Barbara Castro Diniz

O USO DA SÍNTESE DE CIRCUITOS REVERSÍVEIS EM ALGORITMOS DE CRIPTOGRAFIA SIMÉTRICA

Trabalho de Curso DEFENDIDO e APROVADO em 28 de fevereiro de 2020, pela Banca Examinadora constituída pelos membros:

Prof. Dr. André da Cunha Ribeiro
IF Goiano – Campus Rio Verde

Prof.(a) Dr.(a) Diane Castonguay
UFG - Instituto de Informática

Prof. Dr.. Márcio Antônio Ferreira Belo Filho
IF Goiano – Campus Rio Verde

Rio Verde - GO
Fevereiro, 2020

BARBARA CASTRO DINIZ

O USO DA SÍNTESE DE CIRCUITOS REVERSÍVEIS EM ALGORITMOS DE CRIPTOGRAFIA SIMÉTRICA

Projeto do Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia Goiano - Campus Rio Verde ligado ao Ministério da Educação, como requisito parcial da disciplina Trabalho de Conclusão de Curso 2 para a obtenção do título de Bacharelado em Ciência da Computação.

Orientador: Dr. André da Cunha Ribeiro
Instituto Federal Goiano

RIO VERDE
2020

À minha família, por sempre ter acreditado em mim. À minha mãe, por todo carinho e apoio que me concedeu nos momentos mais difíceis. Ao meu pai que apesar de todas as dificuldades fez mais do que pôde para garantir que eu estivesse onde estou. Aos meus irmãos, amigos e namorado pela presença e paciência que significou a certeza de que não estou sozinha nessa caminhada, pelas alegrias, tristezas e dores compartilhadas.

AGRADECIMENTOS

Agradeço à Deus, por ter me dado energia e recurso para seguir em frente. Agradeço aos meus pais.Minha mãe por sempre me apoiar em todas as minhas decisões e ao meu pai por garantir e prover tudo que eu precisei. Por estarem sempre comigo, independente da condição e da distância. Aos meus irmãos Lorena e José David por todo carinho e amor que me deram mesmo de longe. Agradecimento especial ao meu namorado Anderson, por estar comigo ao longo de toda essa jornada e me tranquilizar em momentos difíceis.

Agradeço ao meu orientador Dr. André da Cunha Ribeiro por aceitar me orientar lá em 2016 e ter contribuído de forma excepcional para com esta pesquisa, não só com seu conhecimento mas também com o apoio que me deu para não desistir.

Sou grata ao Instituto Federal Goiano e CNPq pelo apoio financeiro através das bolsas de pesquisas, quando esse projeto ainda era um PIBIT, que contribuiu diretamente para a minha permanência no Instituto e conclusão do mesmo projeto.

RESUMO

DINIZ, Barbara Castro. O uso da síntese de circuitos reversíveis em algoritmos de criptografia simétrica. 2020. 32 f. Projeto do Trabalho de Conclusão de Curso – Bacharelado em Ciência da Computação, Instituto Federal de Educação, Ciência e Tecnologia Goiano - Campus Rio Verde. Rio Verde, 2020.

A criptografia é uma técnica de transformar um texto claro em um texto ilegível, para que este seja transmitido em segurança. A criptografia simétrica possui operações simples e depende do sigilo entre remetente e destinatário, deixando a garantia de privacidade a desejar. Pensando em ampliar a segurança, foi analisado a viabilidade de utilizar a síntese de circuitos reversíveis em algoritmos de criptografia simétrica, com isso foi criado um novo algoritmo de criptografia utilizando o método de síntese Hipercubo, proposto por (RIBEIRO, 2013). Esse método realiza operações de trocas em uma permutação, aplicando uma porta lógica reversível consecutivamente, fazendo com que esta permutação caminhe até sua identidade, gerando assim um conjunto de circuitos que será aplicado em toda mensagem que se deseja criptografar. A quantidade de circuitos tem um limite superior de $(n-1)2^n+1$, tornando a complexidade do algoritmo em $O(T*((n-1)2^n+1))$, onde $T =$ total de letras e $n =$ quantidade de bits.

Palavras-chave: Criptografia Simétrica. Circuitos Reversíveis. Algoritmo.

ABSTRACT

DINIZ, Barbara Castro. Symmetric Criptography Algorithms using Reversible Synthesis. 2020. 32 f. Projeto do Trabalho de Conclusão de Curso – Bacharelado em Ciência da Computação, Instituto Federal de Educação, Ciência e Tecnologia Goiano - Campus Rio Verde. Rio Verde, 2020.

Encryption is a technique of turning a clear text into unreadable text, so that it is transmitted securely. A symmetric encryption has simple operations and depends on the confidentiality between sender and receiver, leaving the privacy guarantee to be desired. Thinking about increasing security, the feasibility of using the synthesis of reversible circuits in symmetric cryptography algorithms was analyzed. With this, a new cryptography algorithm was created using the Hipercubo synthesis method, proposed by (RIBEIRO, 2013). This method performs security operations. exchanges in a permutation, applying a reversible logic gate consecutively, making this permutation move towards its identity, thus generating a set of circuits that will be applied to every message that is to be encrypted. The number of circuits has an upper limit of $(n-1)2^n+1$, making the complexity of the algorithm $O(T*((n-1)2^n+1))$, where T = total letters and n = number of bits.

Keywords: Simmetric Cryptography. Reversible Circuits. Algorithm.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RSA	Rivest-Shamir-Adleman
DES	Data Encryption Standard
3DES	Third Data Encryption Standard
AES	Advanced Encryption Standard
IDEA	International Data Encryption Algorithm
RC4	Rivest Cipher Fourth
KSA	Key-Scheduling algorithm
PRGA	Pseudo-Random generation algorithm
RSADSI	RSA Data Security, Inc
CRS-MX	Criptography Reversible Synthesis - Mixed

LISTA DE ALGORITMOS

SUMÁRIO

1 Introdução

Para fazer uma compra pela internet, transações bancárias ou até mesmo fazer um cadastro em uma rede social é exigido algumas informações pessoais. Para que essas informações estejam seguras na internet é preciso criptografá-las.

O principal objetivo da criptografia é tornar a informação inelegível, a fim de impedir o acesso de pessoas não autorizadas à informação (TRINTA; MACÊDO, 1998).

Pode-se criptografar informações basicamente através de códigos ou cifras.

Os códigos protegem as informações trocando partes das mesmas por códigos predefinidos, em que todas as pessoas autorizadas a ter acesso à uma determinada informação devem conhecer os códigos utilizados.

No caso das cifras, são técnicas nas quais a informação é cifrada através da transposição e/ou substituição das letras da mensagem origina, assim, as pessoas autorizadas podem ter acesso às informações originais conhecendo o processo de cifragem (MORENO; PEREIRA; CHIARAMONTE, 2005).

Os principais tipos de cifra são: as cifras de transposição que é a mistura dos caracteres da informação original; e as cifras de substituição onde é possível criptografar um texto apenas substituindo uma letra por outra (MORENO; PEREIRA; CHIARAMONTE, 2005)(CORMEN, 2014).

A criptografia é dividida em duas partes: simétrica e assimétrica.

A criptografia simétrica é um sistema criptográfico que utiliza apenas uma chave, tanto para cifrar quanto para decifrar uma determinada mensagem.

Já criptografia assimétrica é um sistema criptográfico que usa pares de chaves: chave pública, que é de conhecimento de todos e é utilizada para cifrar uma determinada mensagem, e chave privada que é de conhecimento apenas pelo receptor da mensagem criptografada, e é utilizada para decifrar a mensagem.

Neste trabalho foi abordado apenas a criptografia simétrica, especialmente por ter a propriedade de simetria, podendo desfazer a cifragem com a mesma chave.

A criptografia simétrica possui grande vantagem em ter uma premissa de operações mais simples e com baixo custo computacional, mas por outro lado, utilizar chaves iguais para criptografar e descriptografar, dependendo não só da eficiência da chave, mas também do sigilo entre remetente e destinatário, pode ser um problema na garantia de privacidade. Tendo em vista que segurança de uma criptografia está presente também no tamanho da chave, tamanho do bloco a ser cifrado e na complexidade das operações realizadas.

Nos tempos modernos criptografia é considerada um ramo da matemática e ciência da computação e está estreitamente associado à teoria da informação, segurança e engenharia de computadores (LECTURER; SONIPAT, 2010).

O homem sentiu, desde muito cedo, a necessidade de guardar informações em

segredo; e a criptografia nasceu com a diplomacia e com as transações militares. A importância de não revelar segredos e estratégias às forças inimigas, motivou o desenvolvimento de códigos, cifras e técnicas para mascarar uma mensagem, possibilitando apenas ao destinatário ler o conteúdo (MALAGUTTI; BEZERRA; RODRIGUES, 2010).

Nos dias de hoje a privacidade é importante para pessoas e empresas. E muitos problemas podem acontecer se uma pessoa não autorizada tiver acesso a dados pessoais. E no caso de empresas, os danos podem ser de maior magnitude, atingindo a organização e os próprios funcionários (MORENO; PEREIRA; CHIARAMONTE, 2005).

As aplicações da Criptografia atualmente incluem: sigilo em banco de dados; censos; investigações governamentais; dossiês de pessoas sob investigação; dados hospitalares; informações de crédito pessoal; decisões estratégicas empresariais; sigilo em comunicação de dados; comandos militares; mensagens diplomáticas; operações bancárias; comércio eletrônico; transações por troca de documentos eletrônicos (EDI); recuperação de documentos arqueológicos, hieróglifos; e até mesmo na troca de mensagens por WhatsApp (MALAGUTTI; BEZERRA; RODRIGUES, 2010).

Em sistemas criptográficos a privacidade impede a extração de informações por pessoas não autorizadas, garantindo que apenas a origem e o destino tenham conhecimento (OLIVEIRA, 2012).

No entanto, na criptografia simétrica é necessário para as partes comunicantes compartilharem uma chave que ninguém mais conheça (DIFFIE; HELLMAN, 1976).

Pensando nisso foi feita uma análise sobre a viabilidade de se utilizar circuitos reversíveis em um algoritmo de criptografia simétrica, de forma que seja possível aumentar o tamanho do bloco a ser cifrado, mantendo o custo computacional de uma criptografia comumente utilizada e a simplicidade nas operações porém, melhorando a segurança da mesma.

Assim como processamento de sinal e computação gráfica, a criptografia geralmente requer tranformações reversíveis, onde todas as informações codificadas na entrada devem ser preservadas na saída.

Um exemplo comum é trocar dois valores a e b sem intermediário armazenamento usando operações XOR bit a bit $a = a \oplus b$, $b = a \oplus b$, $a = a \oplus b$.

Dado que transformações reversíveis aparecem em gargalos de algoritmos comumente usados, novas instruções foram adicionadas aos conjuntos de instruções de vários microprocessadores (SAEEDI; MARKOV, 2013).

Uma computação é reversível se puder ser "desfeita" no sentido de que a saída contém informações suficientes para reconstruir a entrada, ou seja, nenhuma informação de entrada é apagado (TOFFOLI, 1980).

Para projetar a síntese de circuitos reversíveis, é necessário um conjunto de portas reversíveis. Vários dessas portas foram propostos há décadas, entre elas podemos citar a porta NOT, CNOT e Toffoli. A porta toffoli tem variações que são as portas G-Toffoli e

CM-Toffoli (MASLOV; DUECK, 2004).

A síntese lógica reversível é o processo de gerar um circuito reversível compacto. Os métodos de síntese geralmente assumem uma função reversível completamente especificada como ponto de partida (MILLER; WILLIE; DUECK, 2009).

Quando uma função f é bijetiva, f possui uma função inversa. Portanto, existe um circuito que, para cada valor de saída y de f, produz o valor de x de modo que $f(x) = y$. Neste caso, dizemos que o circuito é reversível (RIBEIRO, 2013).

Uma $n-$porta reversível realiza uma função bijetiva sobre a permutação $0, 1, ..., 2^n - 1$. Para qualquer porta reversível g, a porta g^{-1} realiza a transformação inversa. Cada permutação é uma sequência de $n2^n$ bits (RIBEIRO, 2013).

O objetivo desse projeto foi desenvolver pesquisas que objetivam analisar e estudar a síntese de circuitos reversíveis aplicada a algoritmos de criptografia simétrica. Com isso foi criado um novo modelo de algoritmo de criptografia simétrica fazendo o uso da síntese de um circuito reversível (CRS-MX).

Esse novo algoritmo simétrico pretende criar um modelo de criptografia onde a segurança da criptografia simétrica seja confiável, porém que seja mantida a eficiência das operações simétricas em relação ao desempenho e tempo de execução de algoritmos que não utilizam circuitos reversíveis.

Substituindo o método de síntese de circuitos que foi utilizado, por qualquer outro existente, faz com que este seja um novo algoritmo. Podendo obter vários algoritmos de criptografia simétrica utilizando vários métodos de síntese de circuitos reversíveis.

Nos próximos capítulos serão apresentados a Fundamentação Teórica no Capitulo 2, o Algoritmo CRS-MX no Capítulo 3 e a Conclusão no Capitulo 4.

2 Fundamentação Teórica

2.1 Criptografia

Quando falamos de informação e transportamos este conceito para o meio digital, particularmente na utilização das redes públicas de computação como a internet, é relevante ao ser humano a credibilidade nos sistemas computacionais. Estes que inseridos nos fundamentos da segurança da informação, são definidos pela disponibilidade, integridade, controle de acesso, autenticidade, não-repudiação e finalmente a privacidade, os quais devem ser de livre compreensão e facilmente perceptíveis ao se efetuar transações computacionais (OLIVEIRA, 2012)(FIARRESGA et al., 2010):

- **Disponibilidade** - garantir que uma informação estará disponível para acesso no momento desejado.
- **Integridade** - garantir que o conteúdo da mensagem não foi alterado.
- **Controle de acesso** - garantir que o conteúdo da mensagem somente será acessado por pessoas autorizadas.
- **Autenticidade** - garantir a identidade de quem está enviando a mensagem.
- **Não-repudiação** - prevenir que alguém negue o envio e/ou recebimento de uma mensagem.
- **Privacidade** - impedir que pessoas não autorizadas tenham acesso ao conteúdo da mensagem, garantindo que apenas a origem e o destino tenham conhecimento

No ano 404 A.C. O cinto era composto por tiras de couro com letras escritas do lado avesso. Parecia uma sequência sem sentido de caracteres, mas quando o pedaço de couro era enrolado em torno de um pedaço de madeira com um diâmetro pré estabelecido, uma mensagem descriptografada poderia ser lida.

Foi utilizada por Júlio César no século 1 A.C. Consiste na substituição de uma letra do alfabeto por seu correspondente três casas adiante, ou seja, a letra A é substituída pela letra D, a letra B pela letra E e assim por diante. Neste caso, o algoritmo da cifra é a troca de uma letra por outra em uma determinada posição. E a chave, neste caso, é o número 3.

A cifra de Vigenère, do século XVI,consiste em até 26 alfabetos distintos para criar a mensagem cifrada. O primeiro passo é montar o chamado quadrado de Vigenère, um alfabeto normal seguido de 26 alfabetos cifrados, cada um deslocando uma letra em relação ao alfabeto anterior. Em resumo, o remetente da mensagem pode, por exemplo, cifrar a primeira letra de acordo com a linha 5, a segunda de acordo com a linha 14 e a terceira de acordo com a linha 21, e assim por diante.

A criptografia e, em parte, o próprio computador, surgiram dos esforços dos aliados durante II Guerra Mundial para decifrar os códigos de comunicação usados pelos

submarinos da Alemanha nazista.Em 1918 Scherbius desenvolveu a"Enigma", a Marinha alemã interessou-se pela ideia e em 1926 adotou-a como principal meio de comunicação e criptografia ficando conhecida como Funkschlüssel C. Em 1928, a própria Alemanha desenvolveu sua versão, a Enigma G, e passou a ser usada tendo suas chaves trocadas mensalmente o que impossibilitava aos Aliados manterem-se um passo a frente dos alemães.

Foi o matemático Marian Rejewski, polonês, quem deduziu a estrutura da Enigma em 1932, avanço considerado o mais marcante da criptoanálise desde sua invenção concomitantemente com os ingleses, os franco-poloneses atingiram agilidade e precisão na quebra da Enigma. Entre os britânicos personalidades como Allan Turing, Gordon Welchman e Max Newman atuaram decisivamente para a quebra da Enigma. Max Newman e seus colegas projetaram e implementaram o primeiro computador digital eletrônico programável, o Colossus, para ajudar com sua criptoanálise.

Claude Shannon é considerado o pai da criptografia matemática, com o livro Communication Theory of Secrecy Systems no Bell System Technical Journal, publicado em 1949, estudou, analisou e abordou a criptografia utilizada na Segunda Guerra Mundial. Este tabalho é considerado a base teórica para a criptografia e também para grande parte da criptoanálise.

Na atualidade, as técnicas de criptografias mais conhecidas envolvem o conceito das chaves criptográficas, que são formadas por bits, com base em algarismos com capacidade de interpretar as informações, ou seja, capaz de decodificar. A chave do emissor deve ser compatível com a do receptor, para assim, as informações serem extraídas. Há quatro tipos básicos de aplicações criptográficas, tipicamente usadas para conseguir os objetivos de segurança da informação: criptografia simétrica, ou de chave secreta, criptografia assimétrica, ou de chave pública, função hash criptográfico e assinatura digital.

A criptografia é a arte de codificar uma mensagem clara para que esta seja transmitida em segurança. As nações passaram a criar departamentos para elaborar sistemas criptográficos. Por outro lado, surgiram os decifradores de códigos, criando uma corrida armamentista intelectual. Ao longo da história, os códigos decidiram o resultado de batalhas. À medida que a informação se torna cada vez mais valiosa, o processo de codificação de mensagens tem um papel cada vez maior na sociedade (MALAGUTTI; BEZERRA; RODRIGUES, 2010).

Existem três princípios fundamentais , que se não forem atendidos, não estamos falando de uma criptografia.

1. **Confidencialidade**: Ninguém consegue ler a mensagem que o emissor está enviando para o receptor.
2. **Integridade**: A mensagem que o emissor mandou , tem que ser a mesma mensagem que o receptor recebeu. Tem por fundamento evitar corrupção ou modificação não-autorizada de dados e/ou informações, em todas as suas fases de existência, isto é, geração ou produção, difusão e tramitação, uso, avaliação e destinação final.

3. **Autenticidade**: Quando o receptor receber a mensagem , o mesmo tem que ter certeza que essa mensagem foi enviada pelo emissor desejado e não outra pessoa aleatória.

O termo ”chave”vem do fato que o número secreto que você escolhe funciona da mesma forma que uma chave convencional. Pra proteger seu patrimônio você instala uma fechadura na porta. Para operar a fechadura você instala e gira a chave. Na criptografia, para proteger o conteúdo dos seus arquivos, você instala uma fechadura (algoritmo de criptografia) na sua porta (computador). A chave na criptografia é combinada com uma operação, ou com um conjunto de operações, que faz com que os dados fique seguro tornando-os ilegíveis, e somente com a chave é possível ter acesso aos dados originais (BURNETT; PAINE, 2002).

A criptografia pode ser feita de duas maneiras, com códigos ou cifras. Os códigos foram as primeiras técnicas de criptografia e são feitas através da troca dos caracteres da mensagem por códigos pré-definidos. As cifras fazem a substituição e/ou transposição na mensagem. Na criptografia de cifras incluem o conceito de chaves e existem dois tipos de algoritmos: os assimétricos e os simétricos (MORENO; PEREIRA; CHIARAMONTE, 2005).

2.1.1 Criptografia Assimétrica

A criptografia assimétrica é o processo no qual cada parte envolvida na comunicação usa duas chaves diferentes e complementares, uma privada e outra pública. Neste caso, as chaves não são apenas senhas, mas arquivos digitais mais complexos. A chave pública pode ficar disponível para qualquer pessoa que queira se comunicar com outra de modo seguro, mas a chave privada deverá ficar em poder apenas de cada titular. É com a chave privada que o destinatário poderá decodificar uma mensagem que foi criptografada para ele com sua respectiva chave pública (OLIVEIRA, 2012). Os principais algoritmos de criptografia assimétrica são: RSA, ElGamal, DiffieHellman e Curvas Elípticas.

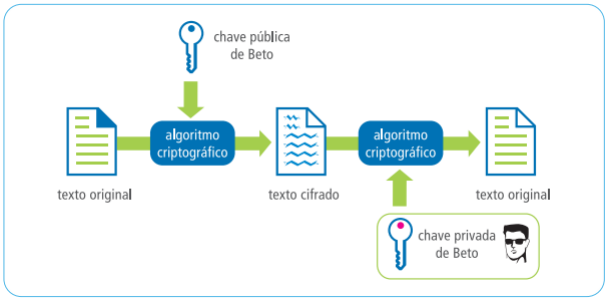

Figura 1 – Representação gráfica da criptografia assimétrica (KOLB, 2015).

2.1.2 Criptografia Simétrica

A criptografia simétrica é um método que utiliza apenas uma chave. Essa chave será utilizada para cifrar e decifrar a mensagem enviada e recebida respectivamente, essa chave deve ser combinada entre o remetente e o destinatário com antecedência. Durante muitos anos foi utilizada a criptografia de chave única em seus métodos mais simples (CORMEN, 2014). Os principais algoritmos de criptografia simétrica são: DES, 3DES, AES, RC4 e IDEA.

A principal vantagem é a simplicidade, esta técnica apresenta facilidade de uso e rapidez para executar os processos criptográficos. Se as chaves utilizadas forem complexas a elaboração de um algoritmo de chave privada se torna bastante fácil, porém as possibilidades de interceptação são correlatas aos recursos empregados, entretanto sua utilização é considerável no processo de proteção da informação, pois quanto mais simples o algoritmo, melhor é a velocidade de processamento e facilidade de implementação.

O principal problema residente na utilização deste sistema de criptografia é que quando a chave de ciframento é a mesma utilizada para deciframento, ou esta última pode facilmente ser obtida a partir do conhecimento da primeira, ambas precisam ser compartilhadas previamente entre origem e destino, antes de se estabelecer o canal criptográfico desejado, e durante o processo de compartilhamento a senha pode ser interceptada, por isso é fundamental utilizar um canal seguro durante o compartilhamento, este independente do destinado à comunicação sigilosa, uma vez que qualquer um que tenha acesso à senha poderá descobrir o conteúdo secreto da mensagem.

- **Cifras de Substituição**

Em criptografia, uma cifra de substituição é um método de criptografia que opera de acordo com um sistema pré-definido de substituição. Para criptografar uma mensagem, unidades do texto - que podem ser letras isoladas, pares ou outros grupos de letras - são substituídas para formar a cifra. As cifras de substituição são decifradas pela substituição inversa.

Uma substituição simples pode ser expressa escrevendo o alfabeto numa ordem diferente, que se designa alfabeto de substituição. Pode ser deslocado de um passo fixo (como na cifra ROT13, exemplo de uma cifra de César) ou baralhado de forma mais complexa(CORMEN, 2014).

É habitual que se escolha uma palavra fácil de lembrar e sem letras repetidas para que se inicie o alfabeto de cifragem por ela, e completando-o com as letras não usadas. Por exemplo, com a chave 'PORTUGAL' teremos os seguintes alfabetos:

- Alfabeto normal: abcdefghijklmnopqrstuvwxyz
- Alfabeto para a cifragem: PORTUGALBCDEFHIJKMNQSVWXYZ

Assim, a mensagem Fujam todos depressa! Fomos descobertos! é cifrada para
GSCPF QITIN TUJMUNNP! GIFIN TUNRIOUMQIN!

Tradicionalmente, o texto cifrado é escrito em blocos de comprimento fixo ("gru-

pos") sem pontuação nem espaços para que não se perceba o comprimento das palavras individuais. O mais comum são grupos de cinco letras, muito usados nos tempos do telégrafo. Assim a mensagem cifrada seria:

GSCPF QITIN TUJMU NNPGI FINTU NRIOU MQIN

Uma desvantagem deste método é que as últimas letras do alfabeto (que geralmente têem menos frequência de uso) tendem a ficar no fim.

- **Cifras de Blocos**

Mas e se a chave for maior que o texto a ser cifrado, ou o texto for maior que a chave? Bom, para isso é preciso utilizar as cifras de blocos. Se um texto for muito longo, a chave de cifração também terá que ser longa, o que pode ser bastante inflexível. Para resolver esse problema existe um sistema de chave simétrica que desmembram o texto em vários blocos menores e aplica a chave em cada um desses blocos, o sistema é conhecido como cifra de bloco (CORMEN, 2014).

A cifra de blocos opera sobre blocos de dados. O texto antes de ser cifrado é dividido em blocos que variam normalmente de 8 a 16 bytes que serão cifrados ou decifrados. Quando o texto não completa o número de bytes de um bloco, este é preenchido com dados conhecidos (geralmente valor zero "0") até completar o número de bytes do bloco, cujo tamanho já é predefinido pelo algoritmo sendo usado (MORENO; PEREIRA; CHIARAMONTE, 2005).

- **Cifra de fluxo**

As Cifras de fluxo, por outro lado, produzem uma sequência pseudo-aleatória de bits que são combinados com a mensagem legível via XOR ou outra operação simples. Assim, não há restrição no tamanho das mensagens que podem ser processadas, nem necessidade de padding ou um modo de operação. Apesar de cifras de fluxo serem potencialmente mais rápidas do que cifras de bloco, a arte de projetar cifras de bloco é atualmente melhor dominada, o que costuma motivar uma mais ampla adoção destas últimas. De fato, quando um comportamento de fluxo é desejado, é comum usar uma cifra de bloco em um modo de operação que emule o comportamento de cifras de fluxo (NIST, 2001) (MARGI et al., 2009).

2.1.3 RC4

O RC4 é uma cifra de fluxo, nesta estrutura a chave é ingressada para um Gerador Pseudo aleatórios por byte que produz um fluxo de 8 bits que são aparentemente aleatórios. A saída do gerador (keystream) é combinada byte por byte com o Plaintext utilizando a operação exclusiva XOR e obtendo o fluxo de byte de Texto Cifrado (PALMA; PEREIRA, 2011).

O algoritmo RC4 encripta uma mensagem utilizando um fluxo de bytes gerados aleatoriamente a partir do tamanho variável de uma chave através do operação lógica

XOR. O RC4 é dividido em duas partes: KSA e PRGA.

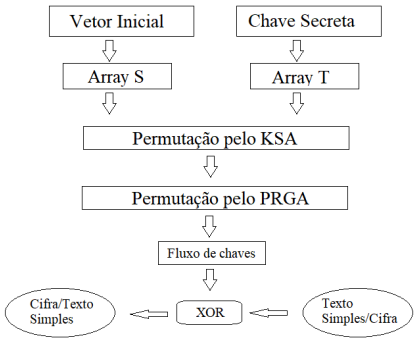

Figura 2 – Representação gráfica do RC4.

No primeiro momento, o algoritmo preenche o vetor S com os valores de 1 a 256, depois ele preenche o vetor T com a chave, o vetor T terá 256 bytes, caso a chave seja menor que 256 o vetor será preenchido com repetições da chave. Apos preencher esse dois vetores o algoritmo embaralha os bytes do vetor S, trocando as posições dos valores de i que sempre será de 1 a 256 e de j que varia de acordo com a operação $j = j + S[i] + T[i]$ mod 256.

Algoritmo 1: KSA

início
  para i *de* 0 *até* $N - 1$ **faça**
    $S[i] = i$
    $T[i] = K[i \text{ mod tam da chave}]$
  fim
  $j = 0$ **para** i *de* 0 *até* $N - 1$ **faça**
    $j = (j + S[i] + T[i]) \text{ mod } 256$
    trocar $S[i], S[j]$
  fim
fim

A segunda etapa do algoritmo, PRGA, consiste em gerar um fluxo de bytes contendo números pseudoaleatórios, que será utilizado para fazer a operação XOR com o

fluxo de bytes do texto claro para produzir o texto cifrado (SCHNEIER, 2007).

Algoritmo 2: PRGA

início
- $i = 0$
- $j = 0$
- **para** i *de* 0 *até* $N - 1$ **faça**
  - $i = (i + 1) \bmod 256$
  - $j = (j + S[i]) \bmod 256$
  - trocar $(S[i], S[j])$
  - $t = (S[i] + S[j]) \bmod 256$
  - $k = S[t]$
- **fim**

fim

A RSADSI (RSA Data Security, Inc) afirma que o algoritmo é imune a diferenças e criptoanálise linear, não parece ter ciclos pequenos, e é altamente não linear. (Não há resultados criptoanalíticos públicos. RC4 pode estar em cerca de 256! x 2562 estados possíveis: um número enorme). A S-box lentamente evolui com o uso: garante que cada elemento muda e garante que os elementos mudam aleatoriamente. O algoritmo é simples o suficiente para que a maioria programadores podem codificá-lo rapidamente a partir da memória (SCHNEIER, 2007).

O algoritmo RC4 trabalha com permutações e pseudo aleatoriedade, o que torna interessante a criptografia da mensagem, já dados pseudoaleatórios dão a impressão de aleatoriedade. Quase todas as operações desse algoritmo não é feito com a mensagem, exceto pela ultima operação, que é um XOR do fluxo aleatório (gerado pelo KSA e PRGA) com a mensagem.

Essa manipulação com a chave do algoritmo é muito importante para a segurança do mesmo. E que vai ser uma operação a mais para garantir a segurança do algoritmo proposto neste trabalho.

2.2 Síntese de Circuitos Reversíveis

2.2.1 Circuitos Reversíveis

Uma função booleana reversível é uma função booleana de saída múltipla com tantas saídas quanto entradas, que é reversível. Uma combinação de circuitos reversíveis é uma combinação de circuitos lógicos acíclicos, no qual todos as portas são reversíveis e estão interconectados sem fan-outs (número de portas de entrada ao qual ela está conectada) explícitos e loops (SAEEDI; MARKOV, 2013).

Considere a função (x, y)ßxy (onde a concatenação denota a operação lógica AND). É impossível fazer que essa função se torne reversível adicionando uma única saída. Uma

das maneiras de torná-la reversível é adicionar uma entrada e duas saídas, assim a função passa a ser como está mostrada na Tabela 1 (MASLOV; DUECK, 2004).

Tabela 1 – Função Reversível e Lógica AND (MASLOV; DUECK, 2004)

x	y	z		x	y	$z \oplus xy$
0	**0**	0		**0**	**0**	0
0	0	1		0	0	1
0	**1**	0		**0**	**1**	0
0	1	1		0	1	1
1	**0**	0		**1**	**0**	0
1	0	1		1	0	1
1	**1**	0		**1**	**1**	0
1	1	1		1	1	1

Uma $n-$porta reversível é uma porta lógica aplicada a uma entrada de n bits e produzem n bits de saída, que realizam uma função bijetiva sobre uma permutação $0, 1, ..., 2^n - 1$. Cada permutação é uma sequência de $n2^n$ bits. A síntese de circuitos reversíveis é o processo que organiza as $n-$portas reversíveis que aplicada a uma permutação obtém outra (TOFFOLI, 1980).

A maioria das portas usadas no design digital não são reversíveis. Das portas comumente usadas, somente a porta NOT é reversível. É necessário um conjunto de portas reversíveis para projetar circuitos reversíveis. Vários dessas portas foram propostos há décadas no passado, entre elas podemos citar a porta NOT, CNOT e Toffoli (MASLOV; DUECK, 2004).

A porta lógica NOT é a mais básica de todas as portas lógicas, possuindo uma única entrada, e sua saída é o complemento dessa entrada. O valor lógico da entrada é invertido na saída. Assim se uma entrada X possui nível lógico igual a 0, então a saída Y terá o nível lógico 1, e se a entrada estiver em nível lógico 1, a saída terá o nível 0. A porta CNOT (também conhecida como controlled-NOT) é uma porta reversível que o segundo bit (o bit alvo) se e somente se o primeiro bit (o bit de controle) for 1 (SAEEDI; SEDIGHI; ZAMANI, 2007). A porta Toffoli é uma operação de três bits que inverte o estado de um bit condicionado ao estado de dois bits de controle (FEDORV et al., 2012).

Uma porta Toffoli generalizada ou porta G-Toffoli C^n NOT $(x_1, x_2, ..., x_n)$ mantém as primeiras $n - 1$ linhas, chamadas de linhas de controles, inalteradas. Esta porta inverte a $n-$ésima linha, chamada de linha alvo, se e somente se, cada linha de controle tem valor igual a 1. Por exemplo, a Figura 2 mostra uma porta $C4$ NOT(a, b, c, d) a linha do topo denota o bit menos significativo. Para $n = 0, 1, 2$ as portas são nomeadas por NOT (ou N), CNOT (ou C), e Toffoli (ou T), respectivamente. Estas três portas compõem a biblioteca CNT (TOFFOLI, 1980), que é um conjunto universal de portas para a computação clássica reversível (RIBEIRO, 2013).

Uma porta Toffoli de controles mistos ou porta C_M-Toffoli C^4_M NOT $(x_1, x_2, ..., x_n)$

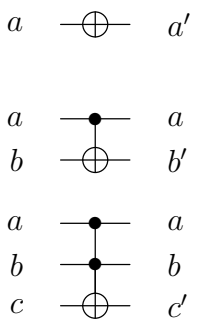

Figura 3 – Circuito representando a biblioteca de portas CNT

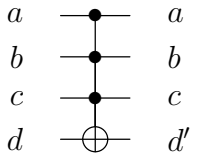

Figura 4 – Porta G-Toffoli representando C^4 NOT (a, b, c, d'). A linha do topo denota o bit menos significativo

mantém as primeiras $n-1$ linhas, chamadas de linhas de controles, inalteradas. Esta porta inverte a n-ésima linha, chamada de linha alvo, se e somente se cada linha de controle positivo (ou negativo) tem valor igual a 1 (ou 0). Indica-se a linha que está com controle negativo colocando 0 após o controle. Veja Figura 3 para um exemplo de uma porta Toffoli de controles mistos 4 x 4 com um padrão negativo-positivo-negativo de linhas de controles e de alvo na última linha, que pode ser denotada por $C^4_M(a', b, c', d)$ (RIBEIRO, 2013).

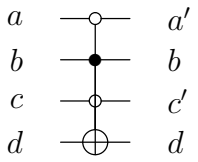

Figura 5 – Porta C_M- Toffoli representada por C^4_M NOT (a', b, c', d)

O número de portas tem sido utilizado na literatura como medida para avaliar as abordagens de sínteses. Para um circuito arbitrário C que consiste numa sequência $p_1, p_2, ..., p_k$ de k portas quânticas, a métrica número de portas é definido como $np(C) \equiv k$. Nós também referimos à noção de custo quântico para medir o custo da implementação dos circuitos quânticos. Mais precisamente, o custo quântico é definido como o número de operações elementares quânticas necessárias para realizar uma porta (RIBEIRO, 2013).

A síntese lógica reversível lida com a geração de um eficiente circuito reversível a partir de uma determinada especificação reversível. A síntese de circuitos reversíveis difere da tradicional irreversíveis no que diz respeito às caracterizações de lógica (ARABZADEH; SAEEDI; ZAMANI, 2010).

Tabela 2 – Custo quântico de portas reversíveis (RIBEIRO, 2013)

tipo de porta	tamanho	lixo	custo quântico
NOT	1	0	1[34]
CNOT	2	0	1[34]
Toffoli	3	0	5[4]
Toffoli com um controle negativo	3	0	5
Toffoli com um controle positivo	3	0	7
G-Toffoli			
$C^nNOT(x_1, x_2, ..., x_n)$	n	0	$2^n - 3$[4]
$C^nNOT(x_1, x_2, ..., x_n)$	n	1	$24n - 88$[4, 31]
$C^nNOT(x_1, x_2, ..., x_n)$	n	$n-3$	$10n - 25$[24,30]
CM-Toffoli			
$C^nNOT(x_1, x_2, ..., x_n)$	n	0	$2^n - 3 + 2m$
$C^nNOT(x_1, x_2, ..., x_n)$	n	1	$24n - 86$
$C^nNOT(x_1, x_2, ..., x_n)$	n	$n-3$	$10n - 23$

A síntese é o passo mais importante durante a construção circuitos complexos. Considerando o design convencional fluxo, a síntese é realizada em várias etapas individuais tais como síntese de alto nível, síntese lógica, mapeamento e roteamento. Para sintetizar lógica reversível, ajustes e extensões são necessárias. Por exemplo, outras tarefas tais como incorporação de funções irreversíveis devem ser adicionadas (MILLER; WILLIE; DUECK, 2009) (MASLOV; DUECK, 2004). Além disso, ao longo de todo o fluxo, as restrições causadas pela reversibilidade e uma biblioteca completamente nova deve ser considerado também (WILLIE, 2011).

3 CRS-MX

A proposta foi desenvolver um algoritmo onde se utilizasse a síntese de circuito reversível como um método de criptografia, onde cada circuito gerado pela síntese passasse pela mensagem que se deseja proteger e assim realizar a criptografia.

Sendo que quando utilizado o mesmo circuito na mensagem criptografada é dada a descriptografia, tendo então a mensagem original.

Esse processo simétrico é representado na Figura 6.

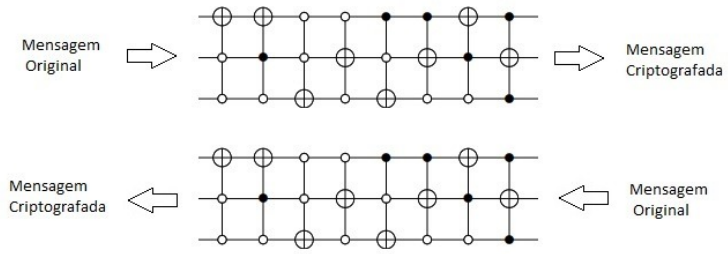

Figura 6 – Novo método de criptografia simétrica com síntese de circuitos reversíveis.

3.1 Hipercubo

A síntese é um processo capaz de gerar os circuitos reversíveis atravéz de portas lógicas também reversíveis. Para realizar essa síntese utilizamos alguns algoritmos que foram desenvolvidos ao longo dos anos, como por exemplo o método MMD, proposto por Maslov e Dueck em 2005 e o método Hipercubo proposto por Ribeiro em 2013.

O método Hipercubo é um método que gera uma síntese de circuito reversível através de aplicações consecutivas da porta C_M-Toffoli em uma permutação. O método utiliza a representação binária dos elementos da permutação - cada elemento é composto de n bits e realiza no máximo n trocas. Estas trocas utilizam as portas C_M-Toffoli para colocar cada bit do elemento da permutação na sua posição correta (RIBEIRO, 2013).

Na tabela 3 é mostrado um breve exemplo da evolução de uma permutação quando aplicada o Hipercubo. Os elementos que serão trocados são apresentados em negrito e os elementos ordenados são apresentados sublinhados. A permutação é lida na ordem da direita para a esquerda, chamamos de ordem direita.

O algoritmo Hipercubo percorre todos os bits da permutação comparando com os bits da permutação identidade, para assim aplicar a porta C_M-Toffoli e trocar o bit que está na posição incorreta.

Tabela 3 – Evolução da permutação π sendo transformada em ι pelo método Hipercubo (RIBEIRO, 2013)

porta aplicada	7	4	1	0	3	2	6	5
	111	100	001	000	011	010	110	101
passo $i = 7$	**111**	100	001	000	011	010	110	**101**
$C^3NOT(a, c, b)$	101	100	001	000	011	010	110	111
passo $i = 6$	101	100	001	000	011	010	110	111
passo $i = 5$	101	100	001	000	**011**	**010**	110	111
$C^3(b, c', a)$	101	100	**001**	000	010	**011**	110	111
$C^3(a, c', b)$	**101**	100	011	000	010	**001**	110	111
$C^3(a, b', c)$	001	100	011	000	010	101	110	111
passo $i = 4$	001	100	011	**000**	**010**	101	110	111
$C^3(a', c', b)$	001	**100**	011	010	**000**	101	110	111
$C^3(a', b', c)$	001	000	011	010	100	101	110	111
passo $i = 3$	001	000	**011**	**010**	100	101	110	111
$C^3(b, c', a)$	001	000	010	011	100	101	110	111
passo $i = 2$	001	000	010	011	100	101	110	111
passo $i = 1$	**001**	**000**	010	011	100	101	110	111
$C^3(b', c', a)$	000	001	010	011	100	101	110	111
	0	1	2	3	4	5	6	7

Algoritmo 3: HIPERCUBO

```
Entrada: π_b vetor
Saída: circuito pilha
início
    N = tamanho do vetor π_b
    n = log_2 N
    para i = N − 1 até 1 faça
        se π_b[i] ≠ ι_b[i] então
            para j = n − 1 até 0 faça
                se π_b[i][j] ≠ ι_b[i][j] então
                    Adicione ao circuito a porta C_M−Toffoli com alvo na
                     posição j e controles nas posições π_b − j coloque
                     C^n NOT(π_b[0], ..., π_b[n − 1], π_b[j]) no circuito troque π_b[i][j]
                fim
            fim
        fim
    fim
fim
```

A permutação utilizada no algoritmo hipercubo vai ser gerada pelo algoritmo KSA, citado no capítulo 2. Assim como no RC4, inicialmente o vetor S vai ser preenchido com os caracteres da tabela ascii, ou seja, de 0 a 255, e a partir de uma chave esse vetor vai ser embaralhado. Inicializar o vetor S é fácil: primeiro, preencha linearmente: $S_0 = 0, S_1 = 1, ..., S_{255} = 255$. Em seguida, preencha outro vetor de 256 bytes com a chave,

repetindo a chave o quanto for necessário para preencher todo vetor: $t_0, t_1, ..., t_{255}$. Defina o índice j como zero. E então $j = (j + s[i] + t[i])$ e troca S[i] e S[j], 256 vezes.

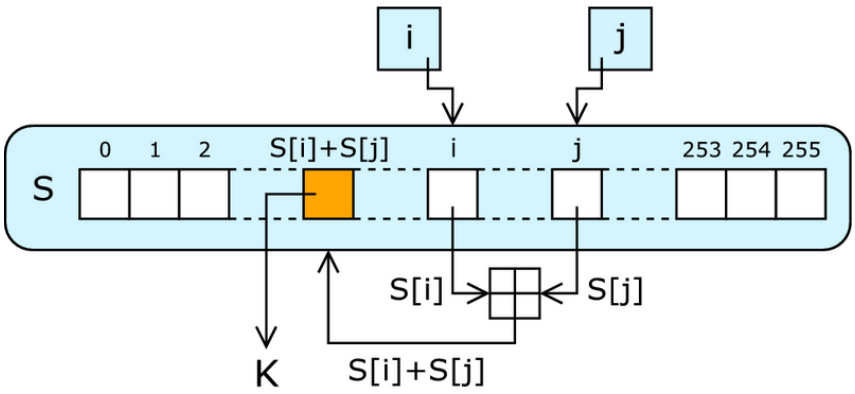

Figura 7 – Esquema de um ciclo do algoritmo KSA.

Independente da chave escolhida, o vetor t sempre terá o tamanho 256, pois o algoritmo KSA a força ter esse tamanho. Então se o usuário digitar uma chave de 5 letras, por exemplo ”tempo”, o vetor da chave vai ser preenchido com a palavra tempo até obter 256 posições.

Tabela 4 – Vetor t sendo preenchido pela chave

t	e	m	p	o	t	e	m	...	...	...	...	p	o
k[0]	k[1]	k[2]	k[3]	k[4]	k[5]	k[6]	k[7]	k[...]	k[...]	k[...]	k[...]	k[254]	k[255]

O hipercubo será aplicado nessa permutação e com isso será gerado um circuitos para cada aplicação da porta C_m-Toffoli. Quando toda a permutação estiver ordenada, o conjunto de circuitos estará pronto para ser utilizado na mensagem, fazendo assim a criptografia.

Algoritmo 4: CRS-MX

Entrada: $chave, mensagem$ string
Saída: $cripto$ string
início
  $tchave$ = tamanho da chave
  $tmen$ = tamanho da mensagem
  $permutao = KSA(chave)$
  C_M-Toffoli = Hipercubo($permutao$)
  $cripto$ = CRS-MX($mensagem, tmen, C_M-$Toffoli)
  retorna $cripto$
fim

3.2 Resultados e Discussões

Um ponto importante em algoritmos de criptografia simétrica é o tamanho dos blocos a serem cifrados, tamanho de chave utilizada e complexidade das operações. O Algoritmo 3 gera $(n-1)*2^n+1$ circuitos no máximo, e isso foi provado por (RIBEIRO, 2013), através do Teorema 3.2.1.

Teorema 3.2.1. *O circuito reversível retornado pelo Algoritmo 3 tem no máximo* $(n-1)*2^n+1$ *portas Toffoli de controles mistos.*

Tabela 5 – Análise do tempo de execução de acordo com o aumento de caracteres a serem criptografados

Número de Caracteres	Tempo de Execução em milissegundos	Tempo de Execução em Segundos
1000	128.827	0,128
10000	819.828	0,819
100000	8213.286	8,213
500000	37505.542	37,505
1000000	74840.499	74,840

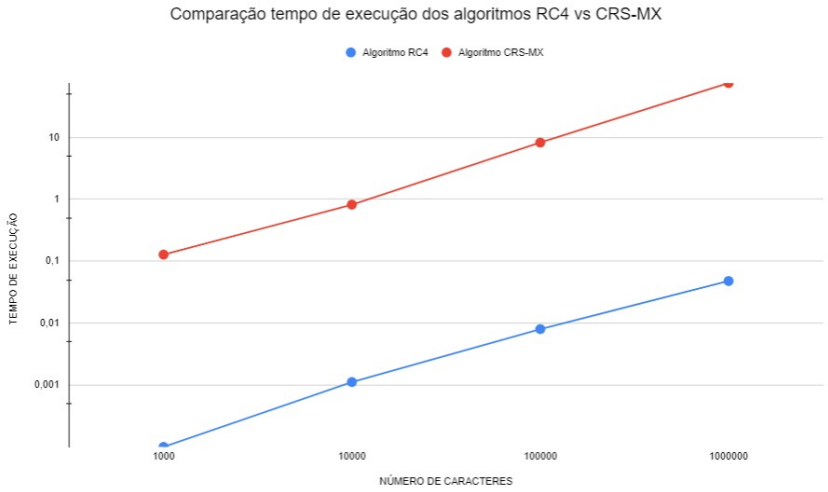

Figura 8 – Gráfico comparando os tempos de execução do RC4 e CRS-MX de acordo com o tamanho da mensagem.

Como podemos observar, o algoritmo se desempenha melhor criptografando uma mensagem com quantidades menores de caracteres. Isso porque os bits se comportam exponencialmente, tendo uma complexidade de $O(T*(n-1)2^n+1)$, onde T = total de letras e n = quantidade de bits.

No intervalo de 1000 a 10000 caracteres, a diferença do tempo de execução é baixa, passando de 0,128 para 0,819 segundos. Isso nos indica que se implementado em sistemas

de troca de mensagens, por exemplo, onde são trocados textos curtos e rápidos o algoritmo poderia significar uma boa eficiência, e a segurança estaria garantida. Em outras situações para criptografia de mensagens pontuais, o CRS-MX pode performar melhor que o RC4, pois o tempo de execução criptografando poucos bits é baixo e a existe mais trocas com a mensagem, logo mais possibilidades de chaves que o RC4.

No grafico demonstrado na Figura 8 pode ser observado que o tempo de execução do CRS-MS aumenta de acordo com o tamanho da mensagem que entra. Uma possível solução para esse tempo elevado seria parallizar o algoritmo, divindo o mesmo em blocos de 256 caracteres (quantidade executada pelo RC4) e ir criptografando esses blocos paralelamente.

4 Conclusão

O algoritmo CRS-MX foi implementado na linguagem C, e apresentou bom desempenho tanto em sua criptografia, quanto em sua descriptografia. Estando sempre dentro do limite superior determinado anteriormente e apresentando uma execução fluida.

A quantidade de circuitos pode ir no máximo até $(n-1)2^n + 1$. Esse resultado depende da permutação gerada no KSA, pois todo o algoritmo é extremamente sensível a essa propriedade já que: a permutação está condicionada à chave inserida, o circuito está condicionado à permutação gerada pelo KSA e a mensagem condicionada ao circuito. A complexidade do algoritmo é igual a $O(T * (n-1)2^n + 1)$, onde T = total de letras e n = quantidade de bits.

4.1 Trabalhos Futuros

Como trabalhos futuros podem ser realizados estudos como:

- Implementação outros métodos de síntese de circuitos reversíveis já existentes, para que venha-se a obter outras variedades de algoritmos utilizando síntese.
- Criptoanálise, afim de verificar se existe algum método que possa quebrar a criptografia do CRS-MX.
- Realizar um estudo mais profundo para poder diminuir a complexidade do algoritmo.

4.2 Considerações Finais

A proposta desse projeto foi satisfatória e cumpriu o objetivo de criar um novo algoritmo de criptografia simétrica utilizando a síntese de circuitos reversíveis, e que pudesse cifrar um bloco de texto maior que o RC4 com uma eficiência aceitável.

Muito se deve a utilização da síntese de circuitos como chave do algoritmo, pois promove a realização de mais operações com a mensagem a ser criptografada, logo mais trocas são realizadas. Isso faz com que essa chave seja muito segura no ponto de vista matemático, justamente por causa das combinações feitas entre permutação, síntese de circuito e mensagem.

Referências

ARABZADEH, M.; SAEEDI, M.; ZAMANI, M. S. Rule-based optimization of reversible circuits. In: IEEE PRESS. **Proceedings of the 2010 Asia and South Pacific Design Automation Conference**. [S.l.], 2010. p. 849–854. Citado na página 12.

BURNETT, S.; PAINE, S. **Criptografia e segurança: o guia oficial RSA**. [S.l.]: Gulf Professional Publishing, 2002. Citado na página 6.

CORMEN, T. **Desmistificando algoritmos**. [S.l.]: Elsevier Brasil, 2014. v. 1. Citado 3 vezes nas páginas 1, 7 e 8.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. **IEEE transactions on Information Theory**, IEEE, v. 22, n. 6, p. 644–654, 1976. Citado na página 2.

FEDORV, A. et al. Implementation of a toffoli gate with superconducting circuits. **Nature**, Nature Publishing Group, v. 481, n. 7380, p. 170, 2012. Citado na página 11.

FIARRESGA, V. M. C. et al. **Criptografia e matemática**. Tese (Doutorado), 2010. Citado na página 4.

KOLB, J. J. **Criptografia Assimétrica**. 2015. Disponível em: <http://jkolb.com.br/criptografia-assimetrica/>. Acesso em: 4 nov. 2015. Citado 2 vezes nas páginas e 6.

LECTURER, A.; SONIPAT, H. A symmetric key cryptographic algorithm. **Hindu College of Engineering**, v. 1, 2010. Citado na página 1.

MALAGUTTI, P. L.; BEZERRA, D. d. J.; RODRIGUES, V. C. d. S. **Aprendendo criptologia de forma divertida**. [S.l.]: Paraıba, 2010. Citado 2 vezes nas páginas 2 e 5.

MARGI, C. et al. Segurança em redes de sensores sem fio. **Minicursos: SBSeg**, p. 149–194, 2009. Citado na página 8.

MASLOV, D.; DUECK, G. W. Reversible cascades with minimal garbage. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 23, n. 11, p. 1497–1509, 2004. Citado 4 vezes nas páginas , 3, 11 e 13.

MILLER, D. M.; WILLIE, R.; DUECK, G. W. Synthesizing reversible circuits for irreversible functions. In: IEEE. **2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools**. [S.l.], 2009. p. 749–756. Citado 2 vezes nas páginas 3 e 13.

MORENO, E. D.; PEREIRA, F. D.; CHIARAMONTE, R. B. Criptografia em software e hardware. **São Paulo: Novatec**, p. 21–42, 2005. Citado 4 vezes nas páginas 1, 2, 6 e 8.

NIST, S. 800-38a. **Recommendation for Block Cipher Modes of Operation**, 2001. Citado na página 8.

OLIVEIRA, R. R. Criptografia simétrica e assimétrica-os principais algoritmos de cifragem. **Segurança Digital [Revista online]**, v. 31, p. 11–15, 2012. Citado 3 vezes nas páginas 2, 4 e 6.

PALMA, S.; PEREIRA, A. **Análise Crítica da Implementação da Cifra RC4 no Protocolo WEP**. [S.l.]: IEEE, 2011. Citado na página 8.

RIBEIRO, A. d. C. **SOBRE GRAFOS DE CAYLEY, PERMUTAC OES E CIRCUITOS REVERSÍVEIS**. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2013. Citado 8 vezes nas páginas , 3, 11, 12, 13, 14, 15 e 17.

SAEEDI, M.; MARKOV, I. L. Synthesis and optimization of reversible circuits—a survey. **ACM Computing Surveys (CSUR)**, ACM, v. 45, n. 2, p. 21, 2013. Citado 2 vezes nas páginas 2 e 10.

SAEEDI, M.; SEDIGHI, M.; ZAMANI, M. S. A novel synthesis algorithm for reversible circuits. In: IEEE. **2007 IEEE/ACM International Conference on Computer-Aided Design**. [S.l.], 2007. p. 65–68. Citado na página 11.

SCHNEIER, B. **Applied cryptography: protocols, algorithms, and source code in C**. [S.l.]: john wiley & sons, 2007. Citado na página 10.

TOFFOLI, T. Reversible computing. In: SPRINGER. **International Colloquium on Automata, Languages, and Programming**. [S.l.], 1980. p. 632–644. Citado 2 vezes nas páginas 2 e 11.

TRINTA, F. A. M.; MACÊDO, R. C. d. Um estudo sobre criptografia e assinatura digital. **Pernambuco: DI/UFPE**, 1998. Citado na página 1.

WILLIE, R. An introduction to reversible circuit design. In: IEEE. **2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)**. [S.l.], 2011. p. 1–4. Citado na página 13.

Apêndices

APÊNDICE A – Implementação do algoritmo CRS-MX para criptografar uma mensagem

Segue o código em C do algoritmo CRS-MX para criptografar uma mensagem.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include <string.h>
#include <locale.h>
#define tam 256
#define lin 1024
#define col 1024

void imprimir_mensagem_descrip(unsigned char *, int);
char ler_mensagem_descrip(unsigned char *);
void imprimir_mensagem_cript(unsigned char *, int);
int ler_mensagem_cript(unsigned char *);
void aplica_circuito_descrip(unsigned char **, unsigned char *, int, int,
void aplica_circuito_cript(unsigned char **, unsigned char *, int, int, i
int hipercubo(int, int, int, unsigned char *, unsigned char **);
int aplica_mx(char *, unsigned char *, int);
void mx_toffoli(unsigned char *, char *, int, int);
void bittochar(unsigned char *, unsigned char *, int);
void chartobit(unsigned char, unsigned char *, int);
void ksa(char *, unsigned char *, int);
unsigned char **Alocar_matriz_real(int, int);
unsigned char **Liberar_matriz_real(int, int, unsigned char **);

int main()
{
    unsigned char permutacao[tam];
    ksa("andre", permutacao, tam);

    int i;
    int N = tam;
    int n = log2(N);
    int limite_superior = ((n - 1) * pow(2, n)) + 1;
```

```
    unsigned char **circuito = Alocar_matriz_real(limite_superior, n);
    int total_circuito;
    total_circuito = hipercubo(N, n, limite_superior, permutacao, circuit

    unsigned char mensagem[lin * col];
    int total_letras;
    mensagem[0] = '\0';

    //criptografar
    total_letras = ler_mensagem_cript(mensagem);
    aplica_circuito_cript(circuito, mensagem, total_circuito, total_letra
    imprimir_mensagem_cript(mensagem, total_letras);

    //descriptografar
    /*total_letras = ler_mensagem_descrip(mensagem);
    aplica_circuito_descrip(circuito, mensagem, total_circuito, total_let
    imprimir_mensagem_descrip(mensagem, total_letras);*/

    Liberar_matriz_real(limite_superior, n, circuito);
    return 0;
}

/********** IMPRESSAO **********/
void imprimir_mensagem_descrip(unsigned char *mensagem, int total_letras)
{
    int i;
    for (i = 0; i < total_letras; i++)
    {

        printf("%c", mensagem[i]);
    }
    printf("\n");
}
/********** FIM DA FUNCAO **********/


/********** LEITURA **********/
char ler_mensagem_descrip(unsigned char *mensagem)
{
```

```
    int i=0;
    int a;
    while (scanf("%d", &a) != EOF)
    {
        mensagem[i] = (char)a;
        i++;
    }
    return i;
}
/********** FIM DA FUNCAO **********/


/************** APLICACAO DO CIRCUITO EM UMA MENSAGEM PARA DESCRIPTOGRAF
void aplica_circuito_descrip(unsigned char **circuito, unsigned char *mer
{

    for (int k = total_circuito -1; k >=0; k--)
    {

        for (int i = 0; i < total_letras; i++)
        {
            unsigned char pimk[n];
            chartobit(mensagem[i], pimk, n);
            //printf("Antes m: %d cir: %s pi: %s\n", mensagem[i], circuit
            int ok = aplica_mx(circuito[k], pimk, n);
            if (ok)
            {
                //printf("depois m: %d cir: %s pi: %s\n", mensagem[i], ci
                bittochar(&mensagem[i], pimk, n);
               // printf("troca: %d pimk: %s\n", mensagem[i], pimk);
            }
        }
    }
}
/********** FIM DA FUNCAO **********/

/********** IMPRESSAO **********/
void imprimir_mensagem_cript(unsigned char *mensagem, int total_letras)
{
```

```
    int i;
    for (i = 0; i < total_letras; i++)
    {

        printf("%d ", mensagem[i]);
    }
    printf("\n");
}
/********** FIM DA FUNCAO **********/


/********** LEITURA **********/
int ler_mensagem_cript(unsigned char *mensagem)
{
    int i=0;
    unsigned char a;
    while (scanf("%c", &a) != EOF)
    {
        mensagem[i] = (char)a;
        i++;
    }
    return i;
}
/********** FIM DA FUNCAO **********/

/*************** APLICACAO DO CIRCUITO EM UMA MENSAGEM PARA CRIPTOGRAFAR*
void aplica_circuito_cript(unsigned char **circuito, unsigned char *mensa
{

    for (int k = 0; k < total_circuito; k++)
    {

        for (int i = 0; i < total_letras; i++)
        {
            unsigned char pimk[n];
            chartobit(mensagem[i], pimk, n);
            //printf("Antes m: %d cir: %s pi: %s\n", mensagem[i], circuit
            int ok = aplica_mx(circuito[k], pimk, n);
            if (ok)
```

```
            {
                //printf("depois m: %d cir: %s pi: %s\n", mensagem[i], ci
                bittochar(&mensagem[i], pimk, n);
                //printf("troca: %d pimk: %s\n", mensagem[i], pimk);
            }
        }
    }
}
/********** FIM DA FUNCAO **********/

/****************** SINTESE DE UM CIRCUITO REVERSIVEL **************/
int hipercubo(int N, int n, int limite_superior, unsigned char *permutaca
{
    int i, j;
    unsigned char id, pi, vid[n], vpi[n], vaux[n];
    int t = 0;

    for (i = N - 1; i > 0; i--)
    {
        id = i;
        pi = permutacao[i];
        chartobit(id, vid, n);
        chartobit(pi, vpi, n);
        if (strcmp(vid, vpi))
        {
            for (j = n - 1; j >= 0; j--)
            {
                if (vid[j] != vpi[j])
                {
                    mx_toffoli(vpi, circuito[t], j, n);
                    aplica_mx(circuito[t], vpi, n);
                    for (int x = N - 1; x >= 0; x--)
                    {
                        unsigned char pimx[n];
                        chartobit(permutacao[x], pimx, n);
                        int ok = aplica_mx(circuito[t], pimx, n);
                        if (ok)
                        {
                            bittochar(&permutacao[x], pimx, n);
```

```
                        }
                    }
                    //printf("id: %s pi: %d\n", circuito[t], permutacao[i
                    t++;
                }
            }
        }
    }
    return t;
}
/********** FIM DA FUNCAO **********/

/************ CM-TOFFOLI SENDO APLICADA NA PERMUTACAO ************/
int aplica_mx(char *mx, unsigned char *letra, int bits)
{
    int ok = 0;
    int controle;
    int x, y, i;
    int c = 1;
    for (i = 0; i < bits; i++)
    {
        if (mx[i] == '2')
        {
            controle = i;
        }
        else if (mx[i] == letra[i])
        {
            c++;
        }
    }
    if (c == bits)
    {
        if (letra[controle] == '0')
        {
            letra[controle] = '1';
        }
        else if (letra[controle] == '1')
        {
            letra[controle] = '0';
```

```
        }
        ok = 1;
    }
    return ok;
}
/********** FIM DA FUNCAO **********/

/********* PORTA CM-TOFOLLI ********/
void mx_toffoli(unsigned char *letra, char *mx, int controle, int bits)
{
    strcpy(mx, letra);
    mx[controle] = '2';
}
/********** FIM DA FUNCAO **********/

/********** CONVERTE UM CHAR E ARMAZENA EM UM VETOR DE BINARIOS *********
void chartobit(unsigned char letra, unsigned char *result, int bits)
{
    int T;
    for (int i = 0; i < bits; i++)
    {
        result[i] = '0';
    }
    for (int j = bits - 1; j >= 0; j--)
    {
        T = letra % 2;
        if (T == 1)
        {
            result[j] = '1';
        }

        letra = letra / 2;
    }
}
/********** FIM DA FUNCAO **********/

/********** CONVERTE UM VETOR DE BINARIOS PARA CHAR **********/
void bittochar(unsigned char *letra, unsigned char *result, int bits)
{
```

```
    int t = 0;
    unsigned char aux[bits];
    int j = 0;
    for (int i = bits - 1; i >= 0; i--)
    {
        if (result[i] == '1')
        {
            aux[i] = 1;
        }
        else
        {
            aux[i] = 0;
        }
        t = t + (aux[i] * pow(2, j));
        j++;
    }
    (*letra) = (unsigned char)t;
}
/********** FIM DA FUNCAO **********/

/********** KSA **********/
void ksa(char *chave, unsigned char *s, int caracter)
{
    int tam_chave = strlen(chave);
    unsigned char t[caracter];
    int i, j;
    for (i = 0; i < caracter; i++)
    {
        s[i] = (char)i;
        t[i] = chave[i % tam_chave];
    }
    j = 0;
    for (i = 0; i < caracter; i++)
    {
        j = (j + s[i] + t[i]) % (caracter);

        char aux;
        aux = s[i];
        s[i] = s[j];
```

```
        s[j] = aux;
    }

    i = 0;
    j = 0;
}
/********** FIM DA FUNCAO **********/

/********** Alocar matriz com ponteiros **********/
unsigned char **Alocar_matriz_real(int m, int n)
{
    unsigned char **v; /* ponteiro para a matriz */
    int i;             /* variavel auxiliar      */
    if (m < 1 || n < 1)
    { /* verifica parametros recebidos */
        printf("* Erro: Parametro invalido *\n");
        return (NULL);
    }
    /* aloca as linhas da matriz */
    v = (unsigned char **)calloc(m, sizeof(char *)); /* Um vetor de m por
    if (v == NULL)
    {
        printf("* Erro: Memoria Insuficiente 1*");
        return (NULL);
    }
    /* aloca as colunas da matriz */
    for (i = 0; i < m; i++)
    {
        v[i] = (unsigned char *)calloc(n, sizeof(unsigned char)); /* m ve
        if (v[i] == NULL)
        {
            printf("* Erro: Memoria Insuficiente 2*");
            return (NULL);
        }
    }
    return v; /* retorna o ponteiro para a matriz */
}
/********** FIM DA FUNCAO **********/
```

```
/********** Liberar matriz com ponteiros **********/
unsigned char **Liberar_matriz_real(int m, int n, unsigned char **v)
{
    int i; /* variavel auxiliar */
    if (v == NULL)
        return (NULL);
    if (m < 1 || n < 1)
    { /* verifica parametros recebidos */
        printf("* Erro: Parametro invalido *\n");
        return (v);
    }
    for (i = 0; i < m; i++)
        free(v[i]); /* libera as linhas da matriz */
    free(v);        /* libera a matriz (vetor de ponteiros) */
    return (NULL);  /* retorna um ponteiro nulo */
}
/********** FIM DA FUNCAO **********/
```