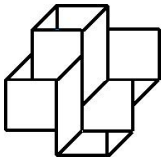

Laboratório Nacional de Computação Científica

Programa de Pós-Graduação em Modelagem Computacional

Análise e Conversão de Algoritmos Criptográficos para Forma Normal Conjuntiva

Natasha do Nascimento Paiva

Petrópolis, RJ - Brasil

Fevereiro de 2017

Natasha do Nascimento Paiva

Análise e Conversão de Algoritmos Criptográficos para Forma Normal Conjuntiva

Dissertação submetida ao corpo docente do Laboratório Nacional de Computação Científica como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências em Modelagem Computacional.

Laboratório Nacional de Computação Científica

Programa de Pós-Graduação em Modelagem Computacional

Orientador: Renato Portugal

Coorientador: Pedro Carlos da Silva Lara

Petrópolis, RJ - Brasil

Fevereiro de 2017

Paiva, Natasha do Nascimento
P149a Análise e Conversão de Algoritmos Criptográficos para Forma Normal Conjuntiva / Natasha do Nascimento Paiva. – Petrópolis, RJ - Brasil, Fevereiro de 2017-
64 p. : il. ; 30 cm.

Orientador(es): Renato Portugal e Pedro Carlos da Silva Lara

Dissertação (M.Sc.) – Laboratório Nacional de Computação Científica
Programa de Pós-Graduação em Modelagem Computacional, Fevereiro de 2017.

1. Criptografia. 2. Satisfatibilidade 3. Sistemas SAT. 4. Segurança da Informação. I. Portugal, Renato. II. LNCC/MCTI. III. Título

CDD: 005.82

Natasha do Nascimento Paiva

Análise e Conversão de Algoritmos Criptográficos para Forma Normal Conjuntiva

Dissertação submetida ao corpo docente do Laboratório Nacional de Computação Científica como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências em Modelagem Computacional.

Aprovada por:

Prof. Renato Portugal, D.Sc.
(Presidente)

Prof. Jauvane Cavalcante de Oliveira, Ph.D.

Prof. Nadia Nedjah, Ph.D.

Petrópolis, RJ - Brasil
Fevereiro de 2017

Agradecimentos

Minha jornada, a fim de obter conhecimentos na pós-graduação, teve início em 2015. Esta dissertação reflete o aprendizado e trabalho que tive na instituição. Algumas pessoas me acompanharam ao longo desta aventura, ao longo destes dois anos, durante o processo de crescimento profissional que obtive no mestrado.

Sou grata a minha família, incluindo a de consideração, pelo apoio e ensinamentos. Sou grata ao professor Luis Rodrigo, pela iniciação ao mundo da pesquisa, por sempre me incentivar a realizar o mestrado, por sempre acreditar.

Agradeço aos amigos de sempre Marcelo, Caio, Rayssa, Ana e Vitor, por mesmo na distância se fazerem presentes, com a amizade incondicional de sempre.

Agradeço aos colegas e amigos do LNCC, dentro destes cito principalmente: Daniel pelas ajudas ao longo das linhas de código e diversas falhas de segmentação, Fortià por ter um dos abraços mais acolhedores que conheço, Leandro pelos sábios ensinamentos acerca da álgebra que envolve este trabalho, Letícia pelo entusiasmo ao longo das horas de descontração, Lucas pela camaradagem e gentileza sempre explícita e Weslley pela amizade e paciência ao longo das conversas duradouras. Aos Thiagos, meus companheiros de sala e café, pois melhor não poderia ser. Ao Tavares, por me acompanhar desde a graduação, pela sua amizade e conselhos valiosos, pela admiração, orgulho e dedicação. Ao Quinelato, por ter se tornado essencial no final desta caminhada, por sempre encorajar com suas sábias e doces palavras, pelas conversas diárias, pelos sorrisos e silêncios confortáveis.

Agradeço aos amigos que encontrei na faculdade e que seguiram me acompanhando ao longo do mestrado, Talita e Vinícius, pela companhia, pelas risadas e por tornarem a trajetória mais suave. Agradeço às meninas da secretaria, Robertinha e Ana Néri, por toda ajuda, risadas, cafés e bolos, conselhos, e principalmente por toda paciência.

Agradeço aos professores que contribuíram com a minha formação, destaco os professores Renato Portugal e Pedro Lara, orientadores deste trabalho e também o professor José Karam, por todo incentivo e aprendizado passados.

Finalmente, ao CNPq pelo apoio financeiro e incentivo.

A todos que fazem parte da minha vida, muito obrigada.

"O Coelho Branco pôs os seus óculos.
- Por onde devo começar, Sua Majestade? - perguntou
- Comece pelo começo - disse o Rei
- e continue até chegar ao fim: então pare."
(Lewis Carroll)

Resumo

Atualmente, além da popularidade da criptografia estar crescendo consideravelmente, existe grande interesse em realizar pesquisas para o desenvolvimento de estratégias que estudem e analisem os métodos criptográficos utilizados atualmente. Isto se dá pelo fato dos métodos atuais serem baseados em problemas matemáticos considerados seguros, entretanto, pesquisas atuais apontam para falhas de segurança na presença de um computador quântico.

Este trabalho consiste em traduzir os algoritmos criptográficos simétricos para o sistema SAT e analisar a saída obtida. Para isto, inicialmente estudamos o modelo do algoritmo criptográfico de interesse e inserimos ele em um software verificador de modelos (CBMC). Este software nos retorna um sistema SAT a ser resolvido, ou seja, reescrevemos o problema como SAT. Além disto, o sistema pode ser interpretado como um grafo, então reduzimos o mesmo através de uma busca em profundidade antes de resolvê-lo, para tornar o problema factível de resolução. A técnica proposta é submetida a um conjunto de testes, utilizando meios propostos pela literatura e alguns meios originais do trabalho, para validar a ferramenta desenvolvida.

Palavras-chave: criptografia. satisfatibilidade. sistemas SAT. segurança da informação.

Abstract

Nowadays, besides the growing popularity of cryptography, there is a big interest from researchers in the development of strategies that study and analyze the current cryptographic methods. This is due to the fact that current methods are based on mathematical problems considered safe, however, researchs points to security flaws in the presence of a quantum computer.

In this work we created and developed a method capable of translating symmetric cryptographic algorithms into Conjunctive Normal Form and then we analyze the obtained output. Initially, we studied the cryptographic algorithm model of interest and used the software CBMC, a bounded model checker software. This software returns a SAT system to be solved, i.e, we rewrite the problem as SAT. In addition, the system can be interpreted as a graph, so we reduce it by implementing a depth-first-search before solving it, to make the problem feasible for resolution. The method proposed is then submitted to a round of tests, aiming to validate the developed tool.

Keywords: cryptography. satisfiability. SAT systems. information security.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

SAT	Satisfatibilidade
FNC	Forma Normal Conjuntiva
CNF	Conjunctive Normal Form
MDC	Máximo Divisor Comum
GF	Galois Field
AES	Advanced Encryption Standard
NIST	National Institute of Standards and Technology
DES	Data Encryption Standard
MPKC	Multivariate Public Key Cryptosystems
BMC	Bounded Model Checking
CBMC	C-Bounded Model Checking
SSA	Single Assignment Form
DFS	Depth-First Search
BCP	Boolean Constraint Propagation
DPLL	Davis-Putnam-Logemann-Loveland
DFA	Differential Fault Analysis

Sumário

1 Introdução

Funções criptográficas estão na base da segurança computacional com cifras de criptografia garantindo a confidencialidade e autenticidade. No geral, a criptografia têm duas características curiosamente correlacionadas: a alta demanda computacional e a relativa simplicidade para desenvolver métodos computacionais para ela [Menezes et al., 1996]. Assim, motivada pelo constante aumento no poder computacional, tais técnicas têm sido amplamente estudadas e utilizadas para melhorar a qualidade das soluções. Uma consequência imediata destes aspectos é, por exemplo, a maior aplicabilidade destas técnicas em problemas cada vez maiores e complexos [Whitman and Mattord, 2011]. Em especial quando é aplicada à segurança da informação e é notório que a segurança da informação é uma área com grande potencial de pesquisa e desenvolvimento.

A criptografia é um ramo da criptologia, uma área que engloba também os estudos de criptoanálise. O objetivo principal da criptografia é ocultar informações, enquanto isto, o objetivo da criptoanálise é quebrar técnicas e sistemas utilizados na criptografia a fim de obter informações geradas pelas codificações.

Podemos dividir os algoritmos criptográficos em duas classes: clássicos e pós-quânticos. A criptografia clássica é formada por algoritmos baseados em problemas matemáticos que os computadores clássicos não são capazes de lidar de forma eficiente. Podemos citar como exemplo o AES, Simon/Speck e o RSA. Entretanto, estudos mostram [Shor, 1999, Costa, 2008] o quanto os algoritmos criptográficos utilizados atualmente tornam-se vulneráveis na presença do computador quântico. Com esta motivação, começaram a desenvolver os chamados Sistemas Criptográficos Pós-Quânticos, como visto em Barreto et al. [2013], que são algoritmos criptográficos baseados em problemas matemáticos e computacionais que não sofrem com a ameaça do Algoritmo de Shor Shor [1999]. Sistemas pós-quânticos conhecidos são os baseados em códigos corretores de erro [McEliece, 1978, Niederreiter, 1986] e sistemas multivariados quadráticos ($\mathcal{MQ}$) [Ding and Schmidt, 2005, Kipnis et al., 1999].

Satisfatibilidade é um conceito elementar de semântica. Uma fórmula é dita satisfazível se é possível encontrar uma interpretação, uma valoração para as variáveis que a compõe, que a torne verdadeira. Ou seja, SAT é um problema de decisão, é necessário estabelecer se há uma interpretação que satisfaça uma dada fórmula booleana de entrada.

Agentes de resolução *SAT* têm demonstrado ser uma ferramenta poderosa para testar hipóteses matemáticas. Neste trabalho, estendemos solucionadores *SAT* para melhor trabalhar no ambiente da criptografia e criptoanálise. Trabalhos anteriores sobre a resolução de problemas de criptografia com solucionadores *SAT* tem se concentrado sobre a melhor

representação matemática de cifras [Bard, 2008].

Dado o cenário complexo discutido, este trabalho tem por objetivo demonstrar a descoberta de relações existentes relevantes entre fatores distintos que podem ser cruciais na recuperação de chaves criptográficas dos algoritmos. No entanto, ao invés de confiar em técnicas tradicionais tentadas antes, vamos empregar um novo paradigma, baseado na união de duas técnicas: a análise automática de software, que diz se um modelo atende a uma especificação e o sistema *SAT* [Soos et al., 2009]. O objetivo principal deste trabalho é realizar uma rápida conversão dos algoritmos criptográficos para a forma normal conjuntiva, gerando assim um sistema SAT. Dentre os objetivos secundários estão a implementação de um solucionador de sistemas eficaz e a comparação dos resultados com os presentes na literatura.

Para isto, realizamos um estudo aprofundado no capítulo 2 acerca dos princípios algébricos necessários para um melhor entendimento dos conceitos de criptografia, além de apresentar o estado da arte. Após tais conceitos serem introduzidos, apresentamos o conceito de satisfatibilidade, que foi mostrado por Cook [1971] ser o primeiro problema provado NP-Completo, no capítulo 3. Além disto, realizamos um estudo sobre a análise automática de software, bem como o software utilizado como base para a implementação desta dissertação. Apresentamos também o método proposto ao longo do trabalho. O capítulo 4 lista e analisa todas as simulações numéricas realizadas para validar o método proposto. Finalmente, resumimos as conclusões do trabalho como um todo e realizamos uma breve perspectiva sobre as possíveis pesquisas futuras no capítulo 5.

2 Criptografia

Presente praticamente no dia a dia de todos, a criptografia é objeto constante de estudo, no que diz respeito à evolução de métodos já conhecidos, a sugestões de novas técnicas e a aplicações práticas. No passado, ficou famosa por ser utilizada em assuntos ligados diretamente às guerras e à diplomacia. Atualmente, a maioria dos métodos depende da computação e esta dependência implica que é importante garantir a sua integridade. Tolerância a falhas é a área computacional que se preocupa com a integridade dos sistemas contra falhas naturais, ou seja, falhas não intencionais. Já falhas maliciosas, arquitetadas propositalmente, são tratadas pela segurança de sistemas [Weber, 2003]. Porém, o alto nível de conectividade atual faz com que os sistemas computacionais sejam extremamente complexos, portanto, prover segurança neste ambiente é desafiador. Neste sentido, aparece a ciência da criptografia.

Segundo Fiarresga et al. [2010], a criptografia possui quatro objetivos principais:

- Confidencialidade: só o destinatário deve ser capaz de extrair o conteúdo da informação;
- Integridade: proteger a informação contra alteração não autorizada;
- Autenticação: o destinatário deve ser capaz de verificar se o remetente é quem ele diz ser;
- Não-repúdio: o remetente não pode negar que enviou a informação.

É conveniente estabelecer uma terminologia básica, no que diz respeito a esta dissertação.

- **Texto plano**: também chamado de texto puro, qualquer dado a ser criptografado, mesmo que este dado em si não represente um texto propriamente dito. Considerado como uma sequência de n bits;
- **Texto cifrado**: consiste do resultado de um texto plano após passar por um processo criptográfico;
- **Chave**: parâmetro que determina a saída funcional de um algoritmo criptográfico, responsável por especificar a transformação de texto plano para texto cifrado e vice-versa.

- **Sistema de criptografia**: consiste de um ou mais algoritmos criptográficos utilizados para encriptar/decriptar um texto plano, através de uma chave, transformando-o em texto cifrado e vice-versa.

Segundo Shannon [1948], há duas características que qualquer sistema de criptografia bem projetado deve possuir: confusão e difusão. A primeira propriedade, confusão, assegura que o texto cifrado é estatisticamente independente do texto puro e da chave. Isto se deve ao fato de uma parta da informação ser alterada, então os bits de saída não possuem relação óbvia com os bits de entrada. Ou seja, confusão torna a relação entre criptograma e chave tão complexa quanto possível. Já a segunda propriedade, difusão, indica que uma pequena perturbação no texto puro ou na chave provocaria uma grande mudança no texto cifrado. Esta característica procura propagar os efeitos de um bit de texto simples para outros bits no texto cifrado.

Os algoritmos criptográficos estão dentre as técnicas mais importantes quando voltadas para tal contexto. Eles fornecem uma forma confiável de confidencialidade de dados, baseados em técnicas matemáticas. Os métodos criptográficos podem ser divididos em duas categorias: criptografia de chave privada, também conhecida como criptografia de chave simétrica, e criptografia de chave pública, chamados de algoritmos de chave assimétrica.

2.1 Conceitos Algébricos

Nesta seção iremos introduzir alguns conceitos matemáticos necessários para a melhor compreensão deste capítulo.

2.1.1 Aritmética Modular

Na matemática, aritmética modular é um sistema voltado para o conjunto dos inteiros $\mathbb{Z}$, onde os números voltam para trás após atingir um certo valor, o módulo.

Esta aritmética modular está relacionada com fenômenos que se repetem após um certo período fixo, fenômenos cíclicos ou periódicos. Quando se está trabalhando com horas, este período é igual a 24, por exemplo. Caso um evento tenha ocorrido 20 horas após o meio dia de um certo dia, ele terá ocorrido às 8 horas da manhã do dia seguinte, visto que $12 + 20 = 32 = 32$ módulo $24 = 8$.

Vamos definir uma relação de equivalência em $\mathbb{Z}$, que é a relação de congruência módulo n.

Definição 1. *Seja n um inteiro não negativo e sejam $a, b \in \mathbb{Z}$. Dizemos que a é congruente a b módulo n se a diferença $(a-b)$ for múltiplo inteiro de n. Esta definição pode ser escrita como:*

$$a \equiv b \ (mod \ n) \iff \exists k \in \mathbb{Z} \mid (a-b) = kn$$

2.1.2 Algoritmo de Euclides

Um passo muito utilizado no processo de criptografia se dá por meio do Algoritmo Estendido de Euclides, o qual tem por objetivo encontrar o máximo divisor comum (mdc) de dois números inteiros ou polinômios.

Sendo a e b inteiros diferentes de zero, nós podemos encontrar o $\text{mdc}(a, b)$ através da construção das seguintes equações:

$$\begin{aligned} a &= bq_1 + r_1 \\ b &= r_1q_2 + r_2 \\ r_1 &= r_2q_3 + r_3 \\ &\cdot \\ &\cdot \\ &\cdot \\ r_{n-2} &= r_{n-1}q_n + r_n \\ r_{n-1} &= r_nq_{n+1} + 0 \end{aligned}$$

Assim, $\text{mdc}(a, b) = r_n$. Em outras aplicações da área, precisaremos determinar não só o máximo divisor comum de dois inteiros mas também os valores u e v que satisfazem $\text{mdc}(a, b) = au + bv$. Utilizamos o Algoritmo de Euclides para esta finalidade também. Utilizando os valores de q_i e r_j encontrados nas equações acima, iremos montar a 1 baseada no algoritmo.

As entradas de cada linha $i = 1, 2, \ldots, n$ da Tabela 1 são construídos como se segue. Os q_i, r_i são da i-ésima equação

$$r_{i-2} = r_{i-1}q_i + r_i \tag{2.1}$$

na lista de equações. Observe que se resolvermos r_i em 2.1, obtemos a seguinte equação.

$$r_i = r_{i-2} - r_{i-1}q_i \tag{2.2}$$

Tabela 1: Tabela baseada no Algoritmo de Euclides.

n	**q**	**r**	**u**	**v**
-1	-	$r_{-1} = a$	$u_{-1} = 1$	$v_{-1} = 0$
0	-	$r_0 = b$	$u_0 = 0$	$v_0 = 1$
1	q_1	r_1	u_1	v_1
2	q_2	r_2	u_2	v_2
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
n	q_n	r_n		v_n

Assim, podemos construir u_i e v_i de forma análoga, seguindo este padrão para construir r_i através de q_i.

$$u_i = u_{i-2} - u_{i-1}q_i \tag{2.3}$$

$$v_i = v_{i-2} - v_{i-1}q_i \tag{2.4}$$

Para exemplificar o algoritmo explicado acima, vamos tomar que $a = 81$ e $b = 64$. Inicialmente vamos formar as seguintes equações:

$$\begin{aligned} 81 &= 64 \cdot 1 + 17 \\ 64 &= 17 \cdot 3 + 13 \\ 17 &= 13 \cdot 1 + 4 \\ 13 &= 4 \cdot 3 + 1 \qquad \rightarrow r_n \\ 4 &= 1 \cdot 4 + 0 \end{aligned} \tag{2.5}$$

Então, $\text{mdc}(81, 64) = 1$. Além disso, podemos verificar facilmente que estas equações produzem a tabela 2:

Tabela 2: Tabela obtida para $a = 81, b = 64$ através do Algoritmo de Euclides.

n	**q**	**r**	**u**	**v**
-1	-	84	1	0
0	-	61	0	1
1	1	17	1	-1
2	3	13	-3	4
3	1	4	4	-5
4	3	1	-15	19

Dessa forma, baseado nas equações 2.5 e na tabela 2, temos que $u = -15$ e $v = 19$ satisfazem a $\text{mdc}(81, 64) = 81u + 64v$.

2.1.3 Corpos de Galois

O corpo de Galois é amplamente utilizado na criptografia, e por isso iremos discutilo nesta seção. É um corpo finito e leva o nome de Corpo de Galois em honra ao matemático francês Évariste Galois, responsável pela criação da Teoria dos Grupos e Teoria de Galois, dois importantes ramos da álgebra abstrata.

Para melhor entendimento do Corpo de Galois, vamos começar introduzindo o conceito de grupos e anéis.

Definição 2. *Seja G um conjunto não vazio onde está definida uma operação entre pares de G, denotada por $* : G \times G \to G$. Dizemos que o par $(G, *)$ é um grupo se as seguintes propriedades são válidas.*

- Associatividade: $a * (b * c) = (a * b) * c \ \forall a, b, c \in G$;
- Existência do elemento neutro: $\exists e \in G$ tal que $a * e = e * a$, $\forall a \in G$;
- Existência do elemento inverso: $\forall a \in G$, $\exists a^{-1} \in G$ tal que $a * a^{-1} = a^{-1} * a = e$.

Grupos que satisfazem a propriedade abaixo são chamados de *grupos Abelianos.*

- Comutatividade: $\forall g_1, g_2 \in G$, $g_1 * g_2 = g_2 * g_1$.

Definição 3. *Seja A um conjunto não vazio onde estejam definidas duas operações, as quais chamaremos de adição e multiplicação em A de denotaremos por $+$ e $\cdot$. Assim:*

$$\begin{aligned} + : A \times A &\to A & \qquad \cdot : A \times A &\to A \\ (a, b) &\to a + b & (a, b) &\to a \cdot b \end{aligned}$$

Chamaremos $(A, +, \times)$ de anel se as seguintes propriedades forem satisfeitas:

- Associatividade da adição: $(a + b) + c = a + (b + c)$
- Existência de elemento neutro: $\exists 0 \in A$ tal que $a + 0 = 0 + a = a$
- Existência do inverso aditivo: $\forall x \in A$ existe um $-x \in A$ tal que $x + (-x) = (-x) + x = 0$
- Comutatividade da adição: $a + b = b + a$
- Associatividade da multiplicação: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- Distributividade à esquerda e à direita: $a \cdot (b + c) = a \cdot b + a \cdot c$; $(a + b) \cdot c = a \cdot c + b \cdot c$

Além disto, se um anel possuir as duas propriedades abaixo, ele receberá nomes diferenciados:

- Anel comutativo: $x \cdot y = y \cdot x$, $\forall x, y \in A$.
- Anel com unidade: $\exists 1 \in A$, tal que $x \cdot 1 = 1 \cdot x \forall x \in A$

Caso A seja um anel comutativo com unidade, ele receberá o nome de Domínio de Integridade. Por fim, se um Domínio de Integridade possuir a propriedade do inverso multiplicativo, ele será um corpo.

- $\forall x \in A, x \neq 0, \exists y \in A \mid x \cdot y = y \cdot x = 1$.

O chamado Corpo de Galois, denotado por $GF(p^m)$, tem se mostrado extremamente útil em teoria dos códigos, isto devido as suas propriedades. É um corpo que contém um número finito de elementos. Podemos destacar o seguinte teorema 1:

Teorema 1. *Se r é um inteiro positivo, então existe um corpo $\mathbb{K}$ tal que $\#\mathbb{K} = r$, onde $\#$ denota a cardinalidade do corpo, se e somente se $r = p^m$, para algum inteiro $m \geq 1$ e sendo p um número primo.*

O teorema 1 nos diz claramente que a ordem de um corpo finito é sempre um número primo ou uma potência de primo. Quando $m > 1$, é possível representar os elementos de $GF(p^m)$ como polinômios de grau menor que m, onde os coeficientes pertencem a $GF(p)$. O corpo mais utilizado em criptografia é o $GF(2^m)$, portanto os coeficientes dos polinômios pertencem ao conjunto $\mathbb{Z}_2 = \{0, 1\}$. Os elementos de $GF(2^m)$ podem ser vistos como polinômios do tipo:

$$f(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1x + a_0$$

onde $a_i \in \{0, 1\}$.

Um grupo cíclico é aquele que pode ser gerado através de um único elemento. Visto que os elementos de $GF(2^m)$ formam um grupo cíclico, existe um elemento gerador neste grupo. Este elemento gerador é conhecido por polinômio primitivo, que é um polinômio que não pode ser fatorado por outros polinômios de grau menor ou igual ao dele. Podemos obter todos os elementos de $GF(2^m)$ a partir das potências do polinômio primitivo. Como exemplo, iremos listar os elementos de $GF(2^3)$:

$$GF(2^3) = (0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1)$$

onde o polinômio primitivo é x e a operação produto é a multiplicação usual de polinômios módulo $x^3 + x + 1$.

2.1.4 Congruência

Definição 4. *Se um inteiro m, não nulo, divide a diferença $a - b$, onde a e b são inteiros, dizemos que a é congruente a b módulo m, e denotamos por $a \equiv b \pmod{m}$. Se $a - b$ não é divisível por m, dizemos que a não é congruente a b módulo m, e denotamos por $a \not\equiv b \pmod{m}$.*

A seguir seguem algumas propriedades de congruências de números inteiros. Sejam a, b, c, d, m inteiros, então:

1. $a \equiv b \pmod{m}$ e $a - b \equiv 0 \pmod{m}$ são proposições equivalentes;
2. Se $a \equiv b \pmod{m}$ e $b \equiv c \pmod{m}$, então $a \equiv c \pmod{m}$;
3. $a \equiv a \pmod{m}$;
4. Se $a \equiv b \pmod{m}$ e $c \equiv d \pmod{m}$, então $a + c \equiv b + d \pmod{m}$;
5. Se $a \equiv b \pmod{m}$, então $ac \equiv bc \pmod{m}$;
6. Se $ac \equiv bc \pmod{m}$, então $a \equiv b \pmod{m/d}$, onde $d = (c, m)$. Essa propriedade é conhecida como Lei do Cancelamento.

2.1.5 Pequeno Teorema de Fermat

Teorema 2 (Pequeno Teorema de Fermat)**.** *Seja p um número primo. Se p não divide a, então $a^{p-1} \equiv 1 \pmod{p}$.*

Seja o conjunto de valores $a, 2a, 3a, \ldots, (p-1)a$. Sabemos que, pelo fato de p não divide a, $(a, p) = 1$, portanto nenhum dos números deste conjunto é divisível por p. Além disso, temos que, se $aj \equiv ak \pmod{p}$, então $j \equiv k \pmod{p}$, ou seja, todos eles são incongruentes módulo p. Portanto, podemos estabelecer uma relação biunívoca entre os aj, $j = 1, 2, \ldots, (p-1)$ e o conjunto $1, 2, 3, \ldots, (p-1)$, em termos de congruência, isto é, cada um dos termos do primeiro conjunto é congruente a um diferente do segundo conjunto. Deste argumento e da propriedade 5 de congruência, temos que:

$$a(2a)(3a)\ldots(p-1)a \equiv 1.2.3\ldots(p-1) \pmod{p} \tag{2.6}$$

ou seja

$$a^{p-1}(p-1)! \equiv (p-1)! \pmod{p} \tag{2.7}$$

por fim, temos que:

$$a^{p-1} \equiv 1 \ (mod \ p) \tag{2.8}$$

2.1.6 Teorema de Euler

O teorema de Euler visa generalizar o Pequeno Teorema de Fermat para quaisquer números inteiros, utilizando, para isso, a função ϕ de Euler.

Definição 5 (Função ϕ de Euler)**.** *Para qualquer inteiro positivo n, definimos $\phi(n)$ como o número de inteiros positivos menores que n e coprimos de n.*

Teorema 3 (Teorema de Euler)**.** *Se m é um número inteiro positivo e a é um inteiro com $mdc(a, m) = 1$, então $a^{\phi(m)} \equiv 1 \ (mod \ m)$.*

Para demonstrar o Teorema de Euler, vamos seguir no mesmo pensamento utilizado no Pequeno Teorema de Fermat. Se $r_1, r_2, \ldots, r_{\phi(m)}$ englobam todos os restos não-nulos possíveis na divisão por m, então $ar_1, ar_2, \ldots, ar_{\phi(m)}$ também o faz, com $\text{mdc}(a, m) = 1$. Ou seja, podemos fazer uma relação biunívoca entre os dois conjuntos, em congruência, e chegamos em:

$$ar_1 ar_2 \ldots ar_{\phi(m)} \equiv r_1 r_2 \ldots, r_{\phi(m)} \ (mod \ m) \tag{2.9}$$

ou seja

$$a^{\phi(m)} r_1 r_2 \ldots r_{\phi(m)} \equiv r_1 r_2 \ldots, r_{\phi(m)} \ (mod \ m) \tag{2.10}$$

por fim, temos que:

$$a^{\phi(m)} \equiv 1 \ (mod \ m) \tag{2.11}$$

Após introduzidos os conceitos listados nesta seção, podemos prosseguir com os conceitos de criptografia.

2.2 Criptografia de chave simétrica

Algoritmos de chave simétrica utilizam apenas uma chave no processo de encriptação e decriptação do texto. Ou seja, dado um texto plano, o mesmo é criptografado para um texto cifrado utilizando uma chave. Para descobrir o texto plano a partir do texto cifrado, basta realizar o caminho inverso utilizando a mesma chave. O processo torna-se mais simples pelo fato do processo envolver apenas uma chave criptográfica. Na prática, a

chave representa um segredo partilhado por duas partes, que é utilizada para estabelecer um canal de informações confidencial. Usa-se uma só chave na premissa de que esta é conhecida apenas pelas partes envolvidas.

2.2.1 AES

O AES (*Advanced Encryption Standard*), também chamado de *Rijndael*, se originou de um concurso lançado pelo *National Institute of Standards and Technology - NIST* em 1997 e substitui o DES (*Data Encryption Standard*), que vinha apresentando falhas. Em 2000, o algoritmo criado por Vincent Rijmen e Joan Daemen foi escolhido devido as suas qualidades.

O AES trabalha sobre uma matriz de bytes, também chamada de estado, que contém a mensagem, e após cada rodada será modificada. A matriz possui 4 linhas e N_b colunas, onde N_b é número de bits do bloco dividido por 32. O algoritmo pode trabalhar com tamanhos de chave 128, 192 ou 256 bits [Stallings and Vieira, 2008].

Como visto em Daemen and Rijmen [1999], o algoritmo trabalha em rodadas, e cada rodada é composta por 4 etapas: *AddRoundKey*, *SubBytes*, *ShiftRows*, *MixColumns*, na última rodada, a última etapa não é realizada. Na figura 1, podemos ver o esquema do algoritmo AES.

Cada iteração altera o bloco de 128 bits para outro bloco de mesmo tamanho, e cada byte da matriz estado é afetado pelas seguintes transformações:

- *AddRoundKey* - Nesta fase, cada byte do estado é combinado com a sub-chave própria do turno; cada sub-chave é derivada da chave principal usando o algoritmo de escalonamento do Rijndael.

- *SubBytes* - Etapa de substituição não-linear onde cada byte é substituído por outro de acordo com uma tabela de referência. Cada byte na matriz é atualizado utilizando uma S-Box[1].

- *ShiftRows* - Etapa de transposição onde cada fileira do estado é deslocada de um determinado de número de posições. No AES, a primeira linha fica inalterada. Cada byte da segunda linha é deslocado à esquerda de uma posição. Similarmente, a terceira e quarta fileiras são deslocadas de duas e de três posições, respectivamente.

- *MixColumns* - Etapa de mescla que opera nas colunas do estado e combina os quatro bytes de cada coluna usando uma transformação linear. Junto com a etapa anterior,

[1] S-BOX: Funções que recebe um conjunto de bits, reordena-os conforme uma ordem específica e os envia como saída - em suma, realizam a técnica de substituição, conforme alguma tabela ou função.

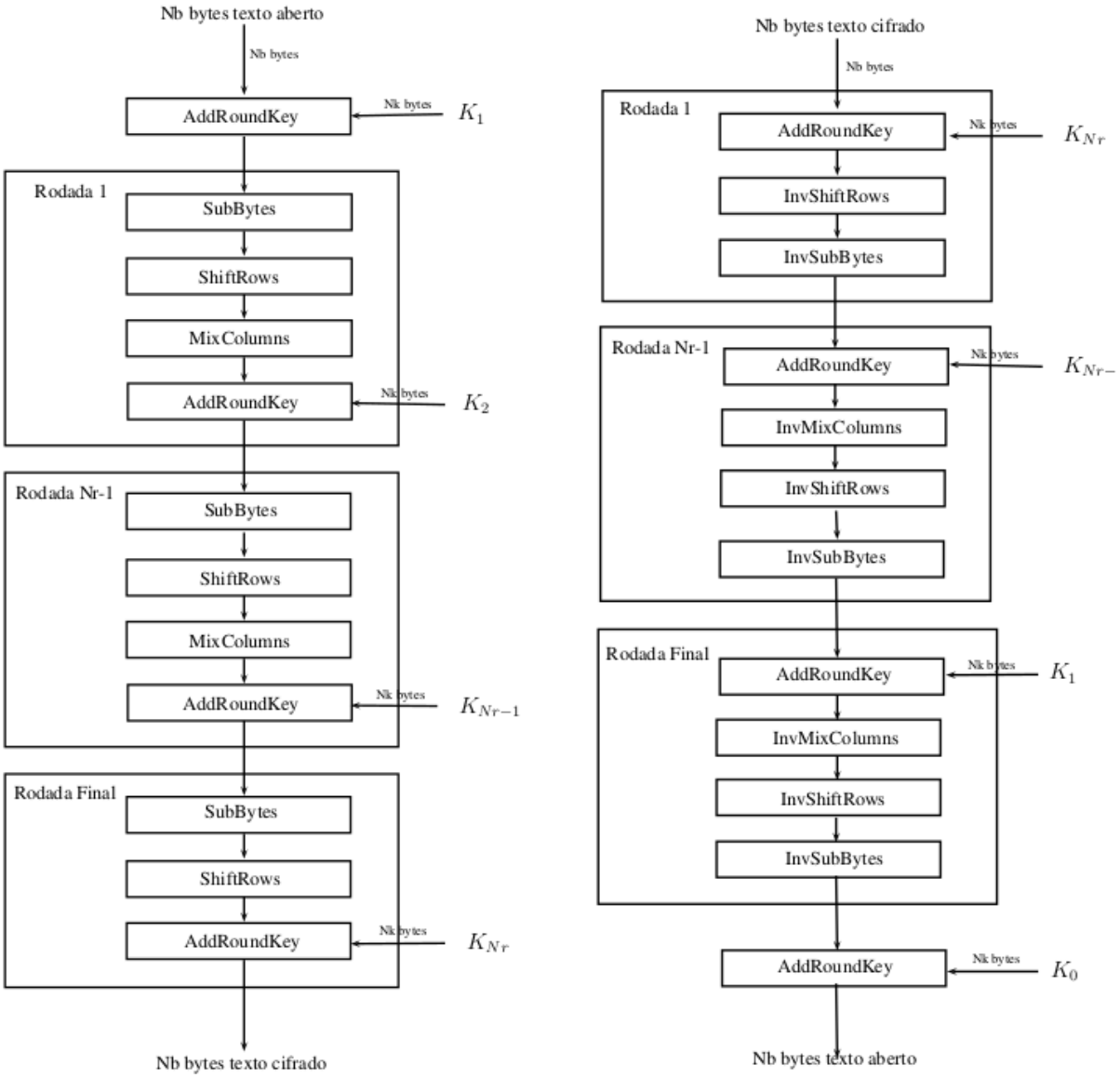

Figura 1: Funcionamento do AES.

esta etapa fornece difusão à cifra. Cada coluna é tratada como um polinômio com coeficientes em GF(2^8) e é então multiplicado em módulo $x^4 + 1$ pelo polinômio fixo $c(x) = 3x^3 + x^2 + x + 2$. Esta etapa pode ser vista também como uma multiplicação matricial no corpo finito de Rijndael.

2.2.1.1 Mini-AES

Mini-AES [Phan, 2002] é uma cifra baseada no AES, mais simples porém a estrutura matemática é similar. Foi proposta com o intuito educacional, para auxiliar no processo criptográfico e de criptoanálise, para um melhor entendimento dos conceitos por trás do AES.

2.3 Criptografia de chave assimétrica

Como mostrado anteriormente, algoritmos de chave simétrica utilizam a mesma chave criptográfica tanto para encriptação quanto para a decriptação dos textos, o que implica das duas partes compartilharem a chave. Tal requisito de ambas as partes possuírem acesso à mesma chave secreta é uma desvantagem da criptografia de chave simétrica. Visando contornar esta desvantagem, Whitfield Diffie e Martin Hellman apresentam o conceito de chave pública em Diffie and Hellman [1976], onde cada parte possui duas chaves: uma pública, que pode ser divulgada, e outra privada, que deve ser mantida em sigilo pelo seu dono. Desta forma, criptografia de chave assimétrica, também chamada de criptografia de chave pública, é uma classe de protocolos de criptografia baseados em algoritmos que requerem duas chaves.

Os sistemas de criptografia assimétrica são baseados em problemas matemáticos que não admitem solução eficiente e são inerentes em determinados relacionamentos como logaritmo discreto, curvas elípticas e fatoração de números inteiros em seus componentes primos. Computacionalmente falando, é fácil gerar um par de chaves, uma pública e uma privada, e utilizá-lo na encriptação e decriptação de dados. A dificuldade está em determinar uma chave privada através de sua chave pública, por isto esta pode ser publicada sem comprometer a segurança do sistema.

2.3.1 RSA

Em 1977, Ron Rivest, Adir Shamir e Len Adleman desenvolveram um método, em Rivest et al. [1978], utilizando o conceito de chave assimétrica proposto por Diffie e Hellman. Desde então, o RSA tem sido amplamente utilizado. Como podemos ver em Costa [2014], o RSA é uma cifra onde ambos os textos, claro e cifrado, são interpretados através de números inteiros entre 0 e $n-1$, onde n representa o tamanho da chave utilizada.

Antes de introduzir o sistema RSA formalmente, vamos considerar um exemplo simples da matemática que está por trás desta cifra. Escolhendo dois números primos $p = 11$ e $q = 23$, e tomando que $n = pq = 253$ e $m = (p-1)(q-1) = 220$. Em seguida, escolhe um a tal que $\text{mdc}(a, m) = 1$, sendo $a = 7$ e um b tal que $ab = 1 \bmod m$, utilizando o Algoritmo de Euclides, sabemos que $b = 63$. Então, para $x = 2$, note que:

$$
\begin{aligned}
(x^a)^b &= (2^7)^{63} \\
&= 567842753355942883241659224912503542463782313036967234594914218109 \\
&\quad 87444383859212759858675837012778559434572000489545151057390752235 52 \\
&= 2 \mod 253 \\
&= x \mod n
\end{aligned}
$$

Podemos notar que:

$$x^{ab} = x \ mod \ n \tag{2.12}$$

E de fato esta equação será verdadeira para todo $x \in Z$, visto que a e b foram escolhidos de forma que:

$$ab = 1 \ mod \ m \tag{2.13}$$

Então, se cifrarmos uma mensagem elevando o texto plano a potência a, nós podemos decifrar a mensagem elevando o texto a potência b e reduzindo ao módulo n.

Por simplicidade, assumiremos que todas as mensagens são escritas no alfabeto $L = \{A, B, C, ..., Z\}$, além disso assumiremos que $R = Z_{26}$ e que $\alpha : L \to R$ é dado por $\alpha(A) = 0$, $\alpha(B) = 1$, ..., $\alpha(Z) = 25$. As correspondências de α são:

Tabela 3: Correspondências de α

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Para cifrar uma mensagem utilizado o RSA, primeiramente devemos converter a mensagem em uma lista de inteiros, aplicando um mapeamento como mostrado na tabela 3. Após isto, podemos seguir os passos:

1. Escolher dois números primos distintos p e q e definir $n = pq$ e $m = (p-1)(q-1)$;
2. Escolher um $a \in Z_m^* \mid mdc(a, m) = 1$ tal que ;
3. Encontrar um $b \mid ab = 1 \ mod \ m$. Este passo é necessário para a decifragem;
4. Mapear o texto através da tabela 3;
5. Elevar cada inteiro da mensagem a potência a e reduzindo ao módulo n.

Um exemplo simples pode ser visualizado a seguir. Iremos cifrar a mensagem **"VIDA LONGA E PROSPERA"**, escolhendo $p = 5$ e $q = 11$, $n = pq = 55$, $m = (p-1)(q-1) = 40$ e $a = 3$. Utilizando a tabela de correspondências, obtemos:

Agora só resta realizar os cálculos:

Tabela 4: Mapeamento do texto em inteiros.

V	I	D	A	L	O	N	G	A	E	P	R	O	S	P	E	R	A
21	8	3	0	11	14	13	6	0	4	15	17	14	18	15	4	17	0

$$
\begin{aligned}
21^3 &= 9261 \ mod\ 55 = \underline{21} \ mod\ 55 \\
08^3 &= 512 \ mod\ 55 = \underline{17} \ mod\ 55 \\
03^3 &= = \underline{27} \ mod\ 55 \\
0^3 &= = \underline{0} \ mod\ 55 \\
11^3 &= 1331 \ mod\ 55 = \underline{11} \ mod\ 55 \\
14^3 &= 2744 \ mod\ 55 = \underline{49} \ mod\ 55 \\
13^3 &= 2197 \ mod\ 55 = \underline{52} \ mod\ 55 \\
06^3 &= 216 \ mod\ 55 = \underline{51} \ mod\ 55 \\
0^3 &= = \underline{0} \ mod\ 55 \\
04^3 &= 64 \ mod\ 55 = \underline{9} \ mod\ 55 \\
15^3 &= 3375 \ mod\ 55 = \underline{20} \ mod\ 55 \\
17^3 &= 4913 \ mod\ 55 = \underline{18} \ mod\ 55 \\
14^3 &= 2744 \ mod\ 55 = \underline{49} \ mod\ 55 \\
18^3 &= 5832 \ mod\ 55 = \underline{2} \ mod\ 55 \\
15^3 &= 3375 \ mod\ 55 = \underline{20} \ mod\ 55 \\
04^3 &= 64 \ mod\ 55 = \underline{9} \ mod\ 55 \\
17^3 &= 4913 \ mod\ 55 = \underline{18} \ mod\ 55 \\
0^3 &= = \underline{0} \ mod\ 55
\end{aligned}
$$

Assim, o texto cifrado é a lista de inteiros sublinhados acima: 21 17 27 0 11 49 52 51 0 9 20 18 49 2 29 9 18 0.

Este exemplo tem por finalidade apenas explicar como o sistema funciona, bem como a matemática envolvida. Para fins de segurança, os números p e q devem ser quão grandes possíveis. Além disto, é aconselhado que o texto seja cifrado em blocos, neste exemplo ciframos caracter a caracter. Para decifrar realizamos o mesmo processo, porém utilizando $b = 27$ ao invés de a, onde b é encontrado através do Algoritmo de Euclides.

$$
\begin{aligned}
21^{27} &= \underline{21} \ mod \ 55 \\
17^{27} &= \underline{8} \ mod \ 55 \\
27^{27} &= \underline{3} \ mod \ 55 \\
0^{27} &= \underline{0} \ mod \ 55 \\
11^{27} &= \underline{11} \ mod \ 55 \\
49^{27} &= \underline{14} \ mod \ 55 \\
52^{27} &= \underline{13} \ mod \ 55 \\
51^{27} &= \underline{6} \ mod \ 55 \\
0^{27} &= \underline{0} \ mod \ 55 \\
09^{27} &= \underline{4} \ mod \ 55 \\
20^{27} &= \underline{15} \ mod \ 55 \\
18^{27} &= \underline{17} \ mod \ 55 \\
49^{27} &= \underline{14} \ mod \ 55 \\
02^{27} &= \underline{18} \ mod \ 55 \\
20^{27} &= \underline{15} \ mod \ 55 \\
09^{27} &= \underline{4} \ mod \ 55 \\
18^{27} &= \underline{17} \ mod \ 55 \\
0^{27} &= \underline{0} \ mod \ 55
\end{aligned}
$$

3 Metodologia e Ferramenta Proposta

Este capítulo apresenta os conceitos acerca de satisfatibilidade, bem como os princípios necessários para um melhor entendimento do tema. Além disso, realizamos um estudo sobre a análise automática de software e apresentamos o software utilizado como base. Por fim, apresentamos o método proposto nesta dissertação.

3.1 Sistemas SAT

O interesse em satisfatibilidade está se expandindo por uma variedade de razões, no mínimo porque atualmente mais problemas estão sendo resolvidos mais rápido pelos solucionadores SAT do que por outros meios. Isto porque a satisfatibilidade está na encruzilhada da lógica, da teoria dos grafos, da ciência e da engenharia da computação e da pesquisa operacional. Assim, diversos problemas que se originam destes campos normalmente têm traduções múltiplas para a satisfatibilidade e existem muitas ferramentas matemáticas disponíveis para os solucionadores SAT que o auxiliam a resolvê-los com melhor desempenho.

Devido às fortes ligações a tantos campos, especialmente à lógica, a satisfatibilidade pode ser melhor compreendida a medida que se desdobra em relação às suas raízes lógicas. Assim, este capítulo segue a presença da satisfatibilidade na lógica a medida que esta foi desenvolvida para modelar o raciocínio científico através de seu uso, e agora como ferramenta de modelagem para resolver uma variedade de problemas práticos. Para ter sucesso nisso, devemos introduzir muitas ideias que surgiram durante inúmeras tentativas de raciocinar com lógica e isso requer alguma terminologia e perspectiva específica.

Lógica trata de validade e consistência. As duas ideias podem ser definidas se fizermos uso da negação ($\neg$): o argumento de $p_1, p_2, \ldots, p_n$ para q é válido se e somente se o conjunto $\{p_1, \ldots, p_n, \neg q\}$ é inconsistente. Assim, validade e consistência são realmente duas maneiras de olhar para a mesma coisa, e cada um pode ser descrito em termos de sintaxe ou semântica.

3.1.1 Lógica Proposicional

Nesta seção iremos apresentar definições básicas acerca da lógica proposicional, para uso em satisfatibilidade. Serão tratadas notações, formas de representação e conversão dentre as formas de representação.

Uma lógica proposicional é um sistema formal no qual fórmulas representam sentenças declarativas, ou proposições que podem ser verdadeiras ou falsas, mas nunca

ambas [Ribas, 2015]. "*O gato está vivo.*" e "*A resposta é 42.*" são exemplos de proposições.

Na lógica proposicional, é possível criar sentenças compostas utilizando conectivos lógicos. São cinco:

- Negação ($\neg$);
- E ($\wedge$);
- OU ($\vee$);
- Se ... então ($\Rightarrow$);
- Se e somente se ($\Leftrightarrow$).

Como mostrado em Enderton [2001], podemos definir as regras que definem recursivamente uma fórmula como:

- Um átomo é uma fórmula;
- Se ϕ é uma fórmula, então $\neg\phi$ também é uma fórmula;
- Sendo ϕ e ρ fórmulas, então $(\phi \wedge \rho)$, $(\phi \vee \rho)$, $(\phi \Rightarrow \rho)$, $(\phi \Leftrightarrow \rho)$,também serão;
- As fórmulas são geradas, sem exceção, pela aplicação das regras anteriores.

As variáveis proposicionais podem ser definidas como $x_1, x_2, \cdots, x_n$ e podem receber valores *verdadeiro* ou *falso*. O valor assinalado em uma variável r é denotado como $v(r)$. Um literal l é uma variável r ou sua negação $\neg r$. Uma cláusula c é a disjunção de literais.

Uma fórmula FNC (Forma Normal Conjuntiva, do inglês *Conjunctive Normal Form*) é a conjunção de cláusulas. Uma cláusula é dita satisfeita quando pelo menos um de seus literais assume o valor 1 (verdadeiro) e não satisfeita quando todos seus literais assumirem valor 0 (falso). Cláusula unitária é aquela que possui apenas um literal. Literal puro é aquele que aparece em apenas uma forma em toda a fórmula, ou seja, a variável r aparece apenas na forma r ou $\neg r$.

O problema de satisfatibilidade consiste em decidir se existe uma valoração que torna a fórmula verdadeira, ou seja, uma atribuição de valores para as variáveis da fórmula que torne a fórmula satisfeita.

3.1.2 Álgebra Booleana

Álgebras Booleanas, vistas a partir do ponto de álgebra abstrata, são estruturas algébricas que se utilizam de propriedades dos operadores lógicos e de conjuntos.

O estado da lógica avançou consideravelmente quando George Boole introduziu o conceito da estrutura lógica que leva seu nome em Boole [1854].

Na álgebra existem três operações básicas: operação **ou**, **e** e **complementação**. Todas as funções Booleanas podem ser representadas em termos destas operações básicas.

- Operação **ou** (adição lógica - $\vee$): a operação **ou** resulta **1** se pelo uma das variáveis de entrada vale 1. Tendo em vista que uma variável, neste contexto, vale 1 ou 0, e o resultado da operação pode ser visto como uma variável, é necessário apenas que seja definido quando a operação vale 1;

Tabela 5: Tabela verdade da operação OU.

A	B	A$\vee$B
0	0	0
0	1	1
1	0	1
1	1	1

Na tabela 5 podemos ver o resultado da operação $A \vee B$ utilizando duas variáveis. Para o caso de mais variáveis, o comportamento pode ser mostrado de forma análoga. É importante notar que, por existir apenas um operador na equação, podemos avaliá-la decompondo-a em pares. Por exemplo, $A \vee B \vee C$, podemos encontrar o resultado de $A \vee B$ e depois operar os valores com C. Isto pelo fato de atender a propriedade associativa, assim como a ordem adotada entre A, B e C é irrelevante, devido a propriedade comutativa.

- Operação **e** (multiplicação lógica - $\wedge$): a operação **e** resulta **0** se pelo uma das variáveis de entrada vale 0. Por definição, o resultado da operação será 1 se e somente se todas as entradas forem 1.;

Tabela 6: Tabela verdade da operação E.

A	B	A$\wedge$B
0	0	0
0	1	0
1	0	0
1	1	1

Na tabela 6 podemos ver o resultado da operação $A \wedge B$. De forma semelhante é possível encontrar o resultado de $A \wedge B \wedge C$, assim como as propriedades associativa e comutativa valem para a operação **ou**, são válidas também para a operação **e**, então a operação pode ser realizada aos pares e em qualquer ordem.

- Operação negação ($\neg$): é a operação cujo resultado é o valor complementar ao que a variável apresenta.

Tabela 7: Tabela verdade da operação negação.

A	$\neg$A
0	1
1	0

Diferentemente das operações **ou** e **e**, a operação negação é definida sobre uma única variável, como visto na tabela 7, tornando assim o operador negação unitário, enquanto que nas outras operações, o operador é dito binário.

Uma estrutura $\langle B, \vee, \wedge, \neg, 0, 1\rangle$ é uma álgebra Booleana se e somente se $\vee$ e $\wedge$ são operações binárias e $\neg$ é uma operação unitária em B, onde B é fechado e $0, 1 \in B$ e:

- $x \vee y = y \vee x$
- $x \wedge y = y \wedge x$
- $x \vee (y \wedge z) = (x \wedge y) \vee (x \wedge z)$
- $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
- $x \vee \neg x = 1$
- $1 \wedge x = x$
- $0 \vee x = x$
- $x \wedge \neg x = 0$

A principal inovação foi desenvolver uma notação algébrica para propriedades elementares dos conjuntos. O objetivo era fornecer uma teoria mais geral da lógica dos termos que tem a lógica silogística tradicional. Além disto, a notação criada por Boole deu abertura para a criação dos padrões de operações Booleanas em conjuntos e formulações das leis da álgebra Booleana, incluindo a associação, a comutação e operações distributivas.

3.1.3 SAT

Satisfatibilidade Booleana é o problema de decidir se uma fórmula é satisfatível, ou seja, verificar se há uma valoração para as variáveis da fórmula que a torne verdadeira. A questão de SAT é definida como um problema de decisão, em que dada uma fórmula Booleana de entrada, deve-se estabelecer se há uma interpretação que a satisfaça.

O problema SAT é de grande importância em várias áreas da ciência, incluindo teoria da computação, inteligência artificial e verificação se software e hardware. Como visto em [Cook, 1971], SAT foi o primeiro problema provado ser NP-Completo, e não há algoritmos eficientes conhecidos para resolvê-lo, no entanto, alguns resultados relevantes foram obtidos nos últimos anos, como Moskewicz et al. [2001], Eén and Sörensson [2003], Marques-Silva and Sakallah [1999], Sorensson and Een [2005].

Os solucionadores SAT mais conhecidos foram desenvolvidos utilizando duas abordagens: procedimentos heurísticos de busca local Selman et al. [1992] e procedimentos de busca baseados no clássico algoritmo Davis-Putnam-Logemann-Loveland (DPLL) Davis et al., Davis and Putnam [1960]. Entretanto, os algoritmos da primeira abordagem, no geral, não resultam em algoritmos completos, ou seja, não há garantia que esses encontrem uma valoração satisfazível caso ela exista, e não são capazes de provar que uma fórmula é insatisfazível. Tais algoritmos não podem ser utilizados na verificação, pois a garantia de satisfatibilidade é necessária. Alguns solucionadores não completos são Selman et al. [1993] e Stachniak and Belov [2008].

A motivação para converter algoritmos criptográficos para sistemas SAT vem do trabalho de Soos et al. [2009], onde foi mostrado que a *Crypto-1* foi quebrada [Courtois et al., 2008] utilizando SAT, entretanto a conversão realizada por eles foi a parte mais custosa e demorada. *Crypto-1* é uma cifra simétrica que na época era utilizada em sistemas *Mifare Classic Card*, utilizado amplamente para bilhetagem em transporte público e controle de acesso a prédios.

3.1.4 Forma Normal Conjuntiva

Antes que um problema combinatório possa ser resolvido pelos métodos SAT, normalmente o mesmo deve ser transformado para a Forma Normal Conjuntiva (CNF, do inglês *Conjunctive Normal Form*). CNF é uma conjunção ($\wedge$) de cláusulas. Cada elemento de uma restrição é um literal p ou $\neg p$. Uma cláusula é uma disjunção ($\vee$) de literais. A CNF tem a vantagem de ser uma forma simples, levando a uma fácil implementação de algoritmo.

Existem várias maneiras de transformar um problema na forma normal conjuntiva, nesta seção iremos citar apenas alguns.

3.1.4.1 Conversão por álgebra Booleana

Uma fórmula de lógica proposicional pode ser convertida em CNF logicamente equivalente usando as regras da álgebra Booleana. Considerando a seguinte fórmula proposicional:

$$(a \Rightarrow (c \wedge d)) \vee (b \Rightarrow (c \wedge e)) \tag{3.1}$$

As implicações podem ser decompostas:

$$((a \Rightarrow c) \wedge (a \Rightarrow d)) \vee ((b \Rightarrow c) \wedge (b \Rightarrow e)).$$

As conjunções e disjunções podem ser rearranjadas:

$$((a \Rightarrow c) \vee (b \Rightarrow c)) \wedge ((a \Rightarrow c) \vee (b \Rightarrow e)) \wedge ((a \Rightarrow d) \vee (b \Rightarrow c)) \wedge ((a \Rightarrow d) \vee (b \Rightarrow e))$$

As implicações podem ser reescritas como disjunções e literais duplicados podem ser removidos:

$$(\neg a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee c \vee e) \wedge (\neg a \vee b \vee c \vee d) \wedge (\neg a \vee \neg b \vee d \vee e)$$

Finalmente, as cláusulas podem ser removidas, deixando a conjunção:

$$(\neg a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee d \vee e)$$

3.1.4.2 Conversão por Tseitin

Codificação de Tseitin [Tseitin, 1968] é uma forma de obter uma fórmula CNF. Essas codificações geram um número linear de cláusulas ao custo de introduzir novas variáveis e geram uma fórmula satisfatível (apenas se a fórmula inicial for satisfatível). Este método funciona adicionando novas variáveis a fórmula, uma para cada sub-fórmula da fórmula original, juntamente com cláusulas para capturar as relações entre essas novas variáveis. No exemplo, eq. 3.1, a transformação de Tseitin introduziria uma variável f_1 com a definição: $f_1 \leftrightarrow (c \wedge d)$ para representar a sub-fórmula $c \wedge d$.

Esta definição pode ser reduzida à forma clausal: $(\neg f_1 \vee c) \wedge (\neg f_1 \vee d) \wedge (\neg c \vee \neg d \vee f_1)$

De forma similar, temos que: $f_2 \leftrightarrow (c \wedge e)$, que reduz a cláusula: $(\neg f_2 \vee c) \wedge (\neg f_2 \vee e) \wedge (\neg c \vee \neg e \vee f_2)$

Aplicando estas definições na fórmula original: $(\neg a \vee f_1) \wedge (\neg b \vee f_2)$

Analogamente, $f_3 \leftrightarrow (a \vee f_1)$ e $f_4 \leftrightarrow (b \vee f_2)$

e as cláusulas: $(\neg f_3 \vee \neg a \vee f_1) \wedge (a \vee f_3) \wedge (\neg f_1 \vee f_3) \wedge (\neg f_4 \vee \neg b \vee f_2) \wedge (b \vee f_4) \wedge (\neg f_2 \vee f_4)$

A fórmula reduzida, neste momento, é: $(f_3 \vee f_4)$.

Finalmente, uma variável será introduzida com definição $f_5 \leftrightarrow (f_3 \vee f_4)$

e as cláusulas: $(f_5 \vee f_3 \vee f_4) \wedge (f_3 \vee f_5) \vee (f_4 \vee f_5)$

Esta é uma transformação utilizada para converter circuitos em expressões Booleanas, usando as Leis de Morgan e propriedades distributivas para convertê-los em CNF. Entretanto, isto pode resultar em um aumento exponencial no tamanho da equação. A transformação de Tseitin gera uma saída que cresce linearmente em relação à entrada.

3.1.4.3 Formato de arquivo DIMACS para CNF

Um formato de arquivo para problemas SAT em CNF foi concebido em Challenge [1993] e o mesmo tem sido seguido desde então. A existência de um formato padrão de arquivo facilitou na coleta de problemas SAT, e vem estimulado uma grande quantidade de pesquisas em algoritmos e implementações eficientes.

O formato do arquivo é como segue. No início está um preâmbulo contendo as informações acerca do arquivo, uma linha do seguinte formato

p cnf variables clauses

indica o número de variáveis e cláusulas no arquivo, onde *variables* e *clauses* são inteiros positivos, e as variáveis são numeradas de 1 até n. O restante do arquivo contém as cláusulas, onde cada cláusula é representada por uma lisa de inteiros diferente de zero, seguida por um zero, que representa o fim da cláusula. Um inteiro positivo i representa um literal positivo com número i, enquanto que um inteiro negativo $-i$ representa um literal que se encontra na forma $\neg i$. Por exemplo, a linha:

1 2 7 -13 0

representa a cláusula $v_1 \vee v_2 \vee v_7 \vee \neg v_{13}$. As cláusulas podem ter mais de uma linha, espaços e tabulações podem ser inseridos livremente entre inteiros, e tanto a ordem de literais em uma cláusula e a ordem das cláusulas no arquivo são irrelevantes.

3.2 *Model Checking*

Em ciência da computação, Verificação de Modelos, do inglês *Model Checking*, é um termo cunhado por Clarke and Emerson [1981] e refere-se ao seguinte problema: dado um modelo de um sistema, verifica-se de forma exaustiva e automática se este modelo atende a uma determinada especificação. Quando aplicado em sistemas de software ou hardware, a especificação contém os requisitos de segurança, tais como ausência de impasses e estados críticos semelhantes que podem levar a falha do sistema.

A fim de resolver um problema desta classe de forma algorítmica, tanto o modelo do sistema quanto a especificação são formuladas em alguma linguagem matemática. Neste caso, são formulados como uma tarefa lógica, isto é, verificar se uma determinada estrutura satisfaz uma lógica dada uma fórmula. O conceito é geral e se aplica a todos os tipos de lógicas e estruturas adequadas.

Na verificação de modelos, o projeto a ser verificado é modelado como uma máquina de estados finitos e a especificação é formalizada através da escrita das propriedades da lógica temporal. Os estado alcançáveis do projeto são percorridos a fim de verificar as propriedades, no caso de falha da propriedade, um contraexemplo é gerado.

A verificação de modelos é frequentemente utilizada para encontrar erros lógicos ao invés de provar que eles não existem. Os usuários de ferramentas deste tipo de verificação normalmente consideram-no como complementar para os métodos mais tradicionais de

testes e simulação, e não como uma alternativa [Biere et al., 2003]. Estas ferramentas são capazes de encontrar erros que não são suscetíveis de serem encontrados pela simulação. A razão para isso é que, ao contrário dos simuladores que examinam um conjunto relativamente pequeno de casos de teste, verificadores de modelos consideram todos os possíveis comportamentos ou execuções de um sistema.

Model Checking possui três características básicas como uma técnica de verificação:

- É automático: não depende da interação complicada com o usuário. Se uma propriedade não é válida, o mesmo gera um contraexemplo automaticamente;
- Os sistemas que estão sendo verificados são assumidos como finitos;
- A lógica temporal é usada para especificar as propriedades do sistema.

Assim, *model checking* pode ser resumida como uma técnica algorítmica para verificar propriedades temporais de sistemas finitos.

3.3 Bounded Model Checking

Da técnica conhecida por *model checking*, foi criada a técnica *Bounded Model Checking* (BMC) [Biere et al., 1999], onde a motivação inicial era alavancar o sucesso do SAT na resolução de fórmulas Booleanas para a verificação de modelos. Durante os últimos anos, tem havido um enorme aumento no poder de raciocínio dos solucionadores SAT, eles são capazes de lidar com instâncias com centenas de milhares de variáveis e milhões de cláusulas, algo que não era possível num passado não tão distante.

Nos últimos anos, duas tendências principais sobre verificação formal se destacaram. A primeira é que o *bounded model checking* baseado em SAT tornou-se a principal técnica para a verificação de modelos de hardware. BMC constrói uma fórmula proposicional descrevendo todas as execuções possíveis do sistema de comprimento k, para algum limite k. Esta fórmula, juntamente com a negação da especificação, é alimentada em um solucionador SAT, se a fórmula for satisfeita, a especificação é violada. A segunda tendência é que a verificação de software, usando métodos formais, se tornou uma área de pesquisa ativa. Entretanto, adotar a técnica BMC para software causa um grave problema. Esta técnica é sensível ao comprimento do rastreio do erro, isto é, o número de passos de execução até ser atingido um estado de erro. Os rastreamentos de erros são, geralmente, bastante longos e, portanto, um limite k grande é necessário. Por sua vez, isto pode resultar em uma fórmula proposicional que é muito grande para ser tratada por um solucionador SAT. Em [Ivančić et al., 2004] uma forma de encurtar o comprimento do rastreio é proposta, comprimindo instruções dentro de um bloco básico. No entanto, os traços resultantes podem ser longos ainda.

3.3.1 CBMC

C-Bounded Model Checker (CBMC) [Clarke et al., 2004] apresenta uma abordagem diferente para a utilização de um solucionador SAT, a fim de verificar o software. O CBMC traduz um programa sem loops e sem chamadas de função em uma *forma de atribuição única*, do inglês *single assignment form* (SSA). Nesta forma, as variáveis são renomeadas, de tal forma que cada variável seja atribuída apenas uma vez, como resultado, não há necessidade de uma noção de estado. Esse programa pode ser visto como um conjunto de restrições e resolvido usando um solucionador SAT. Esta técnica é menos sensível ao comprimento de um traço.

Além disto, também pode lidar com ponteiros, matrizes e inteiros de tamanho real. Este fato o distingue de outros verificadores de modelo, que usam abstrações para lidar com estes problemas. A maioria dos programas incluem funções e loops, o CBMC manipula isso limitando o número de vezes que cada loop pode ser executado. Então, é possível chamadas de funções *inline* e até mesmo lidar com recursão. Da mesma forma como ocorre em *Bounded Model Checking* comuns, os limites sobre os loops podem ser aumentados iterativamente até que um erro seja encontrado, ou até que o solucionador SAT não seja mais capaz de continuar a execução.

CBMC [Clarke et al., 2004] é uma ferramenta *open source* que obtém um programa C e um limite. Ele converte o programa em um programa delimitado, desenrolando cada loop para o dado limite. Em seguida, o CBMC gera um conjunto de cláusulas. Existe um mapeamento um-para-um das possíveis execuções do programa limitado para as atribuições satisfatórias do conjunto de cláusulas.

O CBMC gera, automaticamente, especificações de limpeza, como não liberar acesso a ponteiros suspensos, sem acesso a posições de vetores ou matrizes fora dos limites existentes e sem violações de assertions. Assertions especificam se um programa satisfaz a certas condições em pontos específicos. Além disto, o CBMC acrescenta uma cláusula que exige que uma destas especificações seja violada, e então ativa um solucionador SAT sobre essas cláusulas. Se encontrar uma atribuição satisfatória para todas as cláusulas, então segue que existe uma execução válida que viola uma das especificações.

O CBMC gera as cláusulas ao traduzir o código para a forma SSA, no qual cada variável é atribuída não mais de uma vez. Para isso, o CBMC gera várias cópias de cada variável, indexadas de zero ao número de atribuições a essa variável.

Cada instrução em um programa C é executada somente se todas as condições *if* que a conduzem forem avaliadas como *true*. Para refletir isso na cláusula gerada, o CBMC também possui variáveis auxiliares. Cada variável auxiliar está associada à conjunção de todas as condições que constam nos *if*'s que levam a uma determinada declaração no código, se a declaração estiver no *else*, a negação da condição é utilizada.

Para melhor entender o CBMC, vamos exemplificar. Suponha que o CBMC recebe um programa como descrito no algoritmo 1, com limite 2.

Algoritmo 1: Algoritmo para exemplo simples do funcionamento do CBMC.

```
1 x=3;
2 while x>1 do
3 |  if x % 2 == 0 then
4 |  |  x = x/2;
5 |  else
6 |  |  x = 3 * x + 1;
7 |  end
8 end
```

O programa desenvolve o primeiro loop, resultando no programa mostrado no algoritmo 2, além disso ele adiciona uma afirmação que garante um desenvolver suficiente, que no nosso caso falhará. O Algoritmo 3 apresenta as cláusulas que representam o código apresentado no Algoritmo 2. Considere a cláusula (4), ela descreve o comportamento da linha (4). Esta instrução é executada somente se os dois *if*'s que levam a ele são verdadeiros, ou seja, aux_2 é verdadeiro. A cláusula apresenta o seguinte comportamento: se aux_2 é verdadeiro, então $x_1 = x_0/2$; caso contrário, a instrução não é executada e $x_1 = x_0$.

Algoritmo 2: Desenvolver do loop do Algoritmo 1.

```
1  x=3;
2  if (x > 1) then
3  |  if x % 2 == 0 then
4  |  |  x = x/2;
5  |  else
6  |  |  x = 3 * x + 1;
7  |  end
8  |  if (x > 1) then
9  |  |  if x % 2 == 0 then
10 |  |  |  x = x/2;
11 |  |  else
12 |  |  |  x = 3 * x + 1;
13 |  |  end
14 |  |  assert(x <= 1);
15 |  end
16 end
```

Algoritmo 3: Cláusulas que representam o código.

```
1  x0=3;
2  aux1 = x0 > 1;
3  aux2 = aux1 & x0 % 2 == 0;
4  x1 = (aux2 ? x0/2 : x0);
5  aux3 = aux1 & !(x0 % 2 == 0);
6  x2 = (aux3 ? 3 * x1 + 1 : x1);
7  aux4 = aux1 & x2 > 1;
8  aux5 = aux4 & x2 % 2 == 0;
9  x3 = (aux5 ? x2/2 : x2);
10 aux6 = aux4 & !(x2 % 2 == 0);
11 x4 = (aux6 ? 3 * x3 + 1 : x3);
12 especificação: !(x4 <= 1);
```

Como dito anteriormente, CBMC suporta a função *assert* e detecta situações em que um *assert* é violado, além disto, ele também suporta a função *assume*. Esta função informa ao programa que todas as execuções legais do programa devem satisfazer a uma certa cláusula. *Assume* é o oposto do *assert* no sentido de que quando uma cláusula não se mantém em uma determinada execução, o CBMC ignora esta execução.

No CBMC, cada atribuição a uma referência de um ponteiro é realmente instanciada para várias atribuições, uma para cada possível valor do ponteiro. As instâncias são limitadas a valores que o ponteiro pode ter obtido em atribuições anteriores.

Além disto, cada atribuição de array é tratada de forma semelhante, instanciado-a para cada possível valor do índice do array. Uma vez que o programa está limitado, o número de chamadas de *malloc*, função do C que permite alocar memória dinâmica, também é limitado. CBMC trata cada memória alocada como uma variável global, e ele também suporta aritmética de ponteiro dentro dos limites do array.

3.4 Ferramenta Proposta

Nesta seção iremos propor uma ferramenta capaz de converter algoritmos criptográficos em sistema SAT, simplificar o sistema gerado e resolvê-lo, a fim de analisar a saída obtida. Após introduzir os conceitos e métodos deste capítulo, a ferramenta proposta neste trabalho pode ser resumida pelos seguintes itens:

1. Realizar a implementação da cifra escolhida, utilizando a linguagem C ou C++;

2. Realizar a conversão da cifra escolhida no passo anterior para a forma normal conjuntiva. Para isto, utilizamos o CBMC, com parâmetros de entrada *p, key, cipher*, onde *p* representa um texto plano conhecido e *key* representa o tamanho da chave para uma cifra *cipher* escolhida no item anterior. O programa executa a cifra escolhida

sobre o texto plano, gerando assim um texto cifrado. Com todos estes dados, o programa gera um sistema SAT e nos retorna o mesmo;

3. Alteramos o software CBMC para nos retornar quais variáveis (literais) do sistema SAT gerado compõe a chave *key*;

4. Simplificar o sistema SAT obtido no item 2. Tal sistema pode ser escrito na forma de grafos, portanto realizamos uma busca em profundidade utilizando as variáveis de *key* para realizar tal redução;

5. Resolver o sistema SAT através do DPLL.

3.4.1 O Algoritmo DPLL

A maneira mais simples e intuitiva de resolver um sistema SAT é enumerar todas as valorações possíveis e verificar quais satisfazem a fórmula. Esta maneira irá verificar 2^n valorações, onde n é o número de variáveis presentes na fórmula, o que torna o processo inviável para valores grandes de n.

Ao invés de procurar enumerar cada combinação possível de valores, Martin Davis, George Logemann e Donald W. Loveland [Davis et al., 1962] refinaram um algoritmo proposto por Davis and Putnam [1960]. O objetivo principal do algoritmo é reduzir o espaço de busca através de técnicas de *backtracking*. Desde então, o DPLL vem sendo implementado com as mais diferentes heurísticas e ganhou diversas competições de implementações na resolução de problemas SAT.

A ideia deste procedimento é a de construir uma valoração para uma dada fórmula, fornecida como um conjunto de cláusulas. Inicialmente todas as variáveis receberam uma valoração " * ", representando um valor indefinido. A cada iteração do algoritmo, um literal p é escolhido, e atribuímos o valor 1 para este literal (faz-se v(p)=1), caso este literal seja negativo na forma $\neg p$ atribuímos o valor 0 (faz-se v(p)=0).

Com este novo valor, procede-se à simplificação da fórmula. Se tal valoração satisfaz todas as cláusulas, significa que a simplificação levou a um conjunto vazio de cláusulas, então tem-se uma valoração que satisfaz a fórmula inicial. É possível que nem todas as cláusulas sejam satisfeitas, o que quer dizer que alguma cláusula foi provada ser falsa durante o processo. Neste caso, procede-se a escolha de um novo literal p e todo o passo explicado anteriormente é realizado novamente. O algoritmo termina quando uma valoração que torne a fórmula satisfazível for encontrada ou quando não existem mais variáveis a serem testadas, o que nos diz que a fórmula é insatisfazível [Da Silva et al., 2006]. Os algoritmos 4 e **??** ilustram o que foi descrito acima.

Algoritmo 4: Algoritmo DPLL.

Input: Fórmula F no formato CNF
Output: True, se F é satisfatível. False caso contrário.

```
1  v(p) = * // para toda variável;
2  F = simplifica(F);
3  if F' = 0 then
4  |  return true;
5  else if F' contém uma cláusula vazia (falsa) then
6  |  return false;
7  end
8  Escolha um literal l com v(l) = * ;
9  if DPLL(F' ∪ L == true) then
10 |  return true;
11 else if DPLL(F' ∪ ¬L) then
12 |  return true;
13 else
14 |  return false;
15 end
```

A linha 8 do Algoritmo 4 contém o passo em que um literal deve ser escolhido para dar início ao processo de construir uma valoração para a fórmula. No nosso caso, escolhemos apenas literais que compõe a chave, pois ela representa o núcleo do nosso problema.

Várias técnicas podem ser utilizadas na fase de simplificação da fórmula Booleana, linha 2 do Algoritmo 4. As mais conhecidas são:

- *Boolean Constraint Propagation* (BCP): é a simplificação feita pela propagação de literais unitários;
- Eliminação de literais puros: um literal é dito puro se ocorre na mesma polaridade nas cláusulas em que aparece. Neste caso, basta atribuir o valor verdadeiro para o mesmo e apagar todas as cláusulas que o contém.
- Resolução de literais simples: um literal é simples se ocorre na forma positiva ou na forma negativa em uma única cláusula. Neste caso, o literal é eliminado, resolvendo-se todas as cláusulas que o contém, diminuindo a complexidade do problema.

Para exemplificar o funcionamento do algoritmo 4, retirado de Jacques Fux [2004], temos que:

$$(\neg x_1) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_3 \vee \neg x_1 \vee x_2)$$

Utilizando a regra das cláusulas unitárias, a heurística assume que $(\neg x_1)$ vale 1, chegando a fórmula:

$$(x_1 \vee x_2 \vee x_4) \wedge (x_3 \vee \neg x_1 \vee x_2)$$

Ainda pela mesma regra, observamos que a cláusula $(x_3 \vee \neg x_1 \vee x_2)$ contém $\neg x_1$, e por isso a mesma é eliminada, ficando:

$$(x_1 \vee x_2 \vee x_4)$$

Sabendo que $\neg(\neg x_1) = x_1$, eliminamos x_1 da cláusula:

$$(x_2 \vee x_4)$$

Assumindo $x_2 = 1$ temos:

$$(x_2 \vee x_4) \wedge (x_2)$$

Recursivamente, aplicando a regra da cláusula unitária, resolvemos o problema com o assinalamento de $x_1 = 0$ e $x_2 = 1$, como pode ser visto na figura 2. A fórmula é dita satisfeita.

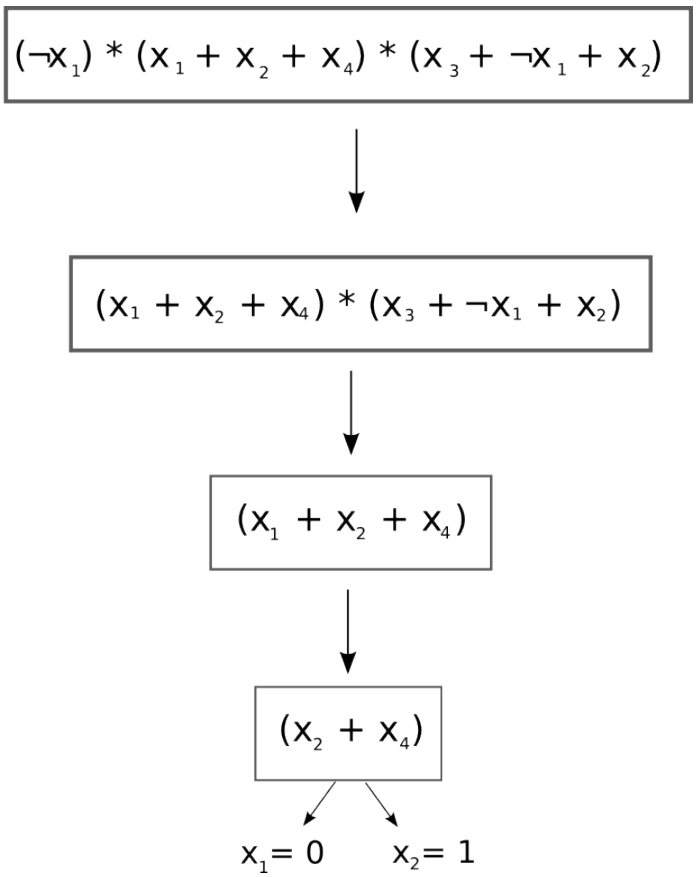

Figura 2: Exemplo de execução do DPLL.

3.4.2 Funcionamento da ferramenta

Nesta seção iremos explicar a forma como a ferramenta desenvolvida foi utilizada para o desenvolvimento desta dissertação. Os experimentos executados foram controlados, ou seja, a partir de um texto plano conhecido e tamanho de chave previamente escolhido. Vamos tomar como exemplo uma cifra genérica, que pode ser visualizada no algoritmo 5.

Algoritmo 5: Cifra genérica.

```
1 char key;
2 char p = 0x4F;
3 char c = 0x15;
4 assert(( key ^ p) != c);
```

O algoritmo 5 mostra um exemplo simples de uma cifra, que representa apenas um XOR entre a chave e o texto plano e tem por objetivo recuperar a chave. Através do *assert*, o CBMC procura gerar um contraexemplo. Neste exemplo, temos um texto plano p, um texto cifrado c e uma chave *key*. O programa faz asserts, verificando se existe alguma combinação possível para a chave utilizada. Rodando este exemplo no CBMC, o software converte o programa e retorna um sistema SAT para ser resolvido. Nossa primeira contribuição foi realizar alterações no software, alteramos o código do CBMC para que o mesmo retornasse como saída os literais correspondentes as variáveis *p, c* e com a chave *key*. Ou seja, resolvendo o sistema sabemos os valores das variáveis de interesse.

O retorno do software para este exemplo pode ser visualizado abaixo.

```
1  CBMC version 4.9 64-bit linux
2  file c.c: Parsing
3  Converting
4  Type-checking c
5  Generating GOTO Program
6  Adding CPROVER library
7  Function Pointer Removal
8  Partial Inlining
9  Generic Property Instrumentation
10 Starting Bounded Model Checking
11 size of program expression: 37 steps
12 simple slicing removed 6 assignments
13 Generated 1 VCC(s), 1 remaining after simplification
14 p cnf 34 24
15 1 0
16 -2 -26 0
17 3 -26 0
18 -4 -26 0
19 5 -26 0
20 6 -26 0
```

```
21 -7 -26 0
22 8 -26 0
23 -9 -26 0
24 2 -3 4 -5 -6 7 -8 9 26 0
25 26 0
26 -27 -29 -31 0
27 27 29 -31 0
28 -27 29 31 0
29 27 -29 31 0
30 -28 -30 -32 0
31 28 30 -32 0
32 -28 30 32 0
33 28 -30 32 0
34 -31 -33 0
35 -32 -33 0
36 31 32 33 0
37 -1 33 0
38 1 -33 0
39 c c::__CPROVER_pipe_count#1
40 c c::__CPROVER_thread_id!0#1
41 c c::__CPROVER_next_thread_id#1
42 c c::main::1::key!0@1#1 2 3 4 5 6 7 8 9
43 c c::main::1::p!0@1#1 10 11 12 13 14 15 16 17
44 c c::__CPROVER_rounding_mode!0#1
45 c c::main::1::p!0@1#2
46 c c::main::1::c!0@1#1 18 19 20 21 22 23 24 25
47 c c::__CPROVER_deallocated#1
48 c c::main::1::c!0@1#2
49 c c::__CPROVER_dead_object#1
50 c c::__CPROVER_malloc_object#1
51 c c::__CPROVER_malloc_size#1
52 c c::__CPROVER_memory_leak#1
```

O sistema propriamente dito começa na linha 15 e vai até a linha 38. A linha 42 nos mostra as variáveis do sistema que representam a chave da cifra, que é a informação de nosso interesse. A figura 3 nos mostra o grafo deste sistema, onde os vértices representam as variáveis e as arestas representam que existe uma equação que associa estas duas variáveis.

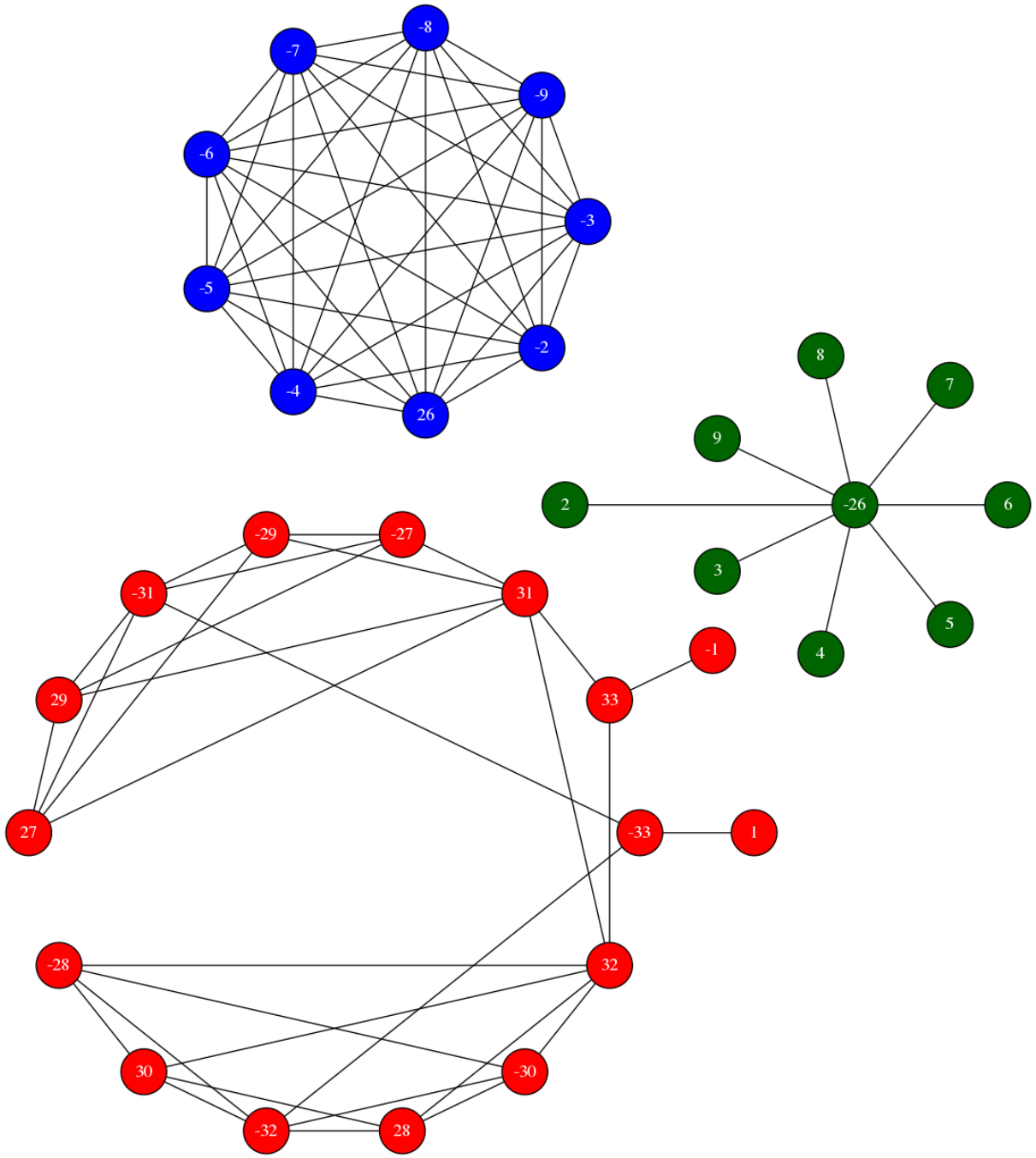

Figura 3: Grafo do sistema.

Analisando os grafos, vemos que existem equações que não possuem relação com as variáveis de nosso interesse, então podemos descartá-las. Para isso, implementamos uma busca em profundidade, do inglês *depth-first search*, visando melhora no tempo de resolução do sistema. O sistema do exemplo citado acima foi reduzido a:

```
1  -2 -26 0
2  3 -26 0
3  -4 -26 0
4  5 -26 0
```

```
6 -26 0
-7 -26 0
8 -26 0
-9 -26 0
2 -3 4 -5 -6 7 -8 9 26 0
26 0
```

A figura 4 nos mostra os grafos resultantes, após eliminar as equações indesejadas.

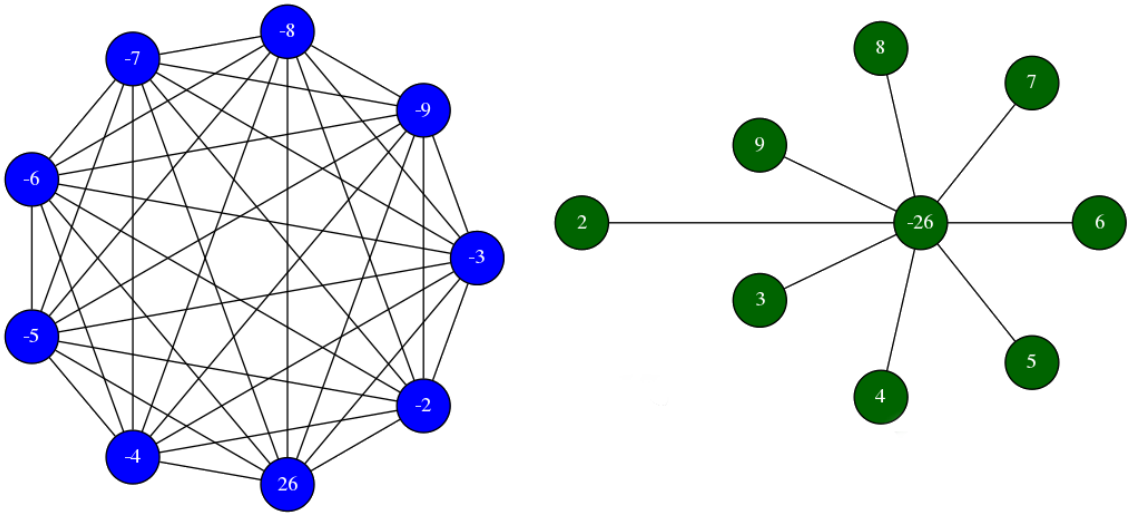

Figura 4: Grafos pós DFS.

Após lapidar o sistema, podemos resolvê-lo através do algoritmo DPLL. Após a resolução do sistema obtemos a seguinte saída:

```
Solucao: satisfativel
-2 3 -4 5 6 -7 8 -9
```

onde os valores negativos representam *false* e os positivos representam *true*. A valoração da variável *key* é, então: 01011010. Realizando os cálculos podemos ver que o valor encontrado é o esperado. Sendo p=0x4F$_{(16)}$ = 1001111$_{(10)}$ e c=0x15$_{(16)}$ = 10101$_{(10)}$, vemos que $p \oplus k = c$.

3.5 Experimentos

Para realizar os experimentos, utilizamos algumas cifras simétricas conhecidas na literatura, tais experimentos serão mostrados no capítulo 4. A ferramenta foi capaz de converter todas as cifras para sistemas SAT. Para validar a ferramenta, utilizamos dois tipos de experimento:

1. Experimentos baseados em executar diversas cifras criptográficas na ferramenta proposta e comparar os tamanhos do sistema antes e depois da simplificação, bem como os tempos de resolução utilizando nossa implementação do DPLL e o MiniSat[1];

2. Trocar um XOR por um AND durante a criptografia de um texto e analisar as saídas.

A troca de um XOR por um AND representa um tipo de ataque, o *Fault Analysis*, mostrado em Biham and Shamir [1997]. O mesmo é um ataque baseado em implementação e tem por objetivo induzir falhas nos sistemas criptográficos por meio de ruídos externos. Em seguida, analisa-se o comportamento da cifra alterada, a fim de recuperar a chave.

[1] Solucionador SAT desenvolvido por Sorensson and Een [2005]

4 Resultados e Discussão

A ferramenta proposta no capítulo 3, que consiste em traduzir algoritmos criptográficos em sistema SAT, foi validada através de uma série de testes. Os resultados obtidos serão exibidos e discutidos neste capítulo.

4.1 Análise de redução do sistema e tempo de resolução

Nesta parte dos resultados realizamos experimentos baseados em executar diversas cifras criptográficas na ferramenta proposta e comparar os tamanhos do sistema antes e depois da simplificação, bem como os tempos de resolução utilizando nossa implementação do DPLL e o MiniSat.

4.1.1 Mini AES

Inicialmente, analisamos a saída da ferramenta quando aplicada ao algoritmo Mini AES. Verificamos o tamanho do sistema gerado e o tempo de resolução do sistema quando utilizamos o MiniSat [Sorensson and Een, 2005]. A tabela 8 exibe os resultados obtidos.

Tabela 8: Tabela contendo o tamanho do sistema gerado e o tempo de resolução utilizando o algoritmo mini AES e resolvendo o sistema através do MiniSat.

Tamanho da Chave	Tamanho do Sistema	Tempo de Resolução
16 bits	1156129 equações	$3,988$ segundos
32 bits	1156084 equações	$4,212$ segundos
64 bits	1155991 equações	$4,136$ segundos

Em seguida, lapidamos o sistema através de uma busca em profundidade, como mostrado anteriormente. A tabela 9 mostra a redução do sistema.

Tabela 9: Tabela contendo a redução do sistema e o tempo de resolução da tabela 8 após aplicar o algoritmo de busca em profundidade.

Tamanho da Chave	Tamanho do Sistema	Redução do Sistema	Tempo de Resolução
16 bits	353380 equações	$69,44\%$	$1,344$ segundos
32 bits	465974 equações	$59,70\%$	$1,896$ segundos
64 bits	695914 equações	$39,80\%$	$2,748$ segundos

Podemos notar que o tempo de resolução do sistema antes do mesmo ser lapidado era relativamente baixo, como mostrado na tabela 8 e que a redução do sistema proposta foi realizada de forma efetiva quando comparamos com a tabela 9.

É importante observar os tamanhos dos sistemas gerados, como pode ser visto na tabela 10. Antes de aplicar a redução, os sistemas gerados para os três tamanhos de chave possuíam número de equações semelhantes, entretanto, a redução mais significativa ocorreu no sistema gerado para a chave de 16 bits. Este fato nos mostra que a real complexidade do problema fica evidente após a redução visto que inicialmente os sistemas se mostravam semelhantes, além disto, nos mostra que a ferramenta proposta atua de forma mais eficaz para problemas menores.

Tabela 10: Tabela contendo os tamanhos dos sistemas antes e depois de reduzi-los.

Tamanho da Chave	Tamanho do Sistema	Tamanho do Sistema Reduzido
16 bits	1156129 equações	353380 equações
32 bits	1156084 equações	465974 equações
64 bits	1155991 equações	695914 equações

Além disto, comparamos o tempo de resolução do sistema utilizando o MiniSat com o tempo de resolução utilizando nossa implementação do DPLL. Os resultados podem ser vistos na tabela 11.

Tabela 11: Tabela comparando os tempos de resolução dos sistemas do Mini AES gerados utilizando o MiniSat e o DPLL.

Tamanho da Chave		Tempo de Resolução (MiniSat)	Tempo de Resolução (DPLL)
16 bits	pré-dfs	$3,988$ segundos	$3,476$ segundos
	pós-dfs	$1,344$ segundos	$0,973$ segundos
32 bits	pré-dfs	$4,212$ segundos	$3,892$ segundos
	pós-dfs	$1,896$ segundos	$1,484$ segundos
64 bits	pré-dfs	$4,136$ segundos	$3,674$ segundos
	pós-dfs	$2,748$ segundos	$2,368$ segundos

Na figura 5 podemos ver os resultados exibidos nas tabelas 8, 9, 11.

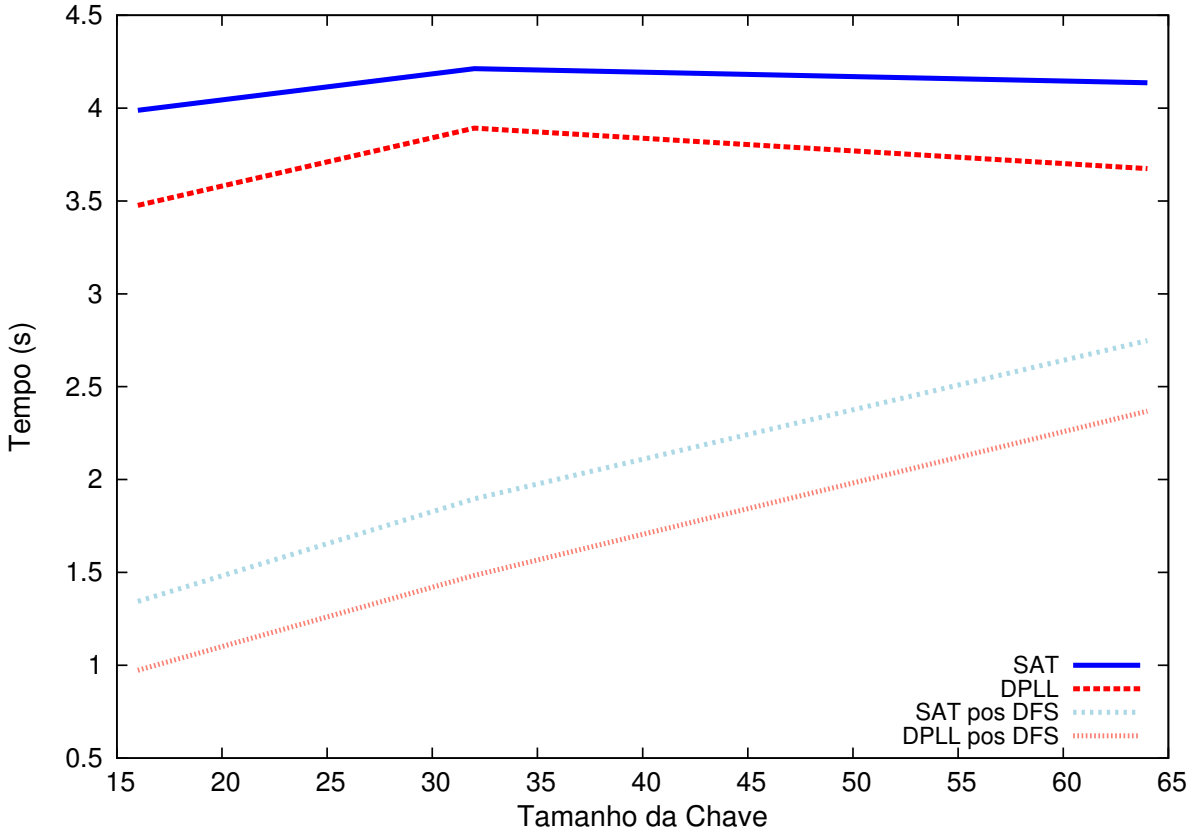

Figura 5: Gráfico contendo os tempos de resolução do sistema Mini AES para os métodos de resolução pré e pós DFS.

4.1.2 AES

Após realizar os experimentos utilizando o Mini AES, aplicamos os mesmos testes sobre o AES, com os tamanhos de chave padrão: 128, 192 ou 256 bits.

Tabela 12: Tabela contendo o tamanho do sistema gerado e o tempo de resolução utilizando o algoritmo AES e resolvendo o sistema através do MiniSat.

Tamanho da Chave	Tamanho do Sistema	Tempo de Resolução
128 bits	1156129 equações	$5,344$ segundos
192 bits	1156084 equações	$4,408$ segundos
256 bits	1155991 equações	$4,416$ segundos

Após isto, o sistema foi reduzido eliminando as equações que não tem relação com as variáveis da chave. A tabela 13 mostra a redução do sistema.

Tabela 13: Tabela contendo a redução do sistema e o tempo de resolução da tabela 12 após aplicar o algoritmo de busca em profundidade.

Tamanho da Chave	Tamanho do Sistema	Redução do Sistema	Tempo de Resolução
128 bits	1054793 equações	$8,8\%$	$4,812$ segundos
192 bits	1054801 equações	$8,76\%$	$3,748$ segundos
256 bits	1054809 equações	$8,75\%$	$3,964$ segundos

Primeiramente, é importante notar as semelhanças entre as tabelas 8 e 12. O tamanho do sistema gerado independe do tamanho da chave utilizado, assim como o tempo de resolução. Agora, se compararmos os dados das tabelas 9 e 13 podemos notar que a redução dos sistemas utilizando os tamanhos de chave padrão do AES é mínima quando olhamos para a redução dos sistemas gerados utilizando o Mini AES. Apesar disto, o método se mostra efetivo em resolver o sistema e obter os valores das variáveis relacionadas a chave.

Da mesma forma como fizemos com o Mini AES, comparamos os tempos de resolução da resolução do sistema utilizando o MiniSat e DPLL. Os resultados podem ser vistos na tabela 14.

Tabela 14: Tabela comparando os tempos de resolução dos sistemas do AES gerados utilizando o MiniSat e o DPLL.

Tamanho da Chave		Tempo de Resolução (MiniSat)	Tempo de Resolução (DPLL)
128 bits	pré	$5,344$ segundos	$4,977$ segundos
	pós	$4,812$ segundos	$4,387$ segundos
192 bits	pré	$4,408$ segundos	$4,023$ segundos
	pós	$3,748$ segundos	$3,322$ segundos
256 bits	pré	$4,416$ segundos	$3,995$ segundos
	pós	$3,964$ segundos	$3,425$ segundos

No gráfico exibido na figura 6 podemos ver os resultados exibidos nas tabelas 12, 13, 14.

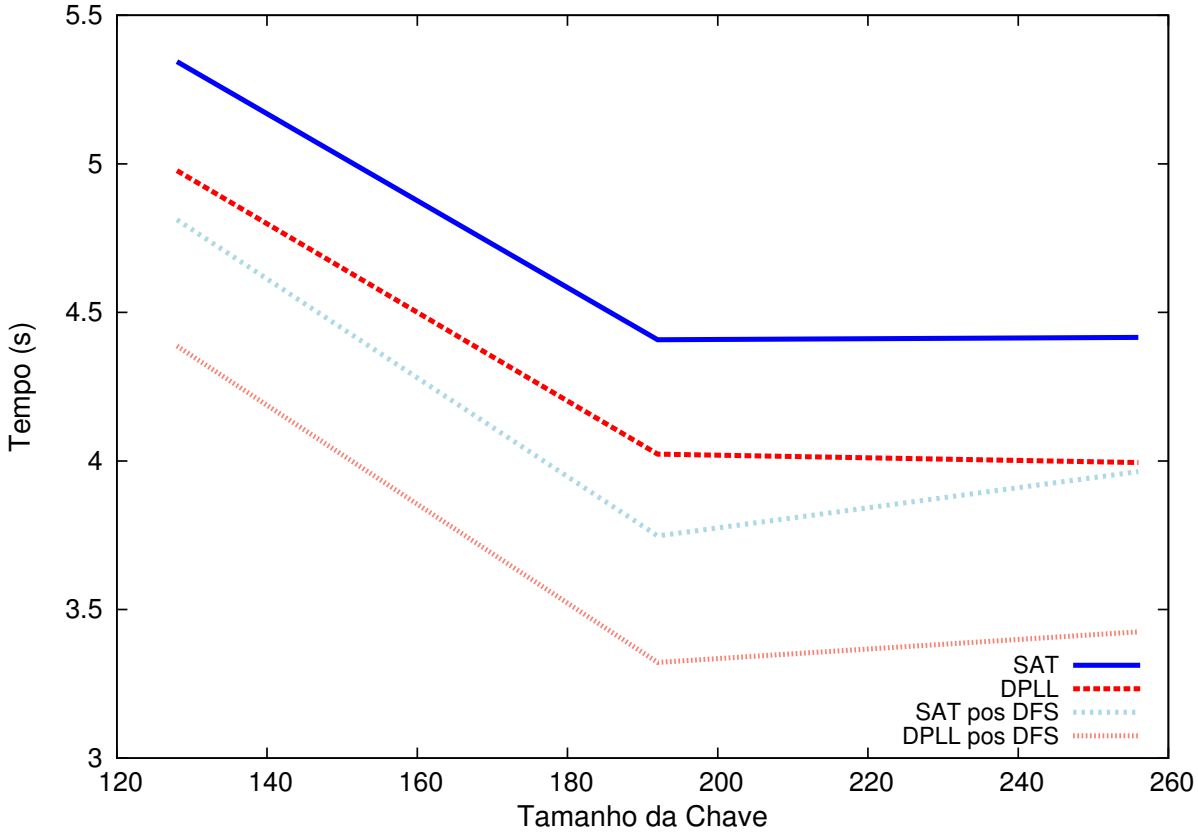

Figura 6: Gráfico contendo os tempos de resolução do sistema AES para os métodos de resolução pré e pós DFS.

4.1.3 DES/3DES

A fim de realizar experimentos mais significativos, submetemos os seguintes algoritmos criptográficos simétricos ao método proposto: DES, 3DES. Como utilizar o DPLL para resolver os sistemas se mostrou mais efetivo que utilizar o MiniSat, os experimentos a partir de agora são todos focados neste tipo de resolução. Os resultados exibidos na tabela 15 correspondem a uma série de execuções.

Tabela 15: Tabela contendo os tempos de resolução dos sistemas.

Tamanho da Chave	Tamanho do Sistema	Tempo de Resolução
56 bits	1024628 equações	$3,128$ segundos
168 bits	1154208 equações	$4,375$ segundos

Após isto, o sistema foi reduzido eliminando as equações que não tem relação com as variáveis da chave. A tabela 16 mostra a redução do sistema.

Tabela 16: Tabela contendo a redução do sistema e tempo de resolução após aplicar o algoritmo de busca em profundidade.

Tamanho da Chave	Tamanho do Sistema	Redução do Sistema	Tempo de Resolução
56 bits	305954 equações	$70,14\%$	$1,083$ segundos
168 bits	849446 equações	$24,49\%$	$3,325$ segundos

O gráfico mostrado na figura 7 foi feito para um melhor entendimento dos resultados exibidos nas tabelas 15 e 16.

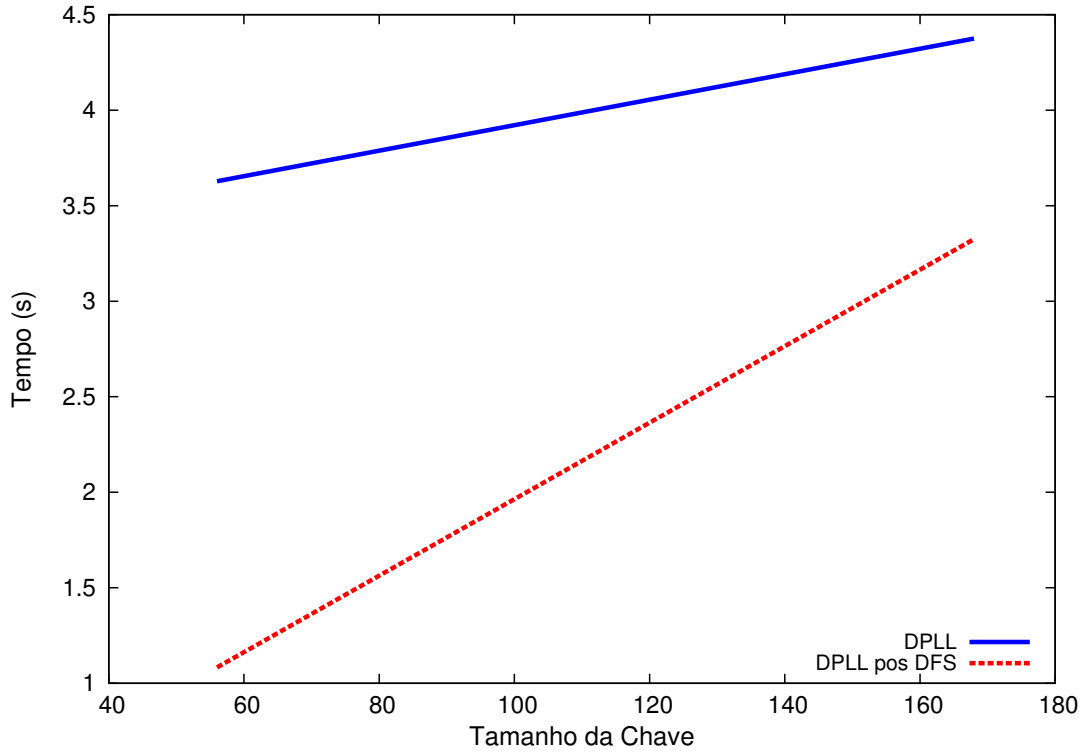

Figura 7: Gráfico contendo os tempos de resolução para os métodos de resolução do sistema pré e pós DFS.

4.1.4 Simon/Speck

A fim de realizar experimentos mais significativos, submetemos os algoritmos criptográficos simétricos ao método proposto: Simon e Speck. Os resultados exibidos na tabela 17 correspondem a uma série de execuções.

Tabela 17: Tabela contendo os tempos de resolução dos sistemas.

Tamanho da Chave	Tamanho do Sistema	Tempo de Resolução
64 bits	1054623 equações	3, 426 segundos
96 bits	1154824 equações	3, 836 segundos
128 bits	1054034 equações	4, 015 segundos
192 bits	1155012 equações	4, 694 segundos
256 bits	1155512 equações	5, 127 segundos

Após isto, o sistema foi reduzido eliminando as equações que não tem relação com as variáveis da chave. A tabela 18 mostra a redução do sistema.

Tabela 18: Tabela contendo a redução do sistema e tempo de resolução após aplicar o algoritmo de busca em profundidade.

Tamanho da Chave	Tamanho do Sistema	Redução do Sistema	Tempo de Resolução
64 bits	503478 equações	52, 26%	1, 441 segundos
96 bits	677074 equações	41, 37%	2, 021 segundos
128 bits	872950 equações	17, 18%	2, 489 segundos
192 bits	1059377 equações	8, 28%	4, 365 segundos
256 bits	1061222 equações	8, 16%	4, 716 segundos

O gráfico mostrado na figura 8 foi feito para um melhor entendimento dos resultados exibidos nas tabelas 17 e 18.

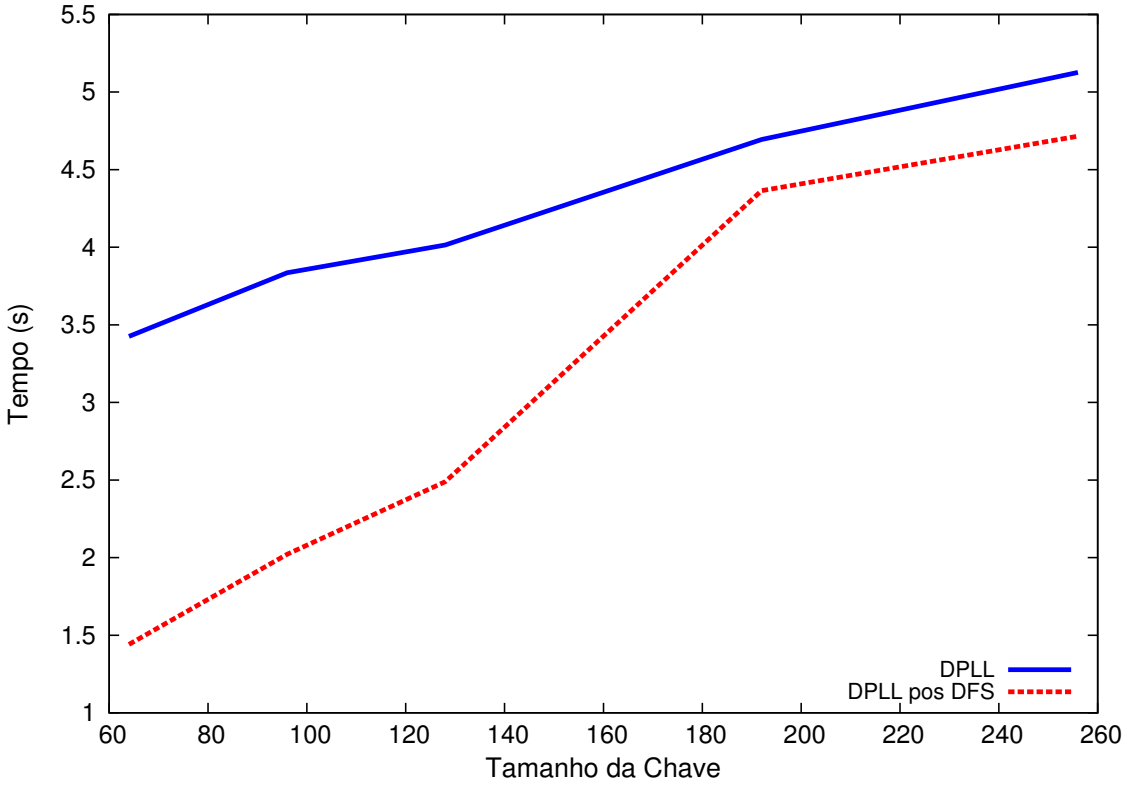

Figura 8: Gráfico contendo os tempos de resolução para os métodos de resolução do sistema pré e pós DFS.

4.2 Análise do impacto da substituição do operador

De forma a obter um resultado diferenciado, analisamos dois sistemas SAT: um gerado sem interferência e outro, utilizando os mesmos parâmetros do primeiro, porém alterando um operador no momento em que um sistema criptográfico está sendo executado. Trocamos um XOR por um AND em um round na função *AddRoundKey*. Realizamos os experimentos utilizando o Mini AES e o AES, para comparar com os resultados obtidos anteriormente.

4.2.1 Mini AES

Como feito anteriormente, analisamos a saída da ferramenta quando aplicada ao algoritmo mini AES. Verificamos o tamanho do sistema gerado e o tempo de resolução do sistema gerado. A tabela 19 exibe os resultados obtidos.

Tabela 19: Tabela contendo o tamanho do sistema gerado e o tempo de resolução utilizando o algoritmo mini AES e resolvendo o sistema através do MiniSat.

Tamanho da Chave	Tamanho do Sistema	Tempo de Resolução
16 bits	1155777 equações	$3,92$ segundos
32 bits	1153115 equações	$4,152$ segundos
64 bits	1149767 equações	$4,004$ segundos

Em seguida, lapidamos o sistema através de uma busca em profundidade. A tabela 20 mostra a redução do sistema.

Tabela 20: Tabela contendo a redução do sistema e o tempo de resolução da tabela 8 após aplicar o algoritmo de busca em profundidade.

Tamanho da Chave	Tamanho do Sistema	Redução do Sistema	Tempo de Resolução
16 bits	353028 equações	$69,45\%$	$1,314$ segundos
32 bits	463006 equações	$59,84\%$	$1,778$ segundos
64 bits	689690 equações	$40,01\%$	$2,576$ segundos

Os gráficos exposto nas figuras 9 e 10 representam uma síntese dos resultados obtidos. Neles comparamos, respectivamente, os tempos de execução do Mini AES antes e depois da modificação proposta e os tamanhos dos sistemas gerados.

É possível notar que os tamanhos dos sistemas gerados, bem como suas reduções são semelhantes. Podemos constatar que a alteração de um operador lógico não apresenta impacto considerável neste caso.

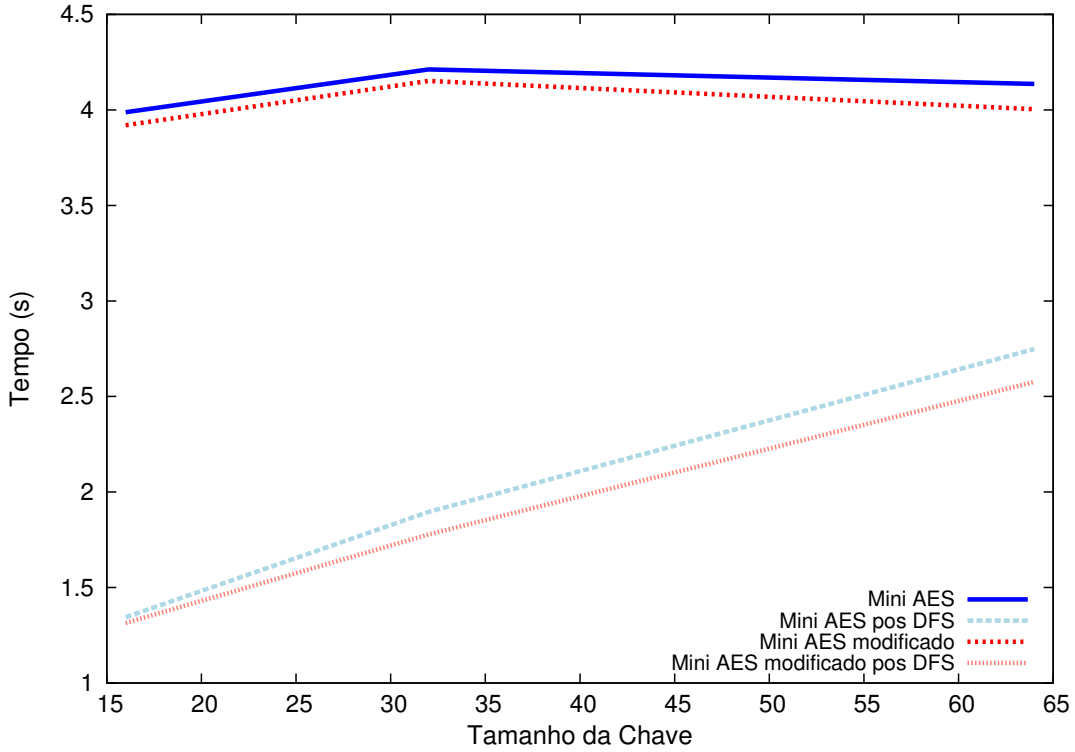

Figura 9: Gráfico comparativo entre os tempos de resolução do sistema Mini AES e Mini AES modificado para os métodos de resolução pré e pós DFS.

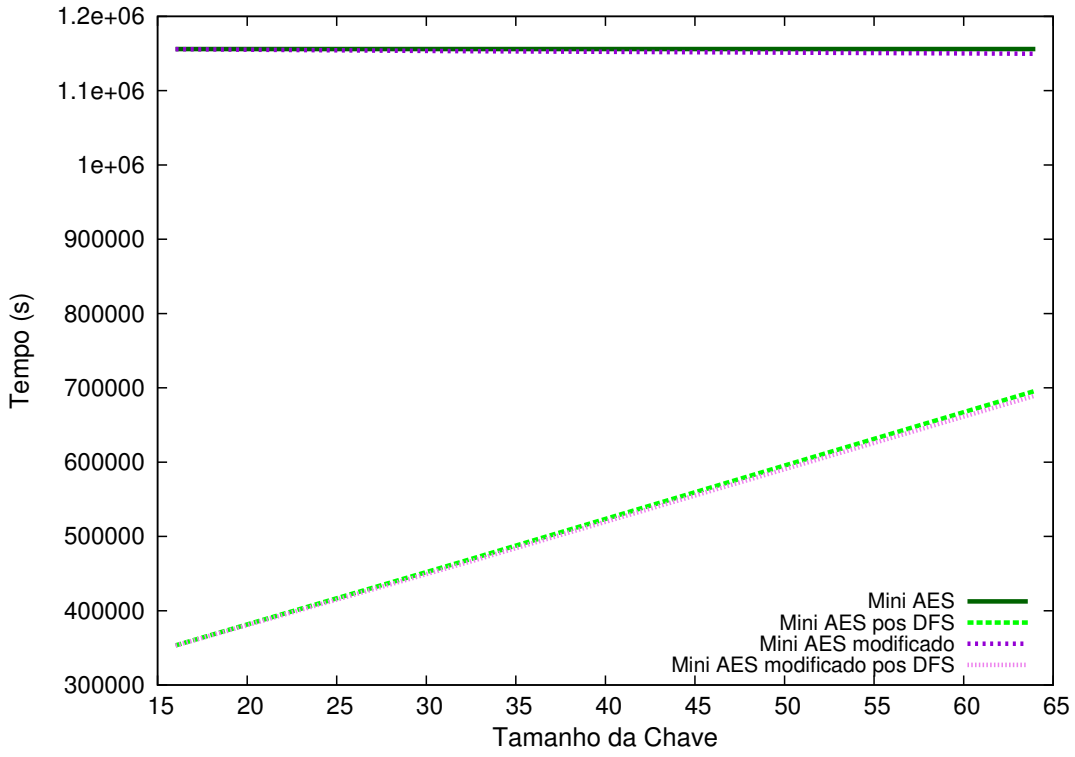

Figura 10: Gráfico comparativo entre os tamanhos do sistema Mini AES e Mini AES modificado para os métodos de resolução pré e pós DFS.

4.2.2 AES

Analisamos a saída da ferramenta quando aplicada ao algoritmo AES. Verificamos o tamanho do sistema gerado e o tempo de resolução quando utilizamos o DPLL para resolver o sistema gerado. A tabela 21 exibe os resultados obtidos.

Tabela 21: Tabela contendo o tamanho do sistema gerado e o tempo de resolução utilizando o algoritmo AES e resolvendo o sistema através do DPLL.

Tamanho da Chave	Tamanho do Sistema	Tempo de Resolução
128 bits	1142987 equações	$5,384$ segundos
192 bits	1142687 equações	$4,276$ segundos
256 bits	1143807 equações	$4,404$ segundos

Após isto, lapidamos o sistema através de uma busca em profundidade, como mostrado anteriormente. A tabela 22 mostra a redução do sistema.

Tabela 22: Tabela contendo a redução do sistema e o tempo de resolução da tabela 8 após aplicar o algoritmo de busca em profundidade.

Tamanho da Chave	Tamanho do Sistema	Redução do Sistema	Tempo de Resolução
128 bits	1042973 equações	$8,75\%$	$5,22$ segundos
192 bits	1042609 equações	$8,75\%$	$4,258$ segundos
256 bits	1043665 equações	$8,75\%$	$4,256$ segundos

Os gráficos exposto nas figuras 11 e 12 representam uma síntese dos resultados obtidos. Neles comparamos, respectivamente, os tempos de resolução do AES antes e depois da modificação proposta e os tamanhos dos sistemas gerados.

Da mesma forma como visto na seção anterior, os tamanhos dos sistemas e os tempos de resolução antes e depois da modificação proposta são semelhantes. Podemos concluir que alterar um operador lógico durante a criptografia não interfere na resolução do sistema da forma proposta nesta dissertação.

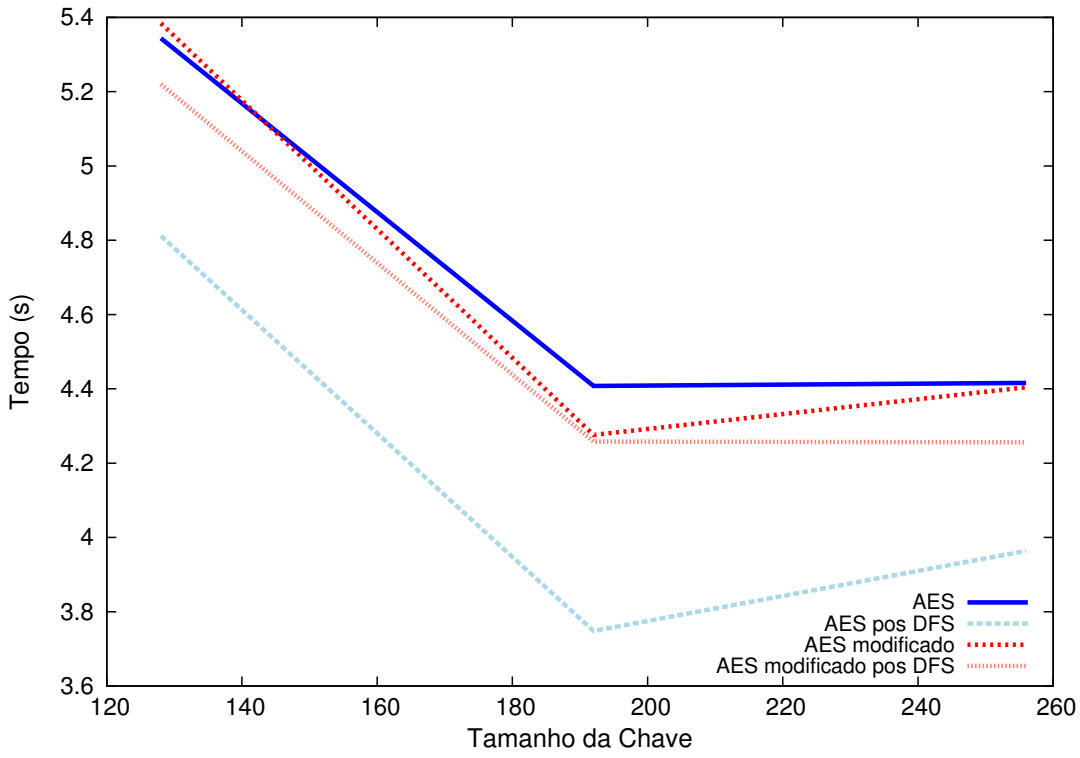

Figura 11: Gráfico comparativo entre os tempos de resolução do sistema AES e AES modificado para os métodos de resolução pré e pós DFS.

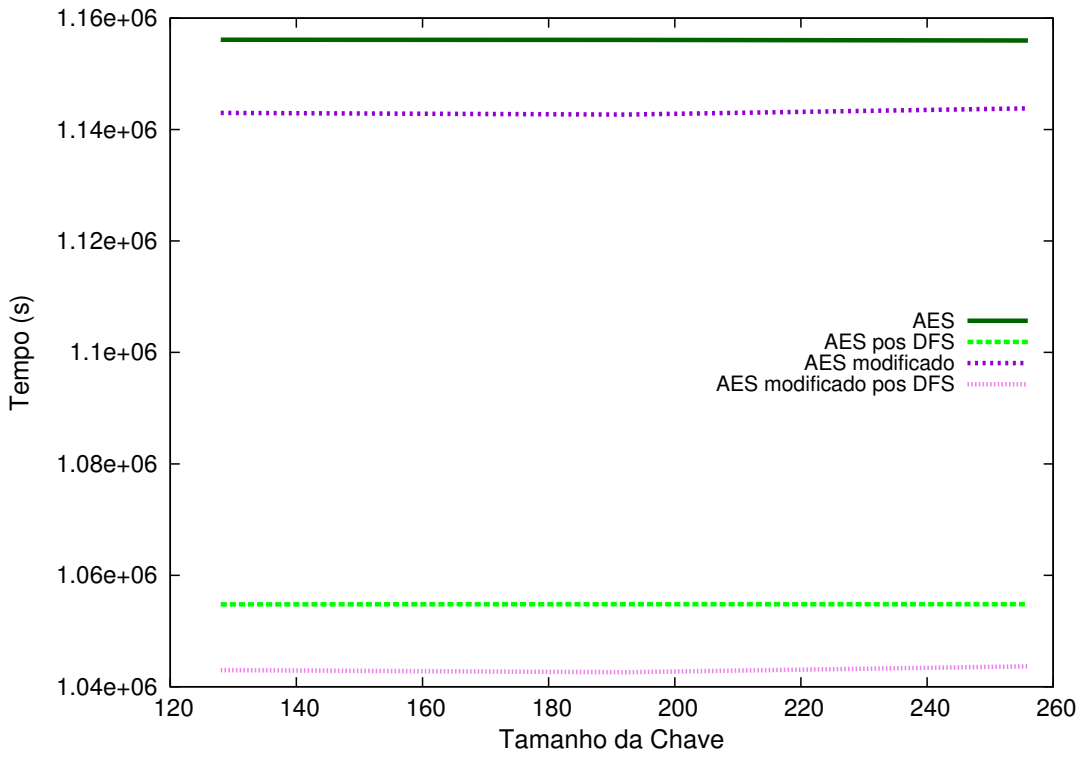

Figura 12: Gráfico comparativo entre os tamanhos do sistema AES e AES modificado para os métodos de resolução pré e pós DFS.

5 Conclusões e Trabalhos Futuros

Através de uma série de estudos que incluem uma análise da literatura e diferentes conjuntos de experimentações sobre análise de algoritmos criptográficos simétricos, este trabalho visa contribuir para o aprofundamento da compreensão acerca do comportamento das cifras quando transformadas em sistemas SAT.

Inicialmente, foi proposto um método que atua na conversão de algoritmos criptográficos para a Forma Normal Conjuntiva utilizando um software verificador de modelos. Através de testes controlados, além de obter um sistema SAT referente a uma cifra, obtemos também as variáveis deste sistema que possuem relação com a chave criptográfica. Em um primeiro momento, o sistema era resolvido com auxílio da ferramenta MiniSat, entretanto realizamos uma implementação do DPLL que se mostrou mais eficaz. Isto se deve ao fato de que tal método funciona através da escolha aleatória de literais do algoritmo na forma normal conjuntiva, porém, como sabemos quais literais tem relação com a chave, escolhemos apenas estes literais. Além disto, reduzimos o sistema, que tinha informações redundantes e desnecessárias para nosso caso.

Realizamos então um estudo comparativo acerca das melhores combinações possíveis em termos das ferramentas utilizadas para resolver os sistemas gerados e um estudo comparativo no sentido de analisar a complexidade dos sistemas gerados por diferentes algoritmos criptográficos. Nossos resultados demonstraram que 1) a superioridade do DPLL modificado frente aos solucionadores SAT convencionais e 2) diferentes cifras simétricas com diferentes tamanhos de chave geram, inicialmente, sistemas igualmente complexos. Acreditamos que estas conclusões preliminares possam ser úteis na formação de novos e mais eficientes modelos para testar a segurança dos algoritmos e modelos para recuperar parcialmente e totalmente as chaves criptográficas ou ainda para motivar a melhoria de modelos já existentes, visando incrementar suas capacidades para estudos teóricos ou resolução de problemas reais.

Após a análise geral, optamos por aprofundar o estudo sobre uma mudança em específico, realizamos uma análise e estudo comparativo dos sistemas gerados antes e depois de uma modificação durante a criptografia de um texto. Alteramos o operador lógico XOR por um AND e analisamos o impacto que isto causa no sistema e na resolução do mesmo. Nossos resultados demonstraram que a troca do operador não altera de forma significativa o método proposto para a resolução do sistema.

Além disto, o estudo inicial na área de segurança de informação gerou um artigo intitulado "*Prediction of steganography in images using a mathematical model*" [Paiva et al., 2015], publicado no XXXVI Ibero-Latin American Congress on Computational

Methods in Engineering - CILAMCE 2015 e o mesmo favoreceu a motivação para dar início a este trabalho.

Dentre os próximos objetivos, podemos destacar a importância de estender o alcance da ferramenta para a análise e resolução de sistemas gerados a partir de algoritmos criptográficos assimétricos. Além disto, gostaríamos de analisar os sistemas e as resoluções quando o mesmo sofre diferentes tipos de ataque, como o *Differential Fault Analysis*.

Referências

Gregory V Bard. Algorithms for the solution of polynomial and linear systems of equations over finite fields, with an application to the cryptanalysis of keeloq. Technical report, Technical report, University of Maryland Dissertation (April 2008), 2008. Citado na página 14.

P Barreto, Felipe Piazza Biasi, Ricardo Dahab, Julio César, Geovandro CCF Pereira, and Jefferson E Ricardini. Introdução à criptografia pós-quântica. *Minicursos do XIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais—SBSeg*, 2013. Citado na página 13.

Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 193–207. Springer, 1999. Citado na página 36.

Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in computers*, 58:117–148, 2003. Citado na página 36.

Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Annual International Cryptology Conference*, pages 513–525. Springer, 1997. Citado na página 47.

George Boole. *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities*. Dover Publications, 1854. Citado na página 31.

DIMACS Challenge. Satisfiability: Suggested format. *DIMACS Challenge. DIMACS*, 1993. Citado na página 34.

Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ansi-c programs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 168–176. Springer, 2004. Citado na página 37.

EM Clarke and EA Emerson. Synthesis of synchronization skeletons for branching time temporal logic, 131, 1981. Citado na página 35.

Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971. Citado 2 vezes nas páginas 14 e 32.

Caio Alonso da Costa. Desenvolvimento de hardware de criptografia rsa em linguagem verilog. 2014. Citado na página 25.

Carlos Henrique Andrade Costa. *Criptografia quântica em redes de informação crítica-aplicação a telecomunicações aeronáuticas.* PhD thesis, Universidade de São Paulo, 2008. Citado na página 13.

Nicolas Courtois, Karsten Nohl, and Sean O'Neil. Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards. *IACR Cryptology ePrint Archive*, 2008:166, 2008. Citado na página 33.

Flávio Soares Corrêa Da Silva, Marcelo Finger, and Ana Cristina Vieira de Melo. *Lógica para computação.* Thomson Learning, 2006. Citado na página 40.

Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999. Citado na página 23.

M Davis, G Putnam, and D Loveland. A machine program for theorem proving, communication of the acm (1966) 394-397. *GS Search.* Citado na página 33.

Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960. Citado 2 vezes nas páginas 33 e 40.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962. Citado na página 40.

Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976. Citado na página 25.

Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *International Conference on Applied Cryptography and Network Security*, pages 164–175. Springer, 2005. Citado na página 13.

Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003. Citado na página 32.

Herbert Enderton. *A mathematical introduction to logic.* Academic press, 2001. Citado na página 30.

Victor Manuel Calhabrês Fiarresga et al. *Criptografia e matemática.* PhD thesis, 2010. Citado na página 15.

F Ivančić, Z Yang, MK Ganai, A Gupta, and P Ashar. Efficient sat-based bounded model checking for software verification. In *in Symposium on Leveraging Formal Methods in Applications.* Citeseer, 2004. Citado na página 36.

A Jacques Fux. *Análise de algoritmos sat para resolução de problemas multivalorados.* PhD thesis, 2004. Citado na página 41.

Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 206–222. Springer, 1999. Citado na página 13.

João P Marques-Silva and Karem A Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999. Citado na página 32.

RJ McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114–116, 1978. Citado na página 13.

Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996. Citado na página 13.

Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001. Citado na página 32.

Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *PROBLEMS OF CONTROL AND INFORMATION THEORY-PROBLEMY UPRAVLENIYA I TEORII INFORMATSII*, 15(2):159–166, 1986. Citado na página 13.

Natasha N Paiva, Renato Portugal, and Pedro Carlos Lara. Prediction of steganography in images using a mathematical model. In *XXXVI CILAMCE – Ibero-Latin American Congress on Computational Methods in Engineering*, 2015. Citado na página 59.

Raphael Chung-Wei Phan. Mini advanced encryption standard (mini-aes): a testbed for cryptanalysis students. *Cryptologia*, 26(4):283–306, 2002. Citado na página 24.

Bruno César Ribas. Um método de pré-processamento de fórmulas sat e pseudo-boolean baseado em técnicas de programação linear inteira mista. 2015. Citado na página 30.

Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. Citado na página 25.

Bart Selman, Hector J Levesque, David G Mitchell, et al. A new method for solving hard satisfiability problems. In *AAAI*, volume 92, pages 440–446, 1992. Citado na página 33.

Bart Selman, Henry Kautz, Bram Cohen, et al. Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability: Second DIMACS implementation challenge*, 26:521–532, 1993. Citado na página 33.

Claude Elwood Shannon. A mathematical theory of communication. *Bell System Tech. J*, 27:379–423, 1948. Citado na página 16.

Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999. Citado na página 13.

Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending sat solvers to cryptographic problems. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 244–257. Springer, 2009. Citado 2 vezes nas páginas 14 e 33.

Niklas Sorensson and Niklas Een. Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT*, 2005:53, 2005. Citado 3 vezes nas páginas 32, 47 e 48.

Zbigniew Stachniak and Anton Belov. Speeding-up non-clausal local search for propositional satisfiability with clause learning. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 257–270. Springer, 2008. Citado na página 33.

William Stallings and Daniel Vieira. *Criptografia e segurança de redes: princípios e práticas*. Pearson Prentice Hall, 2008. Citado na página 23.

G Tseitin. On the complexity ofderivation in propositional calculus. *Studies in Constrained Mathematics and Mathematical Logic*, 1968. Citado na página 34.

Taisy Silva Weber. Tolerância a falhas: conceitos e exemplos. *Apostila do Programa de Pós-Graduação–Instituto de Informática-UFRGS. Porto Alegre*, 2003. Citado na página 15.

Michael Whitman and Herbert Mattord. *Principles of information security*. Cengage Learning, 2011. Citado na página 13.