b) + c, a, b, c \in K$(propriedade associativa).
- Existe um elemento em K, denotado por 0 e chamado de elemento neutro da adição, que satisfaz

$$0 + a = a + 0 = a, a \in K.$$

- Para cada a K, existe um elemento em K, denotado por $-a$ e chamado de *oposto* de a (ou *inverso aditivo* de a) tal que

$$a + (-a) = (-a) + a = 0.$$

b) Da multiplicação:

- $a.b = b.a, a, b \in K$ (propriedade comutativa).
- $a.(b.c) = (a.b).c, a, b, c \in K$(propriedade associativa).
- Existe um elemento em K, denotado por 1 e chamado de *elemento neutro* da multiplicação, que satisfaz

$$1.a = a.1 = a, a \in K.$$

- Para cada elemento não nulo de $a \in K$, existe um elemento em K, denotado por a^{-1} e chamado de inverso multiplicativo de a, tal que

$$a.a^{-1} = a^{-1}.a = 1.$$

c) Distributiva:

- $(a + b).c = a.c + b.c, a, b, c \in K$

Observações:

- A notação do produto é simplificada utilizando-se simplesmente ab para o produto $a.b$.
- Os elementos de um corpo arbitrário são chamados *escalares* [23].

3.7.1 O GRUPO MULTIPLICATIVO DE UM CORPO

Pode-se verificar que o conjunto dos elementos não nulos de um corpo K formam um grupo comutativo sob a multiplicação. A partir daí, podem-se definir os seguintes conceitos [6]:

- No corpo K, o elemento inverso aditivo de um dado elemento a é representado por $-a$. A operação *subtração* de um dado elemento b de um outro elemento a é definida pela adição de a ao elemento inverso aditivo de b, representada por

$$a - b = a + (-b)$$

- O elemento inverso multiplicativo de um elemento $a \neq 0$ é designado por a^{-1} . A operação divisão de a por um dado elemento não-nulo b, no corpo K, é definida pela multiplicação de a pelo elemento inverso multiplicativo de b, ou,

$$a \div b = a.b^{-1}$$

- A ordem de um corpo K é o número de elementos distintos contidos no corpo. Este número pode ser finito ou infinito. O corpo cuja ordem q é finita é dito um **corpo finito** ou **corpo de Galois**, sendo designado por $GF(q)$.

Com estes conceitos, pode-se verificar que o conjunto Z dos números inteiros, junto com as operações de adição e multiplicação de inteiros não se constitui em um corpo, sendo apenas um anel comutativo. O conjunto R dos números reais, associado às operações aritméticas convencionais de adição e multiplicação de números reais, define a estrutura de um corpo de ordem infinita.

3.7.2 PROPRIEDADES DOS CORPOS FINITOS

A ordem q de um corpo finito (corpo de Galois) pode assumir apenas valores da forma $q = p^n$, onde p é um primo e n um inteiro positivo [6]. Para $q = p$, $GF(p)$ é isomorfo a Z_p, onde $Z_p = \{\overline{0}, \overline{1}, ..., \overline{p-1}\}$ e $+$, $*$ são a adição e a multiplicação módulo-p, respectivamente. No caso de $p = 2$, define-se o chamado Corpo Binário $GF(2)$, formado pelo conjunto $\{\overline{0}, \overline{1}\}$ associado às operações aritméticas de adição e multiplicação módulo-2.

Exemplo 3.4: A seguir serão construídos as estruturas de $GF(2)$ e $GF(3)$. Como 2 e 3 são primos, basta tomarmos Z_2 e Z_3 junto com as operações módulo-2 e módulo-3, para obter as estruturas desejadas, como mostram as *Tabelas 3.4 e 3.5*.

Se $q = p^n$, $n > 1$, então **não** se obtém a estrutura de um corpo finito tomando-se os inteiros e operações módulo-q [6].

Para $q = 4 = 2.2$, não existe nenhum elemento $x \in Z_4$ tal que $2.x = x.2 = 1$ mod-4, e portanto, 2 não tem inverso multiplicativo. Também, $2.2 = 0$ e 2 é um divisor de 0!

Algumas propriedades especiais podem ser demonstradas em corpos finitos [6]:

$GF(2)$:

- como $\overline{0} + \overline{0} = \overline{0}$ e $\overline{1} + \overline{1} = \overline{0}$, então $\overline{x} + \overline{x} = \overline{0}$ e $\overline{x} = -\overline{x}$;
- como $\overline{0}.\ \overline{0} = \overline{0}$ e $\overline{1}.\overline{1} = \overline{1}$, então $\overline{x}.\overline{x} = \overline{x}$, e a multiplicação é idempotente;

$GF(3)$:

- como $\overline{x} + \overline{x} = 2\overline{x}$ e $\overline{x} + \overline{x} + \overline{x} = \overline{x} + 2\overline{x} = \overline{0}$, então $2\overline{x} = -\overline{x}$;
- como $0.0 = 0, 1.1 = 1$ e $2.2 = 1$, então $x.x = x^2$ tem apenas dois valores possíveis, e $x.x^2 = x^3 = x$, e assim todas as potências pares de x são equivalentes a x^2, e todas as potências ímpares são equivalentes a x;

$GF(4)$:

- $\overline{0} + \overline{0} = \overline{0}, \overline{1} + \overline{1} = \overline{0}, A + A = \overline{0}$e $B + B = \overline{0}$, e $\overline{x} + \overline{x} = \overline{0}$ como $GF(2)$, e $\overline{x}$= -$\overline{x}$.

Esta propriedade é exibida por todos os corpos de característica 2, $GF(2^n)$, o que simplifica bastante

Tabela 3.4: Exemplo de um Anel Comutativo de 2 elementos com estrutura $GF(2)$

+	$\overline{0}$	$\overline{1}$
$\overline{0}$	0	1
$\overline{1}$	1	0

.	0	$\overline{1}$
$\overline{0}$	0	0
$\overline{1}$	0	1

Tabela 3.5: Exemplo de um Anel Comutativo de 3 elementos com estrutura $GF(3)$

+	$\overline{0}$	$\overline{1}$	$\overline{2}$
$\overline{0}$	0	1	2
$\overline{1}$	1	2	0
$\overline{2}$	2	0	1

.	$\overline{0}$	$\overline{1}$	$\overline{2}$
$\overline{0}$	0	0	0
$\overline{1}$	0	1	2
$\overline{2}$	0	2	1

a manipulação algébrica nestes corpos, mesmo para corpos de ordem maior;

$GF(q)$:

- $\forall a, b \in K$, $\exists c$ único tal que $a + c = b$;
- $\forall a, b \in K$, distintos de 0, $\exists d$ único tal que $a.d = b$;
- $\forall a \in K$, $a.0 = 0$. $a = 0$;
- $\forall a, b \in K$, $-a = (-1).a$

$$-(a + b) = (-a) + (-b) = -a - b;$$

3.8 VETOR

Um vetor é uma n-upla $(a_1, a_2, ..., a_n)$. As coordenadas desta n-upla podem ser números reais, complexos ou pertencentes a um corpo qualquer.

3.8.1 VETORES NO R^n

O conjunto de todas as n-uplas de números reais, representado por R^n, é chamado n-espaço. Uma n-upla em R^n é denotada por

$$\overrightarrow{u} = (u_1, u_2, ..., u_n)$$

Neste caso, $\overrightarrow{u}$ é chamado de **vetor**. Os números reais u_i são chamados *componentes* (ou coordenadas) do vetor .

3.8.2 IGUALDADE

Dois vetores $\overrightarrow{u}$ e $\overrightarrow{v}$ são iguais ($\overrightarrow{u} = \overrightarrow{v}$), se eles têm o mesmo número de componentes e se os componentes correspondentes são iguais.

3.8.3 PRODUTO INTERNO

Sejam $\overrightarrow{u}$ e $\overrightarrow{v}$ dois vetores de R^n :

$$\overrightarrow{u} = (u_1, u_2, ..., u_n) \text{ e } \overrightarrow{v} = (v_1, v_2, ..., v_n)$$

O **produto interno** ou produto escalar de $\overrightarrow{u}$ e $\overrightarrow{v}$, é o número $u.v$ obtido multiplicando os componentes e somando os produtos obtidos:

$$\overrightarrow{u}.\overrightarrow{v} = u_1.v_1 + u_2.v_2 + ... + u_n.v_n$$

Diz-se que os vetores $\overrightarrow{u}$ e $\overrightarrow{v}$ são **ortogonais** (ou perpendiculares) se seu produto interno é zero: $\overrightarrow{u}.\overrightarrow{v} = 0$.

As propriedades básicas do produto interno em R^n são as seguintes [23]:

- $(\overrightarrow{u} + \overrightarrow{v}).\overrightarrow{w} = \overrightarrow{u}.\overrightarrow{w} + \overrightarrow{v}.\overrightarrow{w}$
- $(k\overrightarrow{u}).\overrightarrow{v} = k.(\overrightarrow{v}.\overrightarrow{u})$, onde $k \in$ R
- $\overrightarrow{u}.\overrightarrow{v} = \overrightarrow{v}.\overrightarrow{u}$
- $\overrightarrow{u}.\overrightarrow{u} \geq 0$, e $\overrightarrow{u}.\overrightarrow{u} = 0$ se, e somente se, $\overrightarrow{u} = 0$

3.9 MATRIZ

Sejam m,n dois inteiros positivos e seja K um corpo. Uma **matriz A** m por n sobre K é dada por $m.n$ valores $a_{ij} \in K$, com $1 \leq i \leq m$, $1 \leq j \leq n$, agrupados em m linhas e n colunas que será representada como [10]:

$$A = (a_{ij})_{m,n} = \begin{pmatrix} a_{11} & ... & a_{1n} \\ ... & ... & ... \\ a_{m1} & ... & a_{mn} \end{pmatrix}$$

O conjunto das matrizes m por n sobre K é denotado por $M_{mxn}(K)$.

3.9.1 SOMA DE MATRIZES

Seja $A = (a_{ij})_{m,n}$, $B = (b_{ij})_{m,n} \in M_{mxn}(K)$, então a **soma A + B** é a matriz $C = (c_{ij})_{m,n} \in M_{mxn}(K)$, tal que, para cada par (i, j), temos $c_{ij} = a_{ij} + b_{ij}$.

3.9.2 MULTIPLICAÇÃO POR ESCALAR

Seja $A = (a_{ij})_{m,n} \in M_{mxn}(K)$ e $\lambda \in K$ pode-se definir o **produto** de λ por A como sendo a matriz $B = (b_{ij})_{m,n} \in M_{mxn}(K)$, tal que, para cada par (i, j), tem-se $b_{ij} = \lambda a_{ij}$.

3.9.3 MATRIZ TRANSPOSTA

Dada uma matriz $A = (a_{ij})_{m,n} \in M_{mxn}(K)$, define-se sua **transposta** como sendo a matriz $A^t = (b_{ij})_{m,n} \in M_{mxn}(K)$ tal que $b_{ij} = a_{ji}$ para cada par (i, j).

3.10 ESPAÇO VETORIAL

Para a definição a seguir, é adotada a seguinte notação [23]:

K, o corpo dos escalares ;

a, b, c ou k, os elementos de K;

V, o espaço vetorial dado;

$\overrightarrow{u}, \overrightarrow{v}$, $\overrightarrow{w}$ os elementos de V.

Seja K um corpo e seja V um conjunto não–vazio com operações de adição e multiplicação por escalar que determinam para qualquer , $\overrightarrow{u}, \overrightarrow{v} \in V$ uma soma $\overrightarrow{u} + \overrightarrow{v} \in V$ e para qualquer $\overrightarrow{u} \in V$, $k \in K$ um produto $k\overrightarrow{u} \in V$. Então, V é chamado **espaço vetorial sobre K(** e os elementos de V são chamados vetores) se:

a) Com relação à estrutura aditiva de V:

- Para quaisquer vetores $\overrightarrow{u}, \overrightarrow{v}$, $\overrightarrow{w} \in V$,

$$(\overrightarrow{u} + \overrightarrow{v}) + \overrightarrow{w} = \overrightarrow{u} + (\overrightarrow{v} + \overrightarrow{w});$$

- Há um vetor em V, denotado $\overrightarrow{0}$ e chamado *vetor nulo*, para o qual $(\overrightarrow{u} + \overrightarrow{0}) = \overrightarrow{u}$ para qualquer vetor $\overrightarrow{u} \in V$;

- Para cada vetor $\overrightarrow{u} \in V$ há um vetor em V, denotado $-\overrightarrow{u}$, para o qual

$$\overrightarrow{u} + (-\overrightarrow{u}) = 0;$$

- Para quaisquer vetores , $\overrightarrow{u}, \overrightarrow{v} \in V$,

$$(\overrightarrow{u} + \overrightarrow{v}) = (\overrightarrow{v} + \overrightarrow{u});$$

Os ítens acima podem ser resumidos dizendo-se que V é um **grupo comutativo** com a operação "+".

b) Com relação à ação do corpo K em V:

- Para qualquer escalar $k \in K$ e quaisquer vetores , $\overrightarrow{u}, \overrightarrow{v} \in V$,

$$k(\overrightarrow{u} + \overrightarrow{v}) = k\overrightarrow{u} + k\overrightarrow{v};$$

- Para quaisquer escalares $a, b \in K$ e qualquer vetor $\overrightarrow{u} \in V$,

$$(a+b)\overrightarrow{u} = a\overrightarrow{u} + b\overrightarrow{u};$$

- Para quaisquer escalares $a, b \in K$ e qualquer vetor $\overrightarrow{u} \in V$,

$$(ab)\overrightarrow{u} = a(b\overrightarrow{u});$$

- $1\overrightarrow{u} = \overrightarrow{u}$, para qualquer vetor $\overrightarrow{u} \in V$.

3.10.1 SUBESPAÇOS

Seja W um subconjunto de um espaço vetorial V sobre um corpo K. W é chamado **subespaço de V** se W é um espaço vetorial sobre K em relação às operações de adição de vetores e multiplicação por escalar de V.

Alguns critérios para verificar se W é subespaço de V são citados a seguir [23]:

- W é não-vazio;
- W é fechado para a adição de vetores: , $\overrightarrow{v}, \overrightarrow{w} \in W$ implica $\overrightarrow{v} + \overrightarrow{w} \in W$;
- W é fechado para a multiplicação por escalar: $\overrightarrow{v} \in W$ implica $k\overrightarrow{v} \in W$ para cada $k \in K$;
- $0 \in W$;
- $\overrightarrow{v}, \overrightarrow{w} \in W$ implica $a\overrightarrow{v}$ + $b\overrightarrow{w} \in W$ para todo $a, b \in K$.

3.10.2 COMBINAÇÕES LINEARES

Seja V um espaço vetorial sobre um corpo K e sejam $v_1, ..., v_m \in V$. Qualquer vetor da forma

$$a_1v_1 + a_2v_2 + ... + a_mv_m,$$

onde $a_i \in K$, é chamado uma **combinação linear** de $v_1, ..., v_m$.

3.10.3 SUBESPAÇOS GERADOS

Seja S um subconjunto não-vazio de V. O conjunto $L(S)$ de todas as combinações lineares de vetores em S, é um **subespaço de** V **contendo** S. Além disso, se W é qualquer outro subespaço de V contendo S, então $L(S) \subset W$. Em outras palavras, $L(S)$ é o menor subespaço de V contendo S e é chamado o subespaço gerado por S. Por conveniência define-se $L(\emptyset) = \{\overrightarrow{0}\}$.

A análise de um subespaço vetorial como parte integrante de um espaço vetorial tem uma grande importância na análise de códigos de bloco linear (Capítulo 4).

3.10.4 DEPENDÊNCIA LINEAR

Seja V um espaço vetorial sobre um corpo K. Uma seqüência $v_1, ...v_m$ de vetores é linearmente dependente sobre K se existe uma sequência de escalares $a_1, ..., a_m \in K$, nem todos nulos, tais que,

$$a_1v_1 + a_2v_2 + ... + a_mv_m = 0(*)$$

Caso contrário, isto é, se

$$a_1v_1 + a_2v_2 + ... + a_mv_m = 0,$$

se, e somente se, $a_1 = 0, ..., a_m = 0$, então os vetores são **linearmente independentes**.

Se a relação $(*)$ é válida quando um dos coeficientes a_i não é igual a 0, então os vetores são *linearmente dependentes*. É importante observar também que se um dos vetores $v_1, ..., v_m$, é o vetor nulo 0, por exemplo $v_1 = 0$, então os vetores $v_1, ..., v_m$ são *linearmente dependentes*, pois

$$1v_1 + 0v_2 + ... + 0v_m = 1.0 + 0 + ... + 0 = 0$$

e o coeficiente de v_1 não é igual a 0. Além disso, qualquer vetor não-nulo v é, por si só, *linearmente independente*, pois $kv = 0$, $v \neq 0$ implica $k = 0$.

Observações [23]:

1. Se dois vetores são iguais $(v_1 = v_2)$, então os vetores são dependentes. Pois $v_1 - v_2 + 0v_3 + ... + 0v_m = 0$ e o coeficiente de v_1 não é igual a 0.

2. Dois vetores v_1 e v_2 são dependentes se, e somente se, um deles é múltiplo do outro.

3.10.5 DIMENSÃO DE UM ESPAÇO VETORIAL

Um espaço vetorial V tem dimensão n se, e somente se [15],

- V contém uma seqüência de n vetores linearmente independentes;
- Toda seqüência de $n + 1$ vetores de V é linearmente dependente.

Convenciona-se que o espaço vetorial formado apenas pelo vetor nulo tem dimensão zero.

3.10.6 BASE DE UM ESPAÇO VETORIAL

Seja V um espaço vetorial de dimensão n. Toda seqüência de n vetores linearmente independentes de V chama-se **base** de V.

Uma propriedade fundamental de uma base é que se $(e_1, ..., e_n)$ é uma base do espaço vetorial V de dimensão n, então todo vetor $x \in V$ se exprime de uma única maneira como combinação linear dos vetores $e_1, ..., e_n$.

3.11 ESPAÇO DUAL

Seja S um sub-espaço de dimensão k de um espaço vetorial V. Seja $S^{\perp}$ o conjunto de todos os vetores v_i que pertencem a V e são ortogonais a todos os elementos de S. Nesta situação o sub-espaço vetorial $S^{\perp}$ é denominado como **espaço dual** de S.

Uma propriedade importante que resulta desta definição é que o espaço dual também é um subespaço vetorial de V.

Teorema 3.1: O **Teorema da Dimensão** enuncia uma propriedade importante que relaciona a dimensão de um espaço S com o seu espaço dual $S^{\perp}$. A soma das dimensões destes dois subespaços corresponde à dimensão do espaço vetorial V.

$$dim(S) + dim(S^{\perp}) = dim(V)$$

Este *teorema*, aliado ao conceito de base geradora de um espaço tem uma importância fundamental na construção dos códigos lineares de bloco (ver Capítulo 4).

3.12 TRANSFORMAÇÕES LINEARES

Sejam U e V espaços vetoriais sobre um corpo K. Uma função $T : U \to V$ é uma **transformação linear** se [10]

- $T(u_1 + u_2) = T(u_1) + T(u_2), \forall u_1, u_2 \in U$ e
- $T(\lambda u) = \lambda T(u), \forall \lambda \in K$ e $\forall u \in U$.

3.12.1 O ESPAÇO $L(U, V)$

Para espaços vetoriais U e V sobre K, denota-se por $L(U, V)$ o **conjunto de todas as transformações lineares de** U **a** V. Tal conjunto herda uma estrutura de espaço vetorial sobre K de uma maneira bem natural.

Em primeiro lugar, a função nula de U a V é uma transformação linear. Por outro lado, não é difícil ver que para $F, G \in L(U, V)$e $\lambda \in K$, tem-se que a função $\lambda F + G : U \to V$ dada por $(\lambda F + G)(u) = \lambda F(u) + G(u)$, para cada $u \in U$, é uma transformação linear e, portanto, pertence a $L(U, V)$.

Com isso, mostra-se que $L(U, V)$ é um subespaço vetorial do espaço $\vartheta(U, V)$ de todas as funções de U a V e, em particular, um espaço vetorial sobre K.

3.13 CONCLUSÃO

Neste capítulo foram apresentados alguns fundamentos de estruturas algébricas necessárias ao estudo da teoria dos códigos, como será visto no próximo capítulo.

Capítulo 4

TEORIA DOS CÓDIGOS

4.1 INTRODUÇÃO

O objetivo deste capítulo é mostrar algumas características e conceitos relativos à Teoria dos Códigos, tais como a estrutura de um código linear e algumas formas elementares de detecção e correção de erros.

O capítulo está estruturado da seguinte forma: a seção 4.2 define códigos de blocos e códigos lineares. Na seção 4.3 são apresentados os conceitos de síndrome e detecção de erros e a seção 4.4 mostra o conceito de distância mínima do código. Na seção 4.5 é discutida a capacidade de um código detectar e corrigir erros e na seção 4.6 define-se arranjo padrão. A seção 4.7 mostra como é feita a decodificação por síndrome e a seção 4.8 mostra a probabilidade de um erro indetectável para um código linear num Binary Simetric Channel (BSC).

4.2 CÓDIGOS LINEARES DE BLOCO

4.2.1 CÓDIGOS DE BLOCOS

Em um código, as informações são segmentadas em blocos de mensagens de tamanho fixo. Cada bloco é denotado por u, com k dígitos de informação formada por dígitos binários (0 ou 1). Neste caso, o total de mensagens distintas é igual a 2^k. Um codificador, de acordo com certas regras transforma cada mensagem de entrada em uma n-upla binária v, com $n > k$. Neste caso, v é a **palavra-código** da mensagem u. Se existirem 2^k mensagens, elas corresponderão a 2^k palavras-código. O conjunto de 2^k palavras-código é chamado **código de blocos**. Para que o código seja útil, as palavras-códigos devem ser distintas e a correspondência entre a mensagem u e a palavra-código v deve ser de um para um [4].

4.2.2 CÓDIGO LINEAR

Um código de blocos de tamanho n e 2^k palavras-código é um **código linear ($\mathbf{n}, \mathbf{k}$)**, se e somente se, as 2^k palavras-código formam um sub-espaço k-dimensional do espaço de todas as n-uplas pertencentes a $GF(2)$ [6]. Em outras palavras, um código de blocos binário é linear se a soma módulo-2 de duas palavras-código é também uma palavra-código. A *tabela 4.1* mostra um código de blocos linear $(7, 4)$: $k = 4, n = 7$ [4].

Seja V_n o espaço de todas as n-uplas em $GF(2)$. Se um código linear $C(n, k)$ é um sub-espaço k-dimensional do espaço V_n então é possível encontrar k palavras-código linearmente independentes, g_0, $g_1, g_2, g_3, ..., g_{k-1}$ em C tal que toda palavra-código v em C é uma combinação linear destas k palavras-código. Isto é,

$$v = u_0 g_0 + u_1 g_1 + u_2 g_2 + ... + u_{k-1} g_{k-1} \textbf{ (1)}$$

onde $u_i = 0$ ou 1 para $0 \leq i < k$. Esta é a grande vantagem que esta classe de códigos possui sobre um código genérico qualquer, ou seja, não é preciso armazenar todo o dicionário do código, pois, a

Tabela 4.1: Código Linear de Bloco com $k = 4$ *e* $n = 7$

Mensagens	Palavras-código
(0000)	(0000000)
(1000)	(1101000)
(0100)	(0110100)
(1100)	(1011100)
(0010)	(1110010)
(1010)	(0011010)
(0110)	(1000110)
(1110)	(0101110)
(0001)	(1010001)
(1001)	(0111001)
(0101)	(1100101)
(1101)	(0001101)
(0011)	(0100011)
(1011)	(1001011)
(0111)	(0010111)
(1111)	(1111111)

partir de apenas k palavras-código pode-se obter qualquer outra palavra-código [6].

4.2.3 MATRIZ GERADORA

As k palavras-código linearmente independentes podem ser dispostas como as linhas de uma matriz kxn, como segue:

$$G = \begin{pmatrix} g_0 \\ g_1 \\ ... \\ g_{k-1} \end{pmatrix} = \begin{pmatrix} g_{00} & g_{01} & g_{02} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & g_{12} & \cdots & g_{1,n-1} \\ ... & ... & ... & ... & ... \\ g_{k-1,0} & g_{k-1,1} & g_{k-1,2} & \cdots & g_{k-1,n-1} \end{pmatrix} \textbf{(2)}$$

onde $g_i = (g_{i0}, g_{i1}, ..., g_{i,n-1})$ para $0 \leq i < k$.

Se $u = (u_0, u_1, ..., u_{k-1})$ é a mensagem a ser codificada, a palavra-código correspondente é dada por:

$$
\begin{aligned}
v &= u.G \\
&= (u_0, u_1, ..., u_{k-1}).\begin{pmatrix} g_0 \\ g_1 \\ ... \\ g_{k-1} \end{pmatrix} \quad \textbf{(3)} \\
&= u_0 g_0 + u_1 g_1 + \ldots + u_{k-1} g_{k-1}
\end{aligned}
$$

As linhas de G geram o código linear $C(n, k)$. Por isso, G é chamada de matriz geradora de C. O processo de codificação em um código linear pode ser visto como uma *transformação linear* aplicada ao *vetor informação*. A transformação é descrita por G. Neste caso, as linhas de G são constituídas pelos k vetores-código linearmente independentes de C. De **(3)** conclui-se que um código linear (n, k) é completamente especificado pelas k linhas da matriz geradora G [6]. O codificador tem que armazenar apenas as k linhas de G e formar a combinação linear destas k linhas baseado na mensagem de entrada $u = (u_0, u_1, ..., u_{k-1})$ [7].

Exemplo 4.1: A matriz geradora do código linear $(7, 4)$ dado na *Tabela 4.1* é mostrada a seguir:

$$
G = \begin{pmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}
$$

Se $u = (1101)$ é a mensagem a ser codificada, de acordo com **(3)**, ela corresponde à seguinte palavra-código:

$$
\begin{aligned}
v &= 1.g_0 + 1.g_1 + 0.g_2 + 1.g_3 \\
&= (1101000) + (0110100) + (1010001) \\
&= (0001101)
\end{aligned}
$$

4.2.4 CÓDIGOS SISTEMÁTICOS

Uma propriedade desejável para um bloco linear é que ele possua uma estrutura de palavras-código como mostra a *Figura 4.1*, onde a palavra-código é dividida em duas partes: **a mensagem** e **a redundância** [4].

A **mensagem** consiste de k dígitos de informação inalterados e a **redundância** consiste de $n-k$ dígitos para checagem de paridade, que consistem de somas lineares dos dígitos da informação. Um código linear de bloco com esta estrutura é dito **código linear de bloco sistemático**.

Um código linear sistemático (n,k) é completamente especificado pela matriz G, kxn, da seguinte forma:

$$G = \begin{pmatrix} g_0 \\ g_1 \\ g_2 \\ \dots \\ g_{k-1} \end{pmatrix} = \left(\begin{array}{cccc|ccccc} p_{00} & p_{01} & \dots & p_{0,n-k-1} & 1 & 0 & 0 & \dots & 0 \\ p_{10} & p_{11} & \dots & p_{1,n-k-1} & 0 & 1 & 0 & \dots & 0 \\ p_{20} & p_{21} & \dots & p_{2,n-k-1} & 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ p_{k-1,0} & p_{k-1,1} & \dots & p_{k-1,n-k-1} & 0 & 0 & 0 & \dots & 1 \end{array}\right) \quad \textbf{(4)}$$

Matriz P | Matriz Identidade kxk

onde $p_{ij} = 0$ ou 1. Se a matriz identidade kxk é denotada por I_k, então $G = [PI_k]$. Considerando-se $u = (u_0, u_1, ..., u_{n-1})$ como mensagem a ser codificada, a palavra-código correspondente é:

$$\begin{aligned} v &= (v_0, v_1, v_2, ..., v_{n-1}) \\ &= (u_0, u_1, u_2, u_{k-1}).G \qquad \textbf{(5)} \end{aligned}$$

De **(4)** e **(5)** pode-se concluir que os componentes de v são [4]:

$$v_{n-k+1} = u_i \text{ para } 0 \leq i < k \qquad \textbf{(6a)}$$

$$\text{e } v_j = u_0 p_{0j} + u_1 p_{1j} + ... + u_{k-1} p_{k-1,j} \qquad \text{para } 0 \leq j < n-k. \textbf{ (6b)}$$

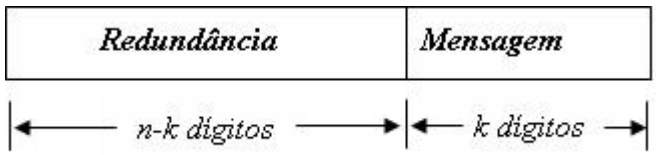

Figura 4.1: Formato de uma palavra-código sistemática

A equação **(6a)** mostra que os k dígitos mais à direita de uma palavra-código v são idênticos aos dígitos da informação $u_0, u_1, ..., u_{k-1}$ a ser codificado e **(6b)** mostra que os $n-k$ dígitos redundantes mais à esquerda são somas lineares dos dígitos da informação. As $n-k$ equações dadas por **(6b)** são chamadas de **equações de verificação de paridade**.

Exemplo 4.2: A matriz G que é geradora do código dado na *Tabela 4.1* (mostrada anteriormente) está na forma sistemática. Tomando-se $u = (u_0, u_1, u_2, u_3)$ como sendo a mensagem a ser codificada e $v = (v_0, v_1, v_2, v_3, v_4, v_5, v_6)$ como sendo a palavra-código correspondente, tem-se:

$$v = (u_0, u_1, u_2, u_3).\begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Pela multiplicação da matriz obtem-se os seguintes dígitos da palavra-código v:

$$v_6 = u_3$$

$$v_5 = u_2$$

$$v_4 = u_1$$

$$v_3 = u_0$$

$$v_2 = u_1 + u_2 + u_3$$

$$v_1 = u_0 + u_1 + u_2$$

$$v_0 = u_0 + u_2 + u_3$$

A palavra-código correspondente para a mensagem (1011) é (1001011).

4.2.5 CÓDIGO DUAL

Para qualquer matriz G_{kxn} com k linhas linearmente independentes existe uma matriz $H_{(n-k)xn}$ com $n-k$ linhas linearmente independentes, tais que qualquer vetor no espaço das linhas de G é ortogonal com as linhas de H. Além disso, qualquer vetor ortogonal com as linhas de H está no espaço das linhas de G. Consequentemente, pode-se descrever o código linear (n,k) gerado por G de forma alternativa, como a seguir [4]:

Uma n-upla v é uma palavra-código gerada por G, se e somente se, $v.H^T = 0$. Esta matriz é chamada de **matriz de verificação de paridade do código**. As 2^{n-k}combinações lineares de linhas da matriz H formam um código linear $C_d(n, n-k)$. Este código é o espaço nulo do código linear C gerado pela matriz G. O código C_d é chamado **código dual**. Portanto, a matriz de verificação de paridade para um código linear C é a matriz geradora para o código dual [7].

Caso a matriz geradora de um código linear (n,k) esteja na forma sistemática de **(4)**, a matriz de verificação de paridade tem a seguinte forma:

$$H = [I_{n-k} P^T]$$

$$= \begin{pmatrix} 1 & 0 & 0 & ... & 0 & p_{00} & p_{01} & ... & p_{0,n-k-1} \\ 0 & 1 & 0 & ... & 0 & p_{10} & p_{11} & ... & p_{1,n-k-1} \\ 0 & 0 & 1 & ... & 0 & p_{20} & p_{21} & ... & p_{2,n-k-1} \\ ... & ... & ... & ... & .. & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & 1 & p_{k-1,0} & p_{k-1,1} & ... & p_{k-1,n-k-1} \end{pmatrix} \quad \textbf{(7)}$$

onde P^T é a transposta da matriz P.

Seja h_j a j-ésima linha de H. Pode-se facilmente verificar que o produto interno da i-ésima linha de G dado por **(4)** e a j-ésima linha de H, dada por **(7)**, é

$$g_i.h_j = p_{ij} + p_{ij} = 0$$

para $0 \leq i < k$ e $0 \leq j < n-k$. Isto implica que $G.H^T = 0$.

4.3 SÍNDROME E DETECÇÃO DE ERROS

4.3.1 VETOR-ERRO

Considere um código linear $C(n,k)$ com matriz geradora G e a matriz de verificação de paridade H. Seja $v = (v_0, v_1, ..., v_{n-1})$ a palavra-código que foi transmitida em um canal com ruído e seja $r = (r_0, r_1, ..., r_{n-1})$ o vetor recebido na saída do canal. Por causa do ruído do canal, r pode ser diferente de v.

O vetor soma

$$e = r + v = (e_0, e_1, ..., e_{n-1}) \qquad \textbf{(8)}$$

é uma n-upla onde $e_i = 1$ para $r_i \neq v_i$ e $e_i = 0$ para $r_i = v_i$. Esta n-upla é chamada de **vetor-erro** ou **padrão de erro**. Os 1's em e são os erros de transmissão causados por ruídos no canal. De **(8)**, conclui-se que o vetor r é o vetor soma da palavra-código transmitida e o vetor-erro. Isto é $r = v + e$.

O destinatário não conhece nem v nem e. Ao receber r, o decodificador deve primeiro verificar se r contém erros de transmissão. Se a presença de erros for detectada, o decodificador deverá tentar localizar os erros e corrigí-los (FEC – Forward Error Correction) ou requisitar a retransmissão de v (ARQ – Automatic Repeat Request). Um dos objetivos da teoria dos códigos é detectar e corrigir tais erros evitando a retransmissão de v.

4.3.2 SÍNDROME

O problema da decodificação está em encontrar o v mais provável de ter sido transmitido, dado o vetor r tenha sido recebido [6]. Quando r é recebido, o decodificador calcula a seguinte $(n-k)$-upla:

$$\begin{aligned} s &= r.H^T \\ &= (s_0, s_1, ..., s_{n-k}) \end{aligned} \qquad \textbf{(9)}$$

que é chamada de síndrome de r [9]. Tem-se que $s = 0$, se e somente se, r é uma palavra-código e $s \neq 0$, se e somente se, r não é uma palavra-código. Quando $s \neq 0$, sabe-se que r não é uma

palavra-código e que a presença de erros foi detectada. Quando $s = 0$, r é uma palavra-código e o destinatário aceita r como sendo a palavra-código transmitida [4].

4.3.3 DETECÇÃO DE ERROS

É possível que alguns erros do vetor-erro não sejam detectados (r contém erros, mas $s = r.H^T = 0$). Isto acontece quando um vetor-erro é idêntico a uma palavra-código diferente de zero. Neste caso, r é a soma de 2 palavras-código que resulta numa terceira palavra-código e, consequentemente,

$$r.H^T = 0$$

Erros deste tipo são chamados de **padrão de erros indetectáveis**. Como há $2^k - 1$ palavras-código diferentes de zero, há $2^k - 1$ padrões de erros indetectáveis. Quando este tipo de erro ocorre, o decodificador provoca um erro de decodificação.

Baseado em **(7)** e **(9)**, os dígitos da síndrome são [4]:

$s_0 = r_0 + r_{n-k}p_{00} + r_{n-k+1}p_{10} + ... + r_{n-1}p_{k-1,0}$

$s_1 = r_1 + r_{n-k}p_{01} + r_{n-k+1}p_{11} + ... + r_{n-1}p_{k-1,1}$

.....

$s_{n-k-1} = r_{n-k-1} + r_{n-k}p_{0,n-k-1} + r_{n-k+1}p_{1,n-k-1} + ... + r_{n-1}p_{k-1,n-k-1}$

Examinando-se as equações acima, conclui-se que a síndrome s é simplesmente o vetor soma dos dígitos recebidos $(r_0, r_1, ..., r_{n-k-1})$ e os dígitos de paridade calculados a partir dos dígitos de informação recebidos $r_{n-k}, r_{n-k+1}, ..., r_{n-1}$. Pode-se dizer também que os dígitos da síndrome são descritos exatamente pelas mesmas equações que definem as checagens de paridade [6].

Exemplo 4.3: Considerando-se novamente o código linear $(7, 4)$ cuja matriz de verificação de paridade já foi mostrada, a síndrome para o vetor recebido $r = (r_0, r_1, r_2, r_3, r_4, r_5, r_6)$ é dada por:

$$s = (s_0, s_1, s_2)$$

$$= (r_0, r_1, r_2, r_3, r_4, r_5, r_6).\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

Os dígitos da síndrome são:

$$s_0 = r_0 + r_3 + r_5 + r_6$$
$$s_1 = r_1 + r_3 + r_4 + r_5$$
$$s_2 = r_2 + r_4 + r_5 + r_6$$

A síndrome s computada do vetor r recebido depende apenas do padrão de erro e, e não da palavra-código transmitida v. Desde que r seja o vetor soma de v e e, conclui-se de **(9)** que

$$s = r.H^T = (v + e).H^T = v.H^T + e.H^T$$

Como $v.H^T = 0$, obtem-se a seguinte relação entre a síndrome e o erro padrão [4]:

$$s = e.H^T \qquad \textbf{(10)}$$

Se a matriz de verificação de paridade H é expressada na forma sistemática como dado em **(7)**, multiplicando-se $e.H^T$produz-se os seguintes relacionamentos lineares entre os dígitos da síndrome

e os dígitos do erro:

$s_0 = e_0 + e_{n-k}p_{00} + e_{n-k+1}p_{10} + ... + e_{n-1}p_{k-1,0}$

$s_1 = e_1 + e_{n-k}p_{01} + e_{n-k+1}p_{11} + ... + e_{n-1}p_{k-1,1}$

.....

$s_{n-k-1} = e_{n-k-1} + e_{n-k}p_{0,n-k-1} + e_{n-k+1}p_{1,n-k-1} + ... + e_{n-1}p_{k-1,n-k-1}$ **(11)**

Os dígitos da síndrome são combinações lineares simples dos dígitos do erro. Eles fornecem informações sobre os dígitos do erro e por isso podem ser usados para a correção de erros. Existem 2^k padrões de erro que resultam a mesma síndrome, e o verdadeiro erro e é apenas um deles. Portanto, o decodificador tem que determinar o verdadeiro erro de um conjunto de 2^k candidatos. Para minimizar a probabilidade de um erro de decodificação, o mais provável erro que satisfaz as equações de **(11**$)$ é escolhido como o erro verdadeiro. Se o canal for um BSC (Binary Simetric Channel), o padrão de erro mais provável é o que tem o menor número de dígitos não-zero [4].

Exemplo 4.4: Considerando-se o código linear $(7, 4)$, seja $v = (1001011)$ a palavra-código transmitida e $r = (1001001)$ o vetor recebido. Após receber r, o destinatário calcula a síndrome:

$$s = r.H^T = (111)$$

Em seguida, o destinatário tenta determinar o verdadeiro erro $e = (e_0, e_1, e_2, e_3, e_4, e_5, e_6)$, usando a síndrome acima. De **(10)** e **(11)** conclui-se que o dígitos do erro estão relacionados com os dígitos da síndrome pelas seguintes equações lineares:

$$1 = e_0 + e_3 + e_5 + e_6$$

$$1 = e_1 + e_3 + e_4 + e_5$$

$$1 = e_2 + e_4 + e_5 + e_6$$

Existem $2^4 = 16$ erros que satisfazem, as equações anteriores. São eles:

$(0000010), (1010011), (1101010), (0111011), (0110110), (1100111), (1011110), (0001111)$
$, (1110000), (0100001), (0011000), (1001001), (1000100), (0010101), (0101100), (1111101)$

O vetor de erro $e = (0000010)$ tem o menor número de componentes não-zero. Se o canal é um BSC, então $e = (0000010)$ é o erro mais provável que satisfaz a equação anterior. Tomando-se $e = (0000010)$ como o verdadeiro erro, o destinatário decodifica o vetor recebido $r = (1001001)$ na seguinte palavra-código:

$v^* = r + e$

$= (1001001) + (0000010)$

$= (1001011)$

v^* é a verdadeira palavra-código transmitida. Consequentemente, o destinatário fará uma decodificação correta.

4.4 DISTÂNCIA MÍNIMA DE UM CÓDIGO DE BLOCO

A **distância mínima** de um código de bloco é um parâmetro que determina a capacidade de um código detectar um erro aleatório e a capacidade de corrigir um erro aleatório.

Seja $v = (v_0, v_1, ..., v_{n-1})$ uma n-upla binária. O peso de v (Hamming Weight), denotado por $w(v)$, é definido como o número de componentes diferentes de zero em v [4]. Por exemplo, o peso de $v = (1001011)$ é 4.

Sejam v e w duas n-uplas. A **distância de Hamming** entre v e w, denotada por $d(v, w)$, é definida como o número de dígitos em que eles se diferem [6]. Por exemplo, a distância de Hamming entre $v = (1001011)$ e $w = (0100011)$ é 3.

A distância de Hamming é a métrica que satisfaz a desigualdade triangular abaixo [7]. Seja v, w

e x três n-uplas. Então

$$d(v,w) + d(w,x) \geq d(v,x) \qquad \textbf{(12)}$$

A distância de Hamming entre duas n-uplas, v e w, é igual ao peso da soma de v e w, isto é,

$$d(v,w) = w(v+w) \qquad \textbf{(13)}$$

Por exemplo, a distância de Hamming entre $v = (1001011)$ e $w = (1110010)$ é igual a 4 e o peso de $v + w = (0111001)$ também é igual a 4.

Dado um código de bloco C, pode-se calcular a distância de Hamming entre duas palavras-código quaisquer. A distância mínima de C, denotada por d_{min} é definida como

$$d_{min} = min\{d(v,w) : v, w \in C, v \neq w\} \qquad \textbf{(14)}$$

Se C é um código linear de bloco, a soma de dois vetores também é um vetor. Conclui-se de **(13)** que a distância de Hamming entre dois vetores-código em C é igual ao peso de um terceiro vetor-código em C [4]. Então, de **(14)** conclui-se que

$$\begin{aligned} d_{min} &= min\{d(v,w) : v, w \in C, v \neq w\} \\ &= min\{w(x) : x \in C, x \neq 0\} \qquad \textbf{(15)} \\ &= w_{min} \text{ (Peso mínimo do código linear } C) \end{aligned}$$

Tais fatos demonstram o teorema a seguir.

Teorema 4.1: A distância mínima de um código linear de bloco é igual ao peso mínimo das palavras-código diferentes de zero [9].

Teorema 4.2: Seja $C(n,k)$ um código linear com a matriz de verificação de paridade H. Para cada vetor-código com peso L, existem l colunas de H, tais que o vetor soma destas colunas é igual ao vetor-zero. Inversamente, se existem l colunas de H tais que o vetor soma é o vetor-zero, existe um vetor-código de peso l em C [4].

Demonstração: Escrevendo-se a matriz de verificação de paridade da seguinte forma:

$$H = [h_0, h_1, ..., h_{n-1}]$$

onde h_i representa a i-ésima coluna de H. Seja $v = (v_0, v_1, ..., v_{n-1})$ um vetor-código de peso L. Isto é, v possui l componentes diferentes de zero. Sejam $v_{i1}, v_{i2}, ..., v_{il}$ os l componentes diferentes de zero de v, onde $0 \leq i_1 < i_2 < ... < i_l \leq n-1$. Então $v_{i1} = v_{i2} = ... = v_{il} = 1$. Se v é um vetor-código, devemos ter

$$\begin{aligned} 0 &= v.H^T \\ &= v_0h_0 + v_1h_1 + ... + v_{n-1}h_{n-1} \\ &= v_{i1}h_{i1} + v_{i2}h_{i2} + ... + v_{il}h_{il} \\ &= h_{i1} + h_{i2} + ... + h_{il} \end{aligned}$$

Isto prova a primeira parte do $Teorema$.

Agora suponha que $h_{i1} + h_{i2} + ... + h_{il}$ sejam l colunas de H tais que

$$h_{i1} + h_{i2} + ... + h_{il} = 0 \textbf{ (a)}$$

Seja a n-upla binária $x = (x_1, x_2, ..., x_{n-1})$ cujos componentes diferentes de zero são $x_{i1}, x_{i2}, ...x_{il}$. O peso de Hamming de x é l. Considere o produto

$$\begin{aligned} x.H^T &= x_0h_0 + x_1h_1 + ... + x_{n-1}hl \\ &= x_{i1}h_{i1} + x_{i2}h_{i2} + ... + x_{il}h_{il} \\ &= h_{i1} + h_{i2} + ... + h_{il} \end{aligned}$$

Segue de **(a)** que $x.H^T = 0$. Então x é um vetor-código de peso l em C. Isto prova a segunda parte do teorema.

C.Q.D

Colorário 4.1: Seja C um código linear com matriz de verificação de paridade H. Se nenhuma das $d-1$ ou menos colunas de Hsomam 0, o código tem peso mínimo de pelo menos d.

Colorário 4.2: Seja C um código linear de bloco com matriz de verificação de paridade H. O peso mínimo (ou distância mínima) de C é igual ao menor número de colunas de H que somam 0.

4.5 CAPACIDADE DE UM CÓDIGO DE BLOCOS DETECTAR E CORRIGIR ERROS

Quando um vetor-código v é transmitido num canal com ruído, um erro resultará em um vetor recebido r diferente do vetor transmitido v em l lugares (isto é, $d(v, r) = l$). Se a distância mínima de um código de bloco C é d_{min}, quaisquer dois vetores-código distintos diferem em no mínimo d_{min} lugares. Para este código C, nenhum padrão de erro com $d_{min} - 1$ ou menos erros pode mudar um vetor-código em outro. Portanto, um padrão de erro com $d_{min} - 1$ ou menos erros resultará no recebimento de um vetor r que não é uma palavra-código em C.

Quando o destinatário detecta que o vetor recebido não é uma palavra-código, diz-se que os erros são detectáveis. Por isso, um código com distância mínima d_{min} é capaz de detectar todos os padrões de erros com d_{min} ou menos erros. Contudo, não se pode detectar todos os padrões de erros com d_{min} erros. Isto ocorre porque existe pelo menos um par de vetores-código w_1, w_2 que diferem em d_{min} lugares e existe um padrão de erro com d_{min} erros tal que $w_1 = w_2 + d_{min}$.

O mesmo argumento é aplicado para padrões de erros maiores que d_{min} erros. Por esta razão, diz-se que a capacidade de detectar um erro aleatório de um código de bloco com distância mínima d_{min} é $d_{min} - 1$.

Mesmo que um código de bloco com distância mínima d_{min} garanta a detecção de todos os padrões de erros com $d_{min} - 1$ ou menos erros, ele também é capaz de detectar uma larga fração de padrões de erros com d_{min} ou mais erros. Um código linear (n, k) é capaz de detectar $2^n - 2^k$ padrões de erro com tamanho n. Isto pode ser mostrado da seguinte forma [4]: Entre os $2^n - 1$ possíveis padrões de erro diferentes de zero, há $2^k - 1$ padrões de erro que são idênticos a $2^k - 1$ palavras-código diferentes de zero. Se qualquer um destes $2^k - 1$ padrões de erro ocorrer, modificará a palavra-código transmitida v em outra palavra-código w. Então, w será recebido e a síndrome será igual a zero. Neste caso, o decodificador aceita w como a palavra-código transmitida e então comete uma decodificação incorreta. Por isso, existem $2^k - 1$ padrões de erro indetectáveis. Se um padrão

de erro não é igual a uma palavra-código diferente de zero, o vetor r recebido não será uma palavra-código e a síndrome não será zero. Neste caso, o erro será detectado. Existem exatamente $2^n - 2^k$ padrões de erro que não são iguais às palavras-código de um código linear (n, k). Estes $2^n - 2^k$ padrões de erro são detectáveis. Para n grande, $2^k - 1$ é, em geral, muito menor que 2^n. Portanto, apenas uma pequena fração de padrões de erro passará pelo decodificador sem ser detectado.

Seja $C(n, k)$ um código linear. Seja A_i o número de vetores-código de peso i em C. Os números $A_0, A_1, ..., A_k$ são chamados de distribuição de pesos em C [9]. Se C é usado apenas para detecção de erros em um BSC (Binary Simetric Channel), a probabilidade do decodificador falhar na detecção da presença de erros pode ser calculada pela distribuição de pesos de C. Seja $P_u(E)$ a probabilidade de um erro indetectável. Desde que um erro indetectável ocorra apenas quando o padrão de erro é idêntico a um vetor-código diferente de zero em C,

$$P_u(E) = \sum_{i=1}^{n} A_i p^i (1-p)^{n-i} \quad \textbf{(16)}$$

onde p é a probabilidade de transição do BSC. Se a distância mínima de C é d_{min} então A_1 até $A_{d_{min}-1}$ são zeros [4].

Exemplo 4.5: Considerando-se o código $(7, 4)$ dado na *Tabela 4.1*. A distribuição de peso deste código é $A_0 = 1, A_1 = A_2 = 0, A_3 = 7, A_4 = 7, A_5 = A_6 = 0$ e $A_7 = 1$. A probabilidade de um erro indetectável é

$$P_u(E) = 7p^3(1-p)^4 + 7p^4(1-p) + 3 + p^7$$

Se um código de bloco C com distância mínima d_{min} é usado para a correção de erros aleatórios, seria desejável saber quantos erros o código está apto a corrigir. A distância mínima d_{min} é par ou ímpar. Seja t um inteiro positivo tal que

$$2t + 1 \leq d_{min} \leq 2t + 2 \quad \textbf{(17)}$$

Um código C é capaz de corrigir todos os erros de peso menor ou igual a t, como mostrado a seguir [4]: Sejam v e r o vetor-código transmitido e o vetor recebido, respectivamente. Seja w um vetor-código em C. A distância de Hamming entre v,r e w satisfaz a desigualdade triangular:

$$d(v,r) + d(w,r) \geq d(v,w) \qquad \textbf{(18)}$$

Supondo-se que um erro de peso t' ocorra durante a transmissão de v. Então o vetor recebido r difere de v em t' lugares e consequentemente $d(v,r) = t'$. Se v e w são vetores-código em C, tem-se

$$d(v,w) \geq d_{min} \geq 2t + 1 \qquad \textbf{(19)}$$

Combinando-se **(18)** e **(19)** e usando o fato de que $d(v,r) = t'$, obtem-se a desigualdade:

$$d(w,r) \geq 2t + 1 - t'$$

Se $t' \leq t$, então

$$d(w,r) > t$$

A desigualdade anterior mostra que se um padrão de erro com t ou menos erros ocorrem, o vetor recebido r está mais próximo (na distância de Hamming) do vetor-código v transmitido do que qualquer outro vetor-código w em C. Para um BSC isto significa que a probabilidade condicional $P(r|v)$ é maior que a probabilidade condicional $P(r|w)$ para $w \neq v$. Baseado na probabilidade máxima do esquema de decodificação, r é decodificado em v, que é o verdadeiro vetor-código transmitido. Isto resulta em uma decodificação correta e então os erros são corrigidos. Por outro lado, o código não é capaz de corrigir todos os padrões de erros com peso l onde $l > t$. Neste último caso um padrão de erro de peso l resulta em um vetor recebido que está mais próximo de um vetor-código incorreto. Isto será mostrado tomando-se v e w, dois vetores-código em C tais que

$$d(v,w) = d_{min}$$

Seja e_1 e e_2 dois padrões de erro que satisfazem as seguintes condições:

(i) $\quad e_1 + e_2 = v + w$

(ii) $\quad e_1$ e e_2 não possuem componentes diferentes de zero no mesmo lugar

Tem-se então

$$w(e_1) + w(e_2) = w(v + w) = d(v, w) = d_{min} \qquad \textbf{(20)}$$

Supondo-se que v é transmitido e é corrompido por um padrão de erro e_1, então o vetor recebido é

$$r = v + e_1$$

A distância de Hamming entre w e r é

$$d(w, r) = w(w + r) = w(w + v + e_1) = w(e_2) \qquad \textbf{(21)}$$

Suponha que o padrão de erro e_1 contenha mais que t erros (isto é, $w(e_1) > t$). Se $2t + 1 \leq d_{min} \leq 2t + 2$, segue de **(20)** que $w(e_2) \leq t + 1$. Combinado-se **(20)** e **(21)** e usando-se o fato de que $w(e_1) > t$ e $w(e_2) \leq t + 1$, obtem-se

$$d(v, r) = w(r, v) = w(e_1) \geq w(e_2) = d(w, r)$$

Conclui-se que

$$d(v, w) \geq d(w, r)$$

Esta desigualdade mostra que existe um erro de peso l $(l > t)$ que resulta em um vetor recebido que está próximo de um vetor-código incorreto w do que do vetor-código transmitido v. Baseado na probabilidade máxima do esquema de decodificação, uma decodificação incorreta será feita.

Baseando-se nos resultados anteriores, pode-se dizer que um código de bloco com distância mínima d_{min} garante a correção de todos os erros e_1 onde $w(e_1) = t$ e

$$t \leq \lfloor (d_{min} - 1)/2 \rfloor$$

Neste caso, $\lfloor (d_{min}-1)/2 \rfloor$ denota o maior inteiro menor ou igual a $(d_{min}-1)/2$. O parâmetro $t = \lfloor (d_{min}-1)/2 \rfloor$ é chamado de **capacidade de correção de erros aleatórios do código**. O código é conhecido como um **código corretor de t erros** [4].

Um código de bloco com capacidade de correção de t erros aleatórios é normalmente capaz de corrigir muitos padrões de erros com peso maior ou igual a $t+1$. Para um código linear (n,k) corretor de t erros, ele é capaz de corrigir um total de 2^{n-k} padrões de erros, incluindo aqueles com peso menor ou igual a t . Se um código de bloco corretor de t erros é usado estritamente para a correção num BSC com probabilidade de transição p, a probabilidade de que o decodificador faça uma decodificação errônea é limitada superiormente por

$$P(E) \leq \sum_{i=1}^{n} \binom{n}{i} p^i (1-p)^n - 1 \qquad (22)$$

Na prática, um código é frequentemente usado para corrigir λ ou menos erros e detectar l $(l > \lambda)$ ou menos erros simultaneamente. Ou seja, o código é capaz de corrigir uma quantidade de erros menor ou igual a λ. Quando a quantidade de erros é maior que λ e menor que $l+1$, o código é capaz de detectar a presença destes erros e não provocar uma decodificação errada.

Para esta proposta, a distância mínima (d_{min}) do código é no mínimo $\lambda + l + 1$. Então, um código de bloco com $d_{min} = 10$, é capaz de corrigir três erros ou menos e detectar seis erros ou menos simultaneamente.

Pela discussão anterior pode-se concluir que a capacidade de detecção de erros aleatórios e correção dos mesmos em um código é determinada pela distância mínima do código [4].

4.6 ARRANJO PADRÃO

Esta seção representa a definição de arranjo padrão que é o fundamento da proposta apresentada no próximo capítulo.

Considere um código linear $C(n,k)$, e $v_0, v_1, v_2, v_3, ..., v_{2^k}$, vetores-código de C, linearmente

independentes. Supondo-se que um vetor-código é transmitido sobre um canal com ruído, o vetor recebido r é uma das 2^n n-uplas em $GF(2^n)$. Um esquema de decodificação usado pelo destinatário pode ser uma regra para partição dos 2^n possíveis vetores recebidos em 2^k conjuntos disjuntos $D_1, D_2, ...D_{2^k}$, tais que o vetor-código v_i está contido em D_i para $1 \leq i \leq 2^k$. Então, cada subconjunto D_i tem correspondência de um-para-um com um vetor-código v_i. Se o vetor recebido r é encontrado no subconjunto D_i, r é decodificado como v_i. A decodificação correta é feita se, e somente se, o vetor recebido r está em um subconjunto D_i que corresponde ao verdadeiro vetor-código transmitido.

Um método de partição de 2^n possíveis vetores recebidos em 2^k conjuntos disjuntos tais que cada subconjunto contenha um e apenas um vetor-código será descrita a seguir [4]: A partição é baseada na estrutura linear do código. Primeiro, os 2^k vetores do código de C são colocados em uma linha com o vetor-zero do código $v_1 = (0, 0, ..., 0)$ como o primeiro elemento. Para as demais $2^n - 2^k$ n-uplas, uma n-upla e_2 é escolhida e colocada abaixo do vetor zero v_1. A segunda linha é formada pela adição de e_2 a cada vetor-código v_i da primeira linha e a soma $e_2 + v_i$ é colocada embaixo de v_i. Tendo completada a segunda linha, uma n-upla e_3 tal que e_3 ainda não está no arranjo é escolhida e são calculadas as n-uplas $e_3 + v_i$ que são dispostas respectivamente nas colunas $i = 1, 2, ..., 2^k$. O processo continua até todas as n-uplas serem usadas. Ao final, haverá um arranjo de linhas e colunas como mostra a *Tabela 4.2*. Este arranjo é chamado arranjo padrão de um dado código linear C.

Como resultado da regra de construção do arranjo padrão é que a soma de quaisquer dois vetores na mesma linha é um vetor-código de C. Segue algumas importantes propriedades de um arranjo padrão.

Teorema 4.3: [4] Não existem duas n-uplas idênticas na mesma linha de um arranjo padrão. Toda n-upla aparece em uma e apenas uma linha.

Demonstração: A primeira parte do *Teorema* segue do fato de que todos os vetores-código de C são distintos. Suponha que duas n-uplas na l-ésima linha sejam idênticas, ou seja, $e_l + v_i = e_l + v_j$ com $i \neq j$. Isto significa que $v_i = v_j$, o que é impossível. Portanto, não existem duas n-uplas

Tabela 4.2: Arranjo Padrão para um código linear (n, k)

$v_1 = 0$	v_2	...	v_i	...	v_{2^k}
e_2	$e_2 + v_2$	...	$e_2 + v_i$	...	$e_2 + v_{2^k}$
e_3	$e_3 + v_2$	...	$e_3 + v_i$	...	$e_3 + v_{2^k}$
e_i	$e_i + v_2$	...	$e_i + v_i$	...	$e_i + v_{2^k}$
...	...	...	...	...	...
$e_{2^{n-k}}$	$e_{2^{n-k}} + v_2$	...	$e_{2^{n-k}} + v_i$	...	$e_{2^{n-k}} + v_{2^k}$

idênticas na mesma linha.

Segue da regra de construção do arranjo padrão que toda n-upla aparece pelo menos uma vez. Agora suponha que uma n-upla apareça na l-ésima e na m-ésima linhas com $l < m$. Então esta n-upla deve ser igual a $e_l + v_i$ para um mesmo i e igual a $e_m + v_j$ para um mesmo j. Como resultado, $e_l + v_i = e_m + v_j$. Desta igualdade obtemos $e_m = e_l + (v_i + v_j)$. Já que v_i e v_j são vetores-código em C, $v_i + v_j$ é também um vetor-código em C, chamado v_s. Então, $e_m = e_l + v_s$. Isto implica que a n-upla em está na l-ésima linha do arranjo, o que contradiz a sua regra de construção que diz que em, o primeiro elemento da m-ésima linha não deve ser usado na linha anterior. Portanto, nenhuma n-upla pode aparecer em mais de uma linha do arranjo. Isto conclui a demonstração da segunda parte do teorema.

C.Q.D

Do *Teorema 4.3* conclui-se que existem $\frac{2^n}{2^k} = 2^{n-k}$ linhas disjuntas no arranjo padrão e que cada linha consiste de 2^k elementos distintos. As 2^{n-k} linhas são chamadas **classes laterais** do código C e a primeira n-upla e_j de cada classe lateral é chamada de **representante da classe lateral**.Qualquer elemento em uma classe lateral pode ser usado como seu representante. Isto não muda os elementos da classe lateral, simplesmente permuta-os.

Exemplo 4.5: Um exemplo do que foi dito anteriormente é mostrado a seguir para um código linear (6,3) gerado pela matriz:

$$G = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

O arranjo padrão deste código é mostrado na *Tabela 4.3*.

Um arranjo padrão de um código linear $C(n,k)$ consiste de 2^k colunas disjuntas. Cada coluna consiste de 2^{n-k} n-uplas com a primeira sendo um vetor-código de C. Denotando-se a j-ésima coluna do arranjo padrão por D_j, então

$$D_j = \{v_j, e_2 + v_j, e_3 + v_j, ..., e_{2^{n-k}} + v_j\}, \qquad \textbf{(23)}$$

onde v_j é um vetor-código C e $e_2, e_3, ..., e_{2^{n-k}}$, são os representantes das classes laterais. As 2^k colunas disjuntas $D_1, D_2, ..., D_{2^k}$, podem ser usadas para decodificar o código C. Supondo-se que o vetor do código v_j seja transmitido sobre um canal com ruído, de **(23)** conclui-se que o vetor recebido r está em D_j se um erro causado pelo canal é um representante da classe lateral. Neste caso, o vetor recebido r será decodificado corretamente no vetor-código transmitido v_j. Por outro lado, se um erro causado por um canal não é o representante da classe lateral, a decodificação resultante será errada. Isto pode ser visto como segue [4]: Considere um erro x causado pelo canal que esta em uma classe lateral e sob algum vetor-código diferente de zero. Suponha que $x = e_l + v_i$. Logo, o vetor recebido é

$$r = v_j + x = e_l + (v_i + v_j) = e_l + v_s.$$

Neste caso, vetor recebido r está em D_s e é decodificado como sendo v_s, que não é o vetor-código transmitido. Isto resulta numa decodificação errada. Deste modo, a decodificação é correta se, e somente se, o erro causado pelo canal é o representante da classe lateral. Por esta razão, os 2^{n-k} representantes da classe lateral (incluindo o vetor zero) são chamados erros corrigíveis. Resumindo-se os resultados acima, tem-se o seguinte teorema:

Teorema 4.4: Todo código linear (n,k) é capaz de corrigir 2^{n-k} erros.

Tabela 4.3: Arranjo padrão de um código $(6, 3)$

Representante da Classe Lateral							
000000	001110	010101	100011	011101	101101	110110	111000
000001	001111	010100	100010	011010	101100	110111	111001
000010	001100	010111	100001	011001	101111	110100	111010
000100	001010	010001	100111	011111	101001	110010	111100
001000	000110	011101	101011	010011	100101	111110	110000
010000	011110	000101	110011	001011	111101	100110	101000
100000	101110	110101	000011	111011	001101	010110	011000
001001	000111	011100	101010	010010	100100	111111	110001

Para minimizar a probabilidade de um erro de decodificação, o erro mais provável de ocorrer para um dado canal deve ser escolhido como representante da classe lateral. Para um BSC, um erro de menor peso é mais provável que um erro de maior peso. Assim, um arranjo padrão é formado de forma que cada representante de classe lateral seja escolhido como sendo um vetor de peso mínimo em relação aos vetores restantes disponíveis [6]. Escolhendo os representantes das classes laterais desta maneira, cada representante tem peso mínimo nesta classe lateral. Como resultado, a decodificação baseada no arranjo padrão é a distância mínima decodificada (isto é, probabilidade máxima de decodificação).

É possível mostrar isto tomando-se r como o vetor recebido e supondo-se que r é encontrado na ith coluna D_i e lth classe lateral do arranjo padrão. Então r é decodificado no vetor-código v_i. Desde que $r = e_l + v_i$, a distância entre r e v_i é [4]

$$d(r, v_i) = w(r + v_i) = w(e_l + v_i + v_i) = w(e_l) \qquad \textbf{(24)}$$

Considerando-se a distância entre r e um outro vetor-código v_j,

$$d(r, v_j) = w(r + v_j) = w(e_l + v_i + v_j)$$

Já que v_i e v_j são dois diferentes vetores-código, seu vetor-soma v_i+v_jé um vetor-código diferente de zero. Considere que $v_s = v_i + v_j$. Logo,

$$d(r, v_j) = w(e_l + v_s) \qquad \textbf{(25)}$$

Já que e_l e $e_l + v_s$ estão na mesma classe lateral, então $w(e_l) \leq w(e_l + v_s)$. Segue de **(24)** e **(25)** que

$$d(r, v_i) \leq d(r, v_i)$$

Isto significa que o vetor recebido é decodificado em um vetor-código com menor distância de Hamming. Consequentemente, se cada representante da classe lateral é escolhido como tendo peso mínimo nas suas classes laterais, a decodificação baseada no arranjo padrão é a decodificação da distância mínima.

Seja α_i o número de representantes da classe lateral de peso i. Os números $\alpha_0, \alpha_1, ..., \alpha_n$ são chamados de **distribuição de pesos dos representantes das classes laterais**. Conhecendo-se estes números, pode-se calcular a probabilidade de um erro de decodificação. Como um erro de decodificação ocorre se, e somente se, o erro não é o representante da classe lateral, então a probabilidade de erro indetectável $P(E)$ é dada por: [4]

$P(E) \leq \sum_{i=1}^{n} \alpha_i p^i (1-p)^n - i$ **(26)**

onde p é a probabilidade de transição.

Um código linear (n, k) é capaz de detectar $2^n - 2^k$ erros. Porém, é capaz de corrigir apenas 2^{n-k} erros. Para um n grande, 2^{n-k} erros é uma fração muito pequena de $2^n - 2^k$. Portanto, a probabilidade de um erro de decodificação é mais alta que a probabilidade de um erro não ser detectado.

Teorema 4.5: [4] Para um código linear $C(n, k)$ com distância mínima d_{min}, todas as n-uplas de peso menor ou igual a $t = \lfloor (d_{min} - 1)/2 \rfloor$ podem ser usadas como representantes da classe lateral de um arranjo padrão. Se todas as n-uplas de peso menor ou igual a t são usadas como representantes da classe lateral, há pelo menos uma n-upla de peso $t + 1$ que não pode ser usada como representante da classe lateral.

Este *Teorema* reconfirma o fato de que um código linear (n, k) com distância mínima d_{min} é capaz de corrigir todos os erros menores ou iguais a $\lfloor (d_{min} - 1)/2 \rfloor$, mas não é capaz de corrigir todos os

erros de peso $t + 1$.

Um arranjo padrão tem uma importante propriedade que pode ser usada para simplificar o processo de decodificação. Seja H a matriz de verificação de paridade de um código linear $C(n, k)$:

Teorema 4.6: [4] Todas as 2^k n-uplas de uma classe lateral têm a mesma síndrome. As síndromes para diferentes classes laterais são diferentes.

Demonstração: Considerando-se uma classe lateral cujo representante é e_l. Um vetor nesta classe lateral é a soma de e_l e algum vetor-código v_i em C. A síndrome de um vetor na classe lateral é dada por:

$$(e_l + v_i).H^T = e_l.H^T + v_i.H^T = e_l.H^T \qquad \text{(já que } v_i.H^T = 0\text{)}$$

A igualdade anterior diz que a síndrome de um vetor em uma classe lateral é igual à síndrome do representante da classe lateral. Portanto, todos os vetores de uma classe lateral têm a mesma síndrome.

Seja e_j e e_l representantes da j-ésima e l-ésima classes laterais, respectivamente, onde $j < l$. Supondo-se que as síndromes destas 2 classes laterais sejam iguais. Então

$$e_j.H^T = e_l.H^T,$$
$$(e_j + e_l).H^T = 0.$$

Isto implica que $e_j + e_l$ é um vetor-código em C. Considere $e_j + e_l = v_i$, logo $e_l = e_j + v_i$. Isto implica que e_l está na j-ésima classe lateral, o que contradiz a regra de construção do arranjo padrão, que diz que um representante da classe lateral não deve ser usado previamente. Logo, não existem 2 classes laterais com a mesma síndrome.

C.Q.D

4.7 DECODIFICAÇÃO POR SÍNDROME

A síndrome de uma n-upla é uma $(n - k)$-upla. Como há 2^{n-k} distintas $(n - k)$-uplas, segue do *Teorema 6* que existe uma correspondência entre um representante da classe lateral (um erro

corrigível) e uma síndrome. Usando-se esta correspondência de um-para-um, pode-se formar uma tabela de decodificação, que é mais simples de usar do que o arranjo padrão. A tabela consiste de 2^{n-k} representantes da classe lateral (os erros corrigíveis) e suas correspondentes síndromes. Esta tabela é armazenada ou "ligada" ao destinatário [4]. Este fato será utilizado na proposta de um sistema seguro e senhas. A decodificação do vetor recebido consiste de três passos [7]:

Passo 1: Calcular a síndrome de r, dada por $r.H^T$.

Passo 2: Determinar o representante da classe lateral e_l, cuja síndrome é igual a $r.H^T$. Então e_l é escolhido para ser o erro causado pelo canal.

Passo 3: Decodificar o vetor recebido r no vetor-código $v = r + e_l$.

O esquema de decodificação descrito anteriormente é chamado de **decodificação por síndrome**. Em princípio, a decodificação por síndrome pode ser aplicada para qualquer código linear (n, k). Ela resulta num tempo de decodificação mínimo e numa probabilidade de erro mínima também. Contudo, para um $n-k$ grande, a implementação deste esquema de decodificação torna-se pouco prático, e um grande armazenamento ou circuito lógico complicado é necessário.

Exemplo 4.6: Considerando-se o código linear $(7, 4)$ dado na *Tabela 4.1*, a matriz de verificação de paridade é

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

O código tem $2^3 = 8$ classes laterais e, consequentemente, há 8 erros corrigíveis (incluindo o vetor zero). Já que a distância mínima do código é 3, ele é capaz de corrigir todos os erros de peso 1 ou 0. Logo, todas as 7-uplas de peso 1 ou 0 podem ser usadas como representantes da classe lateral. Há $\binom{7}{0} + \binom{7}{0} = 8$ vetores de peso 1 ou 0. Para o código linear $(7, 4)$, considerado neste exemplo, o número de erros corrigíveis garantidos pela distância mínima é igual ao número total de erros corrigíveis e suas correspondentes síndromes são dadas na *Tabela 4.4*.

Tabela 4.4: Tabela de decodificação para o código linear $(7, 4)$

Síndrome	Representantes das Classes Laterais
(100)	(1000000)
(010)	(0100000)
(001)	(0010000)
(110)	(0001000)
(011)	(0000100)
(111)	(0000010)
(101)	(0000001)

Suponha que o vetor do código $v = (1001011)$ é transmitido e $r = (1001111)$ é recebido. Para se decodificar r, calcula-se a síndrome de r,

$$s = (1001111).\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} = (011)$$

Pela *Tabela 4.4*, deduz-se que (011) é a síndrome do representante da classe lateral $e = (0000100)$. Então, (0000100) é escolhido como sendo o erro causado pelo canal e r é decodificado em

$$\begin{aligned} v^* &= r + e \\ &= (1001111) + (0000100) \\ &= (1001011) \end{aligned}$$

que é o verdadeiro vetor-código transmitido. A decodificação é correta já que o erro causado pelo canal é um representante da classe lateral.

Considere agora que $v = (0000000)$ é transmitido e $r = (1000100)$ é recebido. Observe que dois erros ocorreram durante a transmissão de v. O erro não é corrigível e causará um erro de decodificação. Quando r é recebido, o destinatário calcula a síndrome

$$s = r.H^T = (111)$$

Pela tabela de decodificação, o representante da classe lateral $e = (0000010)$ corresponde a síndrome $s = (111)$. Como resultado, r é decodificado no vetor-código

$$\begin{aligned} v^* &= r + e \\ &= (1000100) + (0000010) \\ &= (1000110) \end{aligned}$$

Como v^* não é o verdadeiro vetor-código transmitido, um erro de decodificação é cometido. Usando a *Tabela 4.4,* o código é capaz de corrigir qualquer erro simples sobre um bloco de 7 dígitos. Quando 2 ou mais erros ocorrem, um erro de decodificação será cometido.

A decodificação por síndrome de um código linear (n, k) pode ser implementada como mostrado a seguir [4]. A tabela de decodificação é representada como uma tabela com n funções de troca:

$$\begin{aligned} e_0 &= f_0(s_0, s_1, ..., s_{n-k-1}), \\ e_1 &= f_1(s_0, s_1, ..., s_{n-k-1}), \\ &..... \\ e_{n-1} &= f_{n-1}(s_0, s_1, ..., s_{n-k-1}), \end{aligned}$$

Neste caso, $s_0, s_1, ..., s_{n-k-1}$são os dígitos da síndrome, que são considerados como troca de variáveis, e $e_0, e_1, ..., e_{n-1}$ são os dígitos estimados do erro. Quando estas n funções de troca são derivadas e simplificadas, pode ser feito um circuito de combinação lógica com $n - k$ dígitos da síndrome como entradas e o número de dígitos estimados do erro como saídas. O custo de um decodificador depende primariamente da complexidade da combinação lógica do circuito.

4.8 CONCLUSÃO

Neste capítulo foram apresentados alguns aspectos relativos aos códigos lineares de bloco, tais como, características, codificação, decodificação, detecção e correção de erros. No próximo capítulo será mostrado como estes códigos poderão ser utilizados em um sistema de apresentação de senhas.

Capítulo 5

APRESENTAÇÃO DE SENHAS EM MÁQUINAS HOSTIS

5.1 INTRODUÇÃO

O sistema de apresentação de senhas proposto neste trabalho está fundamentado na estrutura do arranjo padrão apresentado na seção 4.6 do capítulo anterior. O objetivo do sistema é propor novos requisitos de segurança além dos sistemas de gerenciamento de senhas já existentes. Propõe-se um ambiente diferenciado para apresentação de senhas, onde não é necessário que o usuário exponha a sua senha. Basicamente, um ruído é adicionado à senha quando esta é apresentada pelo usuário ao servidor.

O capítulo está estruturado da seguinte forma: a seção 5.2 mostra os objetivos do sistema. A seção 5.3 define o cenário proposto para o funcionamento do sistema. A seção 5.4 apresenta os protocolos de comunicação usuário/servidor. Na seção 5.5 é feita uma comparação entre os protocolos. A seção 5.6 mostra uma análise do sistema e na seção 5.7 são definidas algumas vantagens e desvantagens da proposta apresentada.

5.2 OBJETIVOS DO SISTEMA

O sistema proposto atende aos seguintes objetivos: a senha uma vez cadastrada não é mais transmitida pela rede; a senha não é mostrada na tela do terminal do usuário; não é necessário que o usuário digite ou escolha a sua senha na tela do terminal em que esta acessando; o usuário consegue informar a sua senha correta para o sistema sem ter que mostrá-la, dígito a dígito, como ocorre normalmente.

5.3 CENÁRIO PROPOSTO

No sistema de apresentação de senhas proposto, supõe-se que todos os requisitos de segurança relativos ao servidor e à transmissão de dados são previamente atendidos. Isto significa que para o bom funcionamento do sistema, deve ser utilizado um servidor que seja uma máquina segura e a transmissão dos dados também deve ser segura. Uma boa estratégia para a transmissão dos dados é enviá-los cifrados [1]. Além disso, a máquina do usuário é considerada hostil.

Assume-se ainda, que a senha já está previamente cadastrada no servidor. Esta senha denominada $\overrightarrow{P}_w$ tem o formato $\overrightarrow{P}_w = (\alpha_1, \alpha_2, ..., \alpha_n)$, onde $\alpha_i = \{0, 1\}$ e n é o número de dígitos da senha.

Na definição deste sistema são utilizadas algumas estruturas necessárias ao funcionamento do sistema, tais como:

5.3.1 Matriz de Verificação de Paridade

O servidor gera e armazena uma matriz de verificação de paridade H, cujas linhas devem ser linearmente independentes [7], para que a mesma seja utilizada no cálculo da síndrome.

Exemplo 5.1: A matriz abaixo é uma matriz de verificação de paridade:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

5.3.2 Arranjo Padrão:

O servidor constrói um Arranjo Padrão A como descrito na seção 4.6. Este arranjo é definido a partir da escolha de um espaço vetorial e de um de seus subespaços. O objetivo da criação deste arranjo é armazenar a senha do usuário em uma de suas classes laterais. Isto quer dizer que a senha tem a forma $\overrightarrow{P}_w = \overrightarrow{v}_i + \overrightarrow{e}_j$. Concluída a construção do arranjo padrão, um dos vetores de cada linha (classe lateral) deve ser escolhido como líder de cada classe.

Exemplo 5.2: A seguir, na *Tabela 5.1* é mostrado um Arranjo com oito classes laterais.

5.3.3 Síndrome:

A síndrome será o elo de ligação entre o arranjo A e a identificação do usuário. Ela é utilizada para verificar a validade da senha no Protocolo 2.

O servidor calcula a síndrome de todos os representantes das classes laterais. Seja e_i um representante de classe lateral e H a matriz de verificação de paridade, a síndrome é dada por: $s_i = e_i \cdot H^T$. É importante lembrar que cada classe lateral possui uma síndrome diferente.

Exemplo 5.3: Utilizando-se a matriz de verificação de paridade do *Exemplo 5.1*, o arranjo do *Exemplo 5.2* e utilizando-se a fórmula dada acima, obtem-se as seguintes síndromes:

$$(000), (110), (101), (011), (001), (010), (100) \text{ e } (111)$$

Após o cálculo, é montada uma tabela com duas colunas. A primeira armazena a identificação do usuário e a segunda a síndrome de suas respectivas senhas. Esta tabela deve ficar armazenada no servidor.

Exemplo 5.4: Utilizando-se as síndromes encontradas no exemplo anterior tem-se a *Tabela 5.2*.

Tabela 5.1: Exemplo de um Arranjo

000000	001110	010101	100011	011101	101101	110110	111000
000001	001111	010100	100010	011010	101100	110111	111001
000010	001100	010111	100001	011001	101111	110100	111010
000100	001010	010001	100111	011111	101001	110010	111100
001000	000110	011101	101011	010011	100101	111110	110000
010000	011110	000101	110011	001011	111101	100110	101000
100000	101110	110101	000011	111011	001101	010110	011000
001001	000111	011100	101010	010010	100100	111111	110001

Tabela 5.2: Tabela de Identificação

Identificação do Usuário	Síndrome
Alice	(000)
José	(110)
Ana	(101)
Bob	(011)
Pedro	(001)
Maria	(010)
Vilma	(100)
Paulo	(111)

5.3.4 Vetores de Apresentação:

Dados um arranjo A e uma senha $\overrightarrow{P}_w$, o servidor deve definir os conjuntos de vetores de apresentação para esta senha. Estes conjuntos são utilizados em sessões de comunicação entre o servidor e o usuário, da seguinte forma: a senha do usuário é dividida em q blocos com y bits cada um. Para cada bloco P_i são gerados *t* vetores de apresentação dos quais um, e apenas um deles deve conter um bloco P_i que faz parte da senha do usuário. A cada sessão são gerados q conjuntos com t vetores de apresentação. É a partir deles que o usuário prova conhecer a sua senha.

Notação 5.1: $\overrightarrow{P}_w = (\overrightarrow{P}_1, \overrightarrow{P}_2, ..., \overrightarrow{P}_q)$: = {$\overrightarrow{P}_j$ é o j-ésimo bloco de bits da senha}

Os conjuntos de vetores de apresentação devem estar associados a A e a $\overrightarrow{P}_w$ e podem ser gerados por pré-processamento ou não, dependendo da política adotada pelo servidor. Os vetores de apresentação são obtidos pelo algoritmo a seguir:

Dados de entrada: $\overrightarrow{P}_w, y, x, n$

{ $\overrightarrow{P}_w$ é a senha do usuário e pertence a uma das classes laterais do arranjo, y é o tamanho dos blocos de bits em que a senha é dividida, x é o dimensão de cada vetor de apresentação, t é a quantidade de vetores em cada conjunto}

Passos do Algoritmo:

1. Seja $n = \left|\overrightarrow{P}_w\right|$ e $q = \frac{n}{y}$

{para simplificar, é considerado que y divide n, q é igual à quantidade de conjuntos de vetores de apresentação que são apresentados ao usuário}

2. Para j = 1 até q faça:

2.1 Defina t vetores $\overrightarrow{z}^j_1, \overrightarrow{z}^j_2, ..., \overrightarrow{z}^j_j, ..., \overrightarrow{z}^j_t; \overrightarrow{z}^j_i \in R^x$ e existe um único vetor $\overrightarrow{z}^j_i$ tal que o número P_j é coordenada de $\overrightarrow{z}^j_s$ para algum s tal que $1 \leq s \leq t$.

Dados de saída: Conjunto de vetores de apresentação $\left\{\overrightarrow{z}^j_1, \overrightarrow{z}^j_2, ..., \overrightarrow{z}^j_j, ..., \overrightarrow{z}^j_t\right\}$, onde $1 \leq j \leq q$.

Exemplo 5.5: Suponha que os dados de entrada do algoritmo sejam:

$\overrightarrow{P}_w = 100111$, $y = 3$, $x = 3$, $t = 3$

Passo 1:

$q = \frac{n}{y} = \frac{6}{3} = 2$

Logo, $\overrightarrow{P}_w$ é dividida em $\overrightarrow{P}_1 = (100)$ e $\overrightarrow{P}_2 = (111)$

Passo 2:

Para $j = 1$, considere:

$\overrightarrow{z}^1_1 = (000, \infty, \diamond)$, $\overrightarrow{z}^1_2 = (\nabla, 101, \wp)$, $\overrightarrow{z}^1_3 = (\hbar, \Phi, 100)$

Para $j = 2$, considere:

$\overrightarrow{z}^2_1 = (111, \Theta, \Omega)$, $\overrightarrow{z}^2_2 = (\Psi, 010, \wedge)$, $\overrightarrow{z}^2_3 = (\vee, 001, \Rightarrow)$

Portanto, os dados de saída serão:

$\{(000, \infty, \diamond), (\nabla, 101, \wp), (\hbar, \Phi, 100)\}$ e $\{(111, \Theta, \Omega), (\Psi, 010, \wedge), (\vee, 001, \Rightarrow)\}$ que são os dois conjuntos de vetores de apresentação a serem mostrados para o usuário em uma sessão em que a senha

foi dividida em duas partes.

5.3.5 Função de Transformação dos Vetores de Apresentação:

Após a obtenção do conjunto de vetores de apresentação, o protocolo utiliza uma função f, tal que:

$$f : R^2 \rightarrow R$$

O objetivo desta função é transformar o número do vetor escolhido s, utilizado para representar o bloco j da senha em um número binário β_j. Dada f, se o número $\overrightarrow{P}_j$ é coordenada de $\overrightarrow{z}^j_s$, então $f(s, j) = \beta_j$, onde a concatenação dos valores β_j é tal que:

$$\beta = \beta_1 \| \beta_2 \| ... \| \beta_q$$

Notação 5.2: Neste trabalho $\lambda \in$R, a representação binária de λ é considerada um vetor de bits e é indicado por $\overrightarrow{\lambda}$.

Considerando-se β como um número binário, ele pode ser expresso como um vetor de bits, sendo indicado por $\overrightarrow{\beta}$. Além disso, a função f deve ser tal que β pertence à mesma classe lateral de $\overrightarrow{P}_w$.

Exemplo 5.6: A partir do exemplo anterior, escolhendo-se os valores corretos, a função f é aplicada da seguinte forma:

$f(3, 1) = \beta_1 = 100$

$f(1, 2) = \beta_2 = 111$

Logo, $\overrightarrow{\beta} = \beta_1 \| \beta_2 = 100111$, que neste caso é a própria senha do usuário.

5.3.6 Tratamento de Ocorrências Múltiplas:

O tratamento de ocorrências múltiplas definido a seguir também é um conceito utilizado no funcionamento do protocolo 1, definido na próxima seção. O objetivo é mostrar como é feita a interseção

entre um vetor $\overrightarrow{\beta}_j$ e a classe lateral da senha. Se a interseção com todos os vetores da classe lateral da senha for vazia, significa que o valor escolhido pelo usuário não representa um valor da classe lateral da senha. Caso contrário, significa que houve interseção com pelo menos uma parte de um vetor da classe lateral da senha. Supondo que os vetores da classe lateral da senha também são divididos em y bits, $\overrightarrow{\beta}_j$ deve corresponder a uma parte $\overrightarrow{w}_i^j$, também com y bits de pelo menos um dos vetores da classe lateral da senha, como mostrado a seguir:

Seja $\{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\}$tal que $\overrightarrow{w}_i \in GF(2^n)$ e $\overrightarrow{w}_i$é o i-ésimo vetor da classe lateral da senha. Observe que m é a quantidade de vetores de uma classe lateral e $m = 2^k$. O vetor $\overrightarrow{w}_i$tem n bits, pois $\left|\overrightarrow{P}_w\right| = |\overrightarrow{w}_i| \forall i$. Os bits de $\overrightarrow{w}_i$ podem ser divididos em q blocos de y bits. Neste caso, $\overrightarrow{w}_i$ é denotado por $(\overrightarrow{w}_i^1, \overrightarrow{w}_i^2, ..., \overrightarrow{w}_i^q)$ onde $\overrightarrow{w}_i^j \in GF(2^y)$.

Considere $\overrightarrow{\beta}_j \in GF(2^y)$, isto é, $\overrightarrow{\beta}_j$ tem y bits. A interseção de $\overrightarrow{\beta}_j$ e o conjunto de vetores $\{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\}$é denotada por $\overrightarrow{\beta}_j \cap \{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\}$, e é definida como:

$\overrightarrow{\beta}_j \cap \{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\} = \{\overrightarrow{w}_i; \overrightarrow{w}_i = (\overrightarrow{w}_i^1, \overrightarrow{w}_i^2, ..., \overrightarrow{w}_i^q), \overrightarrow{w}_i^j = \overrightarrow{\beta}_j\}$. Observe que $\left|\overrightarrow{w}_i^j\right| = y$, para $j; 1 \leq j \leq q$.

Este conjunto é denominado de conjunto das ocorrências múltiplas de β_j em $\{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\}$. Ele é formado pelos vetores em $\{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\}$que têm β_j como j-ésimo bloco.

Exemplo 5.7: Suponha que é feita a interseção entre $\beta_1 = 100$ e os vetores da classe lateral da senha do usuário, que neste caso é a quarta linha do arranjo mostrado no *Exemplo 5.2*. Assim, todos os vetores desta classe lateral serão divididos em duas partes de 3 bits cada uma, como foi feito com a senha do usuário e mostrado no *Exemplo 5.5*. Como corresponde à primeira parte, então ele é comparado com os 3 primeiros bits de cada vetor da quarta linha do arranjo. Logo, a interseção de β_1 com a classe lateral da senha, é dada por : $\{100111\}$.

5.4 PROTOCOLOS DE COMUNICAÇÃO USUÁRIO/SERVIDOR

A comunicação entre usuário e servidor ocorre quando o usuário acessa um terminal (máquina hostil) e a partir dele se identifica e estabelece comunicação com o servidor (máquina segura) que se

encarregará de permitir ou não a realização das operações desejadas pelo usuário através da verificação de sua senha.

São propostos dois protocolos: no primeiro, considera-se armazenada no servidor uma tabela com duas colunas, a primeira com a identificação do usuário e a segunda com a classe lateral de sua senha $\overrightarrow{P}_w$ em um arranjo A. No segundo protocolo, o servidor armazena uma tabela com duas colunas, onde a primeira contém a identificação do usuário e a segunda a síndrome de sua senha $\overrightarrow{P}_w$, calculada a partir do arranjo A.

5.4.1 Protocolo 1

Os passos do protocolo a seguir mostram o seu funcionamento, onde são armazenadas no servidor as identificações dos usuários e as respectivas classes laterais de suas senhas $\overrightarrow{P}_w$ em um arranjo A. Para um melhor entendimento, serão utilizados os resultados obtidos nos exemplos anteriores, também como exemplos, para ilustrar cada passo:

Passo 1: O usuário envia sua identificação. O usuário tem acesso a um terminal e envia sua identificação para o servidor. Esta identificação pode ser um login, o número de um cartão pessoal ou qualquer outro tipo de identificação. Este passo é importante para que o servidor possa imediatamente associar o usuário à sua respectiva senha.

Exemplo 5.8: Suponha que a identificação seja o nome do usuário e que, neste caso, o usuário seja Bob.

Passo 2: O servidor recebe a identificação do usuário.

Exemplo 5.9: O servidor recebe o nome Bob como sendo a identificação do usuário.

Passo 3: A partir da identificação do usuário, o servidor determina a classe lateral de $\overrightarrow{P}_w$ em A, denotada por $\{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\}$.

Exemplo 5.10: O servidor faz uma pesquisa na Tabela de Identificação mostrada no *Exemplo 5.4* e descobre que a síndrome associada a Bob é (011). A partir da síndrome é possível identificar a classe lateral onde está a senha de Bob. Neste caso, o servidor conclui que a classe lateral procurada é:

$$000100, 001010, 010001, 100111, 011111, 101001, 110010, 111100.$$

Passo 4: O servidor envia o primeiro conjunto $\left\{\overrightarrow{z}^1_1, \overrightarrow{z}^1_2, ..., \overrightarrow{z}^1_t\right\}$ gerado pelo algoritmo de determinação dos conjuntos de apresentação.

Exemplo 5.11: Pelo *Exemplo 5.5,* o primeiro conjunto enviado é $\{(000, \infty, \diamond), (\nabla, 101, \wp), (\hbar, \Phi, 100)\} = \left\{\overrightarrow{z}^1_1, \overrightarrow{z}^1_2, \overrightarrow{z}^1_3\right\}$

O usuário visualiza o conjunto $\left\{\overrightarrow{z}^1_1, \overrightarrow{z}^1_2, ..., \overrightarrow{z}^1_t\right\}$em seu terminal.

Passo 6: O usuário escolhe um vetor do conjunto. Ele deve escolher $\overrightarrow{z}^1_{s_1}$ tal que $\overrightarrow{P}_1$ é coordenada de $\overrightarrow{z}^1_{s_1}$. $\overrightarrow{P}_1$é o primeiro bloco de $\overrightarrow{P}_w$.

Exemplo 5.12: Pelos exemplos anteriores, ele deve escolher $\overrightarrow{z}^1_3$.

Passo 7: O sistema do usuário envia o número do vetor escolhido s_1.

Exemplo 5.13: Nos exemplos mostrados, $s_1 = 3$.

Passo 8: O servidor recebe s_1.

Passo 9: O servidor aplica a função de transformação de vetores de apresentação a s_1e encontra $\overrightarrow{\beta}_1$, ou seja, $f(s_1, 1) = \overrightarrow{\beta}_1$.

Exemplo 5.15: Conforme o *Exemplo 5.6,* $s_1 = 1$ e $f(3, 1) = 100$, isto é, $\overrightarrow{\beta}_1 = 100$.

Passo 10: O servidor calcula a intercessão de $\overrightarrow{\beta}_1$ com a classe lateral de $\overrightarrow{P}_w$ em A, denotada por $\{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\}$. O cálculo é feito para evitar que se leve a operação até o fim, já que é possível saber se já ocorreu um erro do usuário.

Exemplo 5.16: Conforme o *Exemplo 5.7*: $\overrightarrow{\beta}_1 \cap \{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\} = \{100111\}$.

Passo 11: Se $\overrightarrow{\beta}_1 \cap \{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\} = \emptyset$, o servidor aborta a execuçã o do protocolo.

Exemplo 5.17: A interseção vazia significa que $\overrightarrow{\beta}_1 = 100$ não é o início de nenhum vetor da classe lateral da senha. Logo, a escolha do usuário já está errada e o protocolo é interrompido.

Passo 12: Se $\overrightarrow{\beta}_1 \cap \{\overrightarrow{w}_1, \overrightarrow{w}_2, ..., \overrightarrow{w}_m\} \neq \emptyset$, o servidor envia o próximo conjunto, $\left\{\overrightarrow{z}^2_1, \overrightarrow{z}^2_2, ..., \overrightarrow{z}^2_t\right\}$, também gerado pelo algoritmo de determinação dos conjuntos de apresentação.

Exemplo 5.18: Pelos exemplos anteriores, a interseção não vazia significa que $\overrightarrow{\beta}_1 = 100$ é o início de pelo menos um dos vetores da classe lateral da senha e, portanto, o próximo valor pode ser escolhido.

Os Passos de 4 a 12 são repetidos até que se chegue no último valor de q, onde são encontrados e verificados os valores de $\overrightarrow{\beta}_2$ a $\overrightarrow{\beta}_q$. Se durante este processo alguma destas q intercessões for vazia, o protocolo é abortado naquele passo. Caso contrário,isto é, caso nenhuma interseção entre a classe lateral de $\overrightarrow{P}_w$ e um $\overrightarrow{\beta}_j$ seja vazia, ou seja, todos os $\overrightarrow{\beta}_q$ estão na classe lateral de $\overrightarrow{P}_w$, o servidor envia ao usuário uma mensagem de senha reconhecida. Então, o usuário recebe uma mensagem de senha reconhecida e pode prosseguir com suas operações.

5.4.2 Protocolo 2

Os passos a seguir mostram uma variação do *Protocolo 1* apresentado anteriormente, chamada *Protocolo 2*. O objetivo desta variação é mostrar uma opção ao armazenamento da classe lateral de

$\overrightarrow{P}_w$. Neste caso, após a montagem da tabela de síndromes e o cálculo dos conjuntos de vetores de apresentação, o servidor pode abrir mão de manter armazenada a classe lateral de $\overrightarrow{P}_w$, sendo armazenada apenas a síndrome desta classe. Neste caso, também serão feitos alguns comentários, numerados como exemplos, referentes aos resultados obtidos em exemplos anteriores a título de ilustração:

Passo 1: O usuário tem acesso a um terminal e envia sua identificação para o servidor. Esta identificação pode ser um login, o número de um cartão pessoal ou qualquer outro tipo de identificação. Este passo é importante para que o servidor possa imediatamente associar o usuário à sua respectiva senha.

Exemplo 5.19: Suponha que a identificação seja o nome e que neste caso se trate do Bob.

Passo 2: O servidor recebe a identificação do usuário.

Exemplo 5.20: O servidor recebe o nome Bob.

Passo 3: O servidor identifica a síndrome de $\overrightarrow{P}_w$. A síndrome deve estar previamente armazenada numa tabela no servidor.

Exemplo 5.21: O servidor faz uma pesquisa na Tabela de Identificação (*Exemplo 5.4*) e encontra (011) como sendo a síndrome associada a Bob.

Passo 4: O servidor envia todos os conjuntos de vetores de apresentação $\left\{\overrightarrow{z}^j_1, \overrightarrow{z}^j_2, ..., \overrightarrow{z}^j_t\right\}$, com $1 \leq j \leq q$, gerados pelo algoritmo de determinação dos conjuntos de apresentação. Neste caso, q vetores de apresentação são calculados a partir da síndrome de $\overrightarrow{P}_w$, que também identifica a classe lateral de $\overrightarrow{P}_w$.

Exemplo 5.22: Conforme o exemplo são enviados os seguintes conjuntos:

$\{(000, \infty, \diamond), (\nabla, 101, \wp), (\hbar, \Phi, 100)\}$ e $\{(111, \Theta, \Omega), (\Psi, 010, \wedge), (\vee, 001, \Rightarrow)\}$

Passo 5: O usuário visualiza os conjuntos $\left\{\overrightarrow{z}^j_1, \overrightarrow{z}^j_2, ..., \overrightarrow{z}^j_t\right\}$ em seu terminal.

Passo 6: O usuário escolhe um vetor de cada conjunto. Cada vetor escolhido $\overrightarrow{z}^j_{s_j}$ deve conter a parte $\overrightarrow{P}_j$ da senha. Isto significa que $\overrightarrow{P}_j$ é coordenada de $\overrightarrow{z}^j_{s_j}$ e $1 \leq j \leq q$. $\overrightarrow{P}_j$ é o j-ésimo bloco da senha.

Passo 7: O usuário envia os números dos vetores escolhidos s_j.

Exemplo 5.23: Conforme os exemplos anteriores, os valores de s_j devem ser: $s_1 = 3$ e $s_2 = 1$.

Passo 8: O servidor recebe os valores de s_j, com j sempre variando entre 1 e q, respectivamente.

Passo 9: O servidor aplica a função de transformação de vetores de apresentação aos s_j e encontra os respectivos $\overrightarrow{\beta}_j$, ou seja, $f(s_j, j) = \overrightarrow{\beta}_j$.

Exemplo 5.24: No caso dos exemplos, $\beta_1 = 100$ e $\beta_2 = 111$.

Passo 10: Os $\overrightarrow{\beta}_j$ são concatenados para formar $\overrightarrow{\beta}$.

Exemplo 5.25: Seguindo os exemplos, $\overrightarrow{\beta} = (100111)$

Passo 11: O servidor calcula a síndrome de $\overrightarrow{\beta}$, $s = \overrightarrow{\beta}.H^T$.

Exemplo 5.26: O valor da síndrome encontrado para estes exemplos é $s = (011)$.

Passo 12: O servidor faz uma busca na tabela de síndromes. Seu objetivo é encontrar na tabela uma síndrome que seja igual à síndrome de $\overrightarrow{\beta}$.

Passo 13: Se a síndrome de $\overrightarrow{\beta}$ é diferente da síndrome associada ao usuário, a execução do protocolo é abortada, pois a senha não foi reconhecida.

Passo 14: Caso contrário, se a síndrome de $\overrightarrow{\beta}$ é igual à síndrome associada ao usuário, o servidor envia uma mensagem de senha reconhecida.

Passo 15: O usuário recebe uma mensagem de senha reconhecida e pode prosseguir com suas operações.

Observe que, neste caso, o protocolo é executado até o último passo, mesmo que o usuário cometa algum erro logo no início de sua execuç ão.

5.5 COMPARAÇÃO ENTRE OS PROTOCOLOS

5.5.1 Protocolo 1:

- Como a classe lateral de $\overrightarrow{P}_w$ em A deve ficar armazenada e é utilizada durante todo o processo de comunicação, isto eleva o custo de armazenamento;
- A desconexão em caso de erro é rápida pelo fato de que cada valor escolhido pelo usuário é submetido à função f para gerar o $\overrightarrow{\beta}_j$ correspondente. Este valor é imediatamente verificado e leva à interrupção da comunicação em caso de erro. Isto significa que a operação só irá prosseguir até o final se o usuário não cometer nenhum erro;
- Este protocolo contém mais passos que o Protocolo 2. Mas em cada passo, a quantidade de dados transmitidos é igual ou menor que no Protocolo 2.

5.5.2 Protocolo 2:

- Como o arranjo A deve estar no servidor apenas no início da comunicação e logo depois pode ser descartado, isto reduz o custo de armazenamento do sistema;
- A desconexão é lenta, pois o sistema deverá receber todos os valores escolhidos pelo usuário

para serem submetidos à função f e encontrar os $\overrightarrow{\beta}_j$ que são concatenados para formar $\overrightarrow{\beta}$. Só então a validade da senha é verificada. Isto significa que mesmo que o usuário cometa um erro no início de suas escolhas, o sistema continua executando todos os passos do protocolo;

- O Protocolo 2 possui menos passos que o Protocolo 1.

5.6 ANÁLISE DO SISTEMA

A seguir são analisados alguns pontos relativos ao projeto e a alguns custos do sistema. É importante ressaltar que os exemplos apresentados anteriormente são didáticos e têm por objetivo o bom entendimento dos protocolos e não podem ser utilizados para analisar a eficiência dos mesmos, pois trabalham com valores pequenos.

5.6.1 Com relação à senha :

- A senha é composta por n dígitos. Quanto maior for o valor de n, menor a chance dela ser descoberta por força bruta;
- A forma como a senha deve ser dividida, citada anteriormente, fica a critério do servidor. É ele quem escolhe o parâmetro y tal que $q = \frac{n}{y}$;
- Caso o servidor faça a opção por dividir a senha dígito a dígito, o custo computacional aumenta, pois neste caso, $q = n$. Esta situação provoca também um aumento significativo na quantidade de conjuntos de apresentação a serem gerados e mostrados.

5.6.2 Com relação ao arranjo A:

- Um mesmo A pode ser usado para armazenar a senha de diversos usuários;
- Supondo A uma matriz com a linhas e b colunas, a possibilidade dele ser descoberto por força bruta é da ordem de $\frac{1}{a.b}$. Por isso, quanto maiores forem suas dimensões, menores as chances de sucesso de um ataque por força bruta;

• O servidor pode modificar A, a cada intervalo de tempo t' para dificultar um ataque por força bruta, mesmo se tratando de arranjos de grandes dimensões. O intervalo de tempo t‘ deve ser definido pelo servidor;

5.6.3 Com relação à matriz de verificação de paridade H:

• Esta matriz também pode ser alterada em intervalos de tempo t'_1, definidos pelo servidor para dificultar ataques por força bruta.

5.6.4 Com relação à síndrome s:

• No Protocolo 2, foi sugerida a construção de uma tabela onde ficam armazenadas as síndromes de todas as classes laterais. Porém, o servidor pode não armazenar estes dados numa tabela e optar por fazer os cálculos sempre que estes valores forem necessários. Isto acarreta um custo computacional maior e o arranjo A deve ficar armazenado.

• Ainda no Protocolo 2, a construção da tabela de síndromes deve obrigatoriamente ser feita no início da comunicação, já que a proposta abre mão do armazenamento do arranjo A.

5.6.5 Com relação ao algoritmo de determinação dos vetores de apresentação:

• O algoritmo pode ser executado no momento da comunicação ou ser previamente executado e, neste caso, os conjuntos devem ficar armazenados no servidor.

5.6.6 Com relação aos conjuntos de vetores de apresentação:

• Os vetores do conjunto $\left\{ \overrightarrow{z}^j_1, \overrightarrow{z}^j_2, ..., \overrightarrow{z}^j_t \right\}$devem ser alterados a cada nova apresentação de senha;

• A intercessão entre os t vetores de apresentação dos conjuntos mostrados deve ser nula, para evitar que o usuário escolha o conjunto errado e não consiga provar que conhece a sua senha;

• Quanto maior o valor de x, maior a segurança do sistema. O servidor pode também optar por mudá-lo em intervalos de tempo t'_2;

• Quanto maior o valor de t, maior a segurança do sistema, já que isso dificultaria ao espião que tentar fazer a intercessão dos vetores de apresentação ($\overrightarrow{z}_i^j \cap \overrightarrow{v}_i^j$). O valor de t também pode ser alterado em intervalos de tempo t'_3, definidos pelo servidor;

• No Protocolo 1, os conjuntos de vetores de apresentação são enviados um a um para o usuário, o que provoca um tráfego intenso de informações;

• No Protocolo 2, os q conjuntos de vetores de apresentação são enviados ao usuário todos de uma vez, o que diminui o tráfego, mas torna a transmissão mais lenta.

5.6.7 Com relação aos intervalos de tempo:

• Para aumentar ainda mais a segurança do sistema, os intervalos t', t'_1, t'_2 e t'_3 devem ser pequenos, aleatórios e diferentes entre si.

5.6.8 Com relação ao vetor $\overrightarrow{\beta}$:

• No Protocolo 1, $\overrightarrow{\beta}$ não chega a ser formado. Suas partes $\overrightarrow{\beta}_j$ são utilizadas separadamente na verificação de sua intercessão com o arranjo A;

• No Protocolo 2, os blocos $\overrightarrow{\beta}_j$ são gerados todos de uma vez, a partir da aplicação da função f aos valores s_j enviados pelo usuário. Após a geração, eles são concatenados e formam o vetor $\overrightarrow{\beta}$, que é fundamental na verificação da validade da senha. Neste caso, na implementação do sistema pode-se optar por fazer a montagem do vetor $\overrightarrow{\beta}$ na máquina do usuário. A captura de $\overrightarrow{\beta}$ durante sua transmissão, não significa que o espião descobrirá a senha;

5.7 VANTAGENS E DESVANTAGENS DA PROPOSTA APRESENTADA

5.7.1 Vantagens

- O sistema proposto não expõe a senha do usuário em momento algum de seu funcionamento. A qualquer momento da comunicação, caso um espião capture alguma ou todas as informações transmitidas, elas não lhe contarão a senha. Esta segurança deve ser ainda mais reforçada, utilizando-se técnicas de cifragem sobre os dados a serem transmitidos;
- O sistema pode ser utilizado pelo usuário em máquinas hostis, que é a situação mais comum de acontecer;
- Caso haja alguma pessoa observando os conjuntos escolhidos pelo usuário, ela não verá a senha e os conjuntos escolhidos pelo usuário dificultarão a descoberta da senha, já que os mesmos são gerados aleatoriamente pelo servidor e trocados a cada vez que a senha é requisitada;
- A parte algébrica do processo é transparente ao usuário;
- A interface a ser utilizada na apresentação dos conjuntos fica a critério da aplicação onde o sistema é utilizado;
- A utilização da teoria dos códigos é computacionalmente vantajosa, já que seu custo é baixo e, além disso, o sistema pode ser implementado em hardware;

5.7.2 Desvantagens

- Mesmo adotando-se todas as medidas recomendadas no projeto, um ataque por força bruta não pode ser completamente descartado;
- A dificuldade em se montar os conjuntos de vetores de apresentação pode ser alta;
- Os protocolos propostos não têm a mesma segurança de protocolos criptográficos.

5.8 CONCLUSÃO

Neste capítulo foi apresentada uma proposta de um sistema de apresentação de senhas em máquinas hostis, seus objetivos, funcionamento, análise, vantagens e desvantagens. Os protocolos propostos protegem a senha em uma eventual transação, tornando-a mais segura.

Capítulo 6

CONCLUSÃO

Nesta dissertação, é proposto um sistema de apresentação de senhas que visa aperfeiçoar os sistemas existentes [17] de modo a aumentar a segurança e garantir a confidencialidade da senha. Em momento algum o usuário informa sua senha, ele deverá sim, provar que a conhece. Seguidas todas as recomendações sugeridas na análise do sistema, tem-se um sistema que atende o que ele se propõe. Ele pode ser utilizado em qualquer sistema que exija a apresentação de senhas.

São apresentados também nesta dissertação, os fundamentos de criptografia, estruturas algébricas e teoria dos códigos.

Como proposta de trabalhos futuros, sugere-se:

- A definição de um protocolo que utilize o paradigma apresentado e que tenha a segurança de um protocolo criptográfico e não dependa do tamanho do arranjo e dos conjuntos de vetores de apresentação;
- Uma análise matemática da complexidade da segurança do protocolo;
- Implementação do protocolo e avaliação da satisfação de usuários em geral.

Referências Bibliográficas

[1] BUCHMANN, JOHANNES A. "Introdução à Criptografia", Editora Berkeley, 2002.

[2] CHAUDHURI, DIJEN RAY "Coding Theory and Design Theory", Editora Springer, 1990.

[3] COHEN, G., GODLEWSKI, P.(et all) "Coding Theory and Applications"', Editora Springer, 1998.

[4] COSTELLO JR., DANIEL J.,LIN, SHU "Error Control Coding-Fundamentals and Applications", Editora Prentice Hall, 1983.

[5] EVARISTO, JAIME, PERDIGÃO, EDUARDO "Introdução à Álgebra Abstrata"', EDUFAL, 2002.

[6] FILHO, JOEL GUILHERME SILVA "Informação, Codificação e Segurança de Dados", ENE, 1998.

[7] HEFEZ, ABRAMO, VILLELA, MARIA LÚCIA T. "Códigos Corretores de Erros", Editora IMPA, 2002.

[8] HILLEY, SARAH "Ways to Crack a Password", Network Security - ISSN 13534858, 2003.

[9] LINT, JACOBUS-HENDRICUS VAN "Introduction to Coding Theory", Editora Springer, 1992.

[10] LOURENÇO, MARY LILIAN, COELHO, FLÁVIO ULHOA, "Um Curso de Álgebra Linear", EDUSP, 2001.

[11] LUCCHESI, CLÁUDIO LEONARDO "Introdução à Criptografia Computacional", Editora da Unicamp, 1986.

[12] MITCHELL, CHRIS J. "Security Techniques", IEEE, 1995.

[13] RENAUD, KAREN, ANGELI, ANTONELLA DE "My password is here! An investigation into visuo-spatial authentication mechanisms", Interacting With Computers xx 1-25, 2004.

[14] RIVEST, R. SHAMIR, A., ADLEMAN, L. "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", Comunications of the ACM, 1978.

[15] RODRIGUES, ALEXANDRE AUGUSTO MARTINS "Álgebra Linear e Geometria Euclidiana"', Editora Nobel, 1965.

[16] SCHNEIER, BRUCE "Customers, Passwords, and Websites", IEEE Security Privacy, 2004.

[17] SCHULTZ, EUGENE "Security Views", Computers Security 23533-511, 2004.

[18] SOUSA, PEDRO MOISÉS DE "Acesso Mandatário em Ambiente Cifrado"', UFU, 2003.

[19] STALLINGS, W. "Cryptography and Network Security: Principles and Pratice", Editora Prentice Hall, 1999.

[20] STANTON, JEFREY M., STAM, KATHRY R., MASTRANGELO, PAUL, JOLTON, JEFFREY "Analisys of end user security behaviors", Computers Security, 2004.

[21] SUMMERS, WAYNE C., BOSWORTH, EDUARD "Password Policy: The Good, The Bad and The Ugly", IEEE, 2004.

[22] VIEIRA, FERNANDA DE JESUS "Uma Extensão do Protocolo SET para Acesso a Base de Dados Cifrado", UFU, 2003.

[23] LIPSCHUTZ, SEYMOUR, "Álgebra Linear", McGraw-Hill do Brasil, 1972.