UNIVERSIDADE FEDERAL FLUMINENSE

GABRIEL CARDOSO DE CARVALHO

CRIPTOANÁLISE BICLIQUE APLICADA A CIFRAS DE BLOCO

NITERÓI

2019

UNIVERSIDADE FEDERAL FLUMINENSE

GABRIEL CARDOSO DE CARVALHO

CRIPTOANÁLISE BICLIQUE APLICADA A CIFRAS DE BLOCO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Algoritmos e Otimização

Orientador:
LUIS ANTONIO BRASIL KOWADA

NITERÓI

2019

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

D278c De carvalho, Gabriel
Criptoanálise Biclique Aplicada a Cifras de Bloco / Gabriel De carvalho ; Luis Kowada, orientador. Niterói, 2019.
66 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2019.

DOI: http://dx.doi.org/10.22409/PGC.2019.m.15022393700

1. Criptoanálise. 2. Cifras de Bloco. 3. Criptoanálise Biclique. 4. Serpent. 5. Produção intelectual. I. Kowada, Luis, orientador. II. Universidade Federal Fluminense. Instituto de Computação. III. Título.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

GABRIEL CARDOSO DE CARVALHO

Criptoanálise Biclique aplicada a cifras de blocos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Algoritmos e Otimização

Aprovada em 27 de Fevereiro de 2019.

BANCA EXAMINADORA

Luis Antonio Kowada

Prof. LUIS ANTONIO BRASIL KOWADA, UFF

Orientador

Prof. ANTONIO AUGUSTO ARAGÃO, UFF

Fábio Borges de Oliveira

Prof. FÁBIO BORGES DE OLIVEIRA, LNCC

Niterói

2019

Agradecimentos

Agradeço primeiramente aos meus pais e minha namorada por todo apoio que sempre me deram;

Ao meu professor orientador, Luis Kowada e à banca pelo apoio, pela disponibilidade e por aceitar o convite para avaliar esse projeto;

Aos meus amigos da UFF, tanto os que saíram quanto os que ainda estão, pelos momentos de descontração necessários durante a produção desse projeto;

À Universidade Federal Fluminense pela estrutura oferecida e ao Departamento de Computação.

À CAPES e FAPERJ que forneceram o auxílio financeiro durante o mestrado.

Resumo

A criptoanálise é uma das duas áreas da criptologia, sendo a outra a criptografia. Enquanto a criptografia foca os seus esforços na criação e melhoria de cifras, a criptanálise tenta encontrar vulnerabilidades na segurança em cifras. Um dos métodos criptoanalíticos mais recentes e eficientes é a Criptoanálise Biclique. Este ataque foi aplicado a diversas cifras, dentre elas, a mais utilizada cifra de bloco, o AES. Nesta dissertação, estudamos este método aplicado ao AES e o utilizamos para atacar a cifra Serpent. Para o AES, fizemos uma implementação prática do ataque de modo a comprovar os resultados teóricos. Esta implementação mostrou que, como esperado, a aplicação do ataque de maneira prática acaba por adicionar um custo maior de tempo do que o ataque teórico prevê. Além disso, produzimos os primeiros ataques com complexidade de tempo menor que uma busca exaustiva por todas as chaves possíveis da cifra Serpent. Se tratam de dois ataques, um utilizando uma biclique de dimensão 4 cobrindo as 4 últimas rodadas da cifra, e o outro usa uma biclique de dimensão 8 cobrindo as 3 últimas rodadas. Ambos conseguem ser cerca de duas vezes mais rápidos que uma busca exaustiva simples, apesar de precisarem de grandes quantidades de textos cifrados escolhidos.

Palavras-chave: Criptografia, Criptoanálise, Criptoanálise Biclique, AES, Rijndael, Serpent.

Abstract

Cryptanalysis is one of two areas of cryptology, the other one being cryptography. While Cryptography focuses its efforts on the creation and upgrade of ciphers, cryptanalysis tries to find security vulnerabilities in ciphers. One of the most recent and efficient cryptanalytic methods is the Biclique Cryptanalysis. This attack was applied to a variety of ciphers including the most used block cipher, the AES. In this work, we study this method applied to the AES and we utilize it to attack the Serpent cipher. For the AES, we implemented the attack in practice, so we can verify the theoretical results. The implementation showed that, as expected, utilization of the attack in practice ends up adding a higher time cost than the theoretical attack predicts. Moreover, we produce the first attacks with less time complexity than an exhaustive search for all the possible keys of the Serpent cipher. They are two attacks, one utilizing a 4-dimension biclique covering the last 4 rounds of the cipher, and the other uses an 8-dimension biclique covering the last 3 rounds of the cipher. Both are able to beat the basic exhaustive search by approximately a factor of two, although they need a high amount of chosen ciphertexts.

Keywords: Cryptography, Cryptanalysis, Biclique Cryptanalysis, AES, Rijndael, Serpent.

Lista de Figuras

Sumário

Capítulo 1

Introdução

A criptografia nasceu da necessidade de esconder informações de todas as pessoas com exceção daquela para qual a mensagem deveria ser entregue. Seu uso é historicamente ligado aos militares devido ao alto nível de confidencialidade das mensagens a serem passadas.

Existem dois tipos de sistemas criptográficos, ou cifras: sistemas simétricos e assimétricos. Os sistemas simétricos utilizam a mesma chave para cifrar e decifrar, enquanto os sistemas assimétricos, também conhecidos como de chave pública, utilizam chaves distintas, sendo uma delas conhecida por todos (Chave Pública) e a outra apenas por uma das partes (Chave Privada). Em geral, extrair a Chave Privada a partir da Chave Pública, requer saber resolver problemas matemáticos, para os quais não se conhece algoritmo eficiente (com complexidade polinomial).

Se tratando das cifras simétricas, os dois principais tipos são, as cifras de fluxo e cifras de bloco. As cifras de fluxo baseiam-se na geração de uma grande sequência de bits a partir de uma pequena sequência de bits de chave, e então, é feito o ou exclusivo (XOR) dessa sequência com o texto a ser encriptado[1], chamado de texto claro, produzindo assim, o texto cifrado. Já as cifras de bloco dividem o texto claro em blocos de tamanho fixo, que são cifrados para se tornarem textos cifrados de mesmo tamanho.

Cifras de bloco são utilizadas no dia-a-dia das pessoas comuns, garantindo a confidencialidade e protegendo sua privacidade. As mais famosas são o *Data Encryption Standard* (DES) [21] e o seu sucessor o *Advanced Encryption Standard* (AES). [22] O DES foi o padrão internacional de criptografia desde de 1977 [21], se mantendo o padrão por 25 anos, sendo finalmente substituído pelo AES em 2002. O AES é até hoje o padrão atual.

[1] Existem modos de operação que cifram a chave em vez do texto.

O DES é considerado um dos maiores motivos para a popularização do estudo da criptografia, antes restrito aos militares, na área acadêmica de forma geral, assim como do estudo da criptoanálise.

A criptoanálise (ou criptanálise) pode ser definida informalmente como a tentativa de se obter alguma informação secreta de determinada cifra, o que no contexto de cifras de bloco normalmente se traduz na tentativa de se obter a chave secreta usada por determinada cifra. O DES foi uma das primeiras cifras de bloco a sofrer extensiva criptanálise, o que produziu os dois métodos criptanalíticos mais conhecidos e usados: a Criptanálise Diferencial (CD) e a Criptanálise Linear (CL). A CL tenta aproximar a cifra atacada a uma expressão linear, de forma que alguns bits da chave possam ser encontrados, o que diminui o espaço de chaves possíveis a serem buscadas. Já a CD, compara o ou-exclusivo (XOR) de duas entradas com o XOR das duas saídas correspondentes e tenta derivar a chave da última rodada a partir daí. Ambos obtiveram sucesso em atacar o DES, ou seja, ambos são ataques que exigem menos computações do que uma busca exaustiva pela chave no DES.

O método estudado neste trabalho é o primeiro método a conseguir encontrar a chave do AES com complexidade de tempo menor do que uma busca exaustiva por todas as chaves possíveis [8], a chamada Criptanálise Biclique (CB). Esse ataque se baseia, entre outros métodos, na CD e é de extrema relevância atual devido ao fato de que é muito recente e pela sua eficiência contra cifras de bloco em geral, atacando diversas cifras como a IDEA [29], ARIA [11] e HIGHT [25].

O objetivo deste trabalho, portanto, é a aplicação da CB. As cifras escolhidas foram as finalistas do concurso AES, cujo vencedor foi a cifra Rijndael, posteriormente renomeada como AES. Os outros finalistas foram *Serpent*, *Twofish*, RC6 e *MARS* [36]. Todas elas são apresentadas com três tamanhos diferentes de chave: 128, 192 e 256 bits, pois este era um dos critérios do concurso. Outro critério importante é que os candidatos devem ser protegidos contra CD e CL e, portanto, todos os cinco o são.

Nossa Contribuição. As contribuições desta dissertação estão associadas à aplicação da Criptanálise Biclique nas cifras Rijndael e Serpent. A cifra Rijndael já foi alvo deste método. Por isso, para esta cifra foi feita uma implementação prática do ataque ao Rijndael-128 (ou seja, ao AES com chave de 128 bits) com o objetivo de validar na prática a complexidade de tempo estimada de maneira teórica. Para a cifra *Serpent*, propomos dois ataques para a versão *Serpent*-256, ambos mais rápidos que uma busca exaustiva com complexidades de tempo $2^{255,21}$ e $2^{255,45}$, respectivamente.

Estrutura do Texto. O texto da dissertação é feito de maneira a sequencialmente construir os blocos necessários para a compreensão do método, bem como da implementação e dos ataques. O Capítulo 2 introduz e formaliza os conceitos relacionados a cifras de bloco que são necessários durante o texto, também apresentando grande parte das notações necessárias. No Capítulo 3, como no anterior, são introduzidos e formalizados os conceitos e notações necessárias sobre criptanálise de maneira geral. Porém, além disso, as seções detalham o funcionamento dos métodos utilizados pela Criptanálise Biclique, como a CD e *Meet-in-the-Middle*. Finalizando a parte preparatória do texto, o Capítulo 4 explica a CB em si, descrevendo o algoritmo utilizado bem como as bases teóricas para aplicar os ataques. Os capítulos 5 e 6 condensam as contribuições deste trabalho. As maiores contribuições (Capítulo 6) são os primeiros ataques à versão completa da cifra *Serpent*-256 com complexidade de tempo menor que uma busca exaustiva pela chave, através de dois ataques separados, um utilizando uma biclique de dimensão 4 aplicada a 4 rodadas, enquanto o outro utiliza uma biclique de dimensão 8 aplicada a 3 rodadas. Como contribuição menor temos a validação prática das complexidades apresentadas por Bogdanov *et al.* (Capítulo 5) na forma de uma implementação do seu ataque no AES-128, conhecendo alguns bytes de chave para que fosse possível concluir os testes em tempo computacional (porém conhecer esses bytes não influencia o ataque de forma alguma). O Capítulo 7 conclui o trabalho.

Capítulo 2

Cifras de Bloco

2.1 Introdução

As cifras de bloco são um tipo de cifra simétrica. Portanto, antes de definir cifras de bloco precisamos da definição de cifra simétrica. Este capítulo é baseado, entre outras fontes, no livro de Stinson [38].

Suponha que um agente A deseja enviar uma mensagem P para um outro agente B através de um canal público inseguro (a internet por exemplo). Essa mensagem P é chamada *texto claro*. Para que apenas eles possam conhecer a mensagem, A e B combinam, de maneira segura, um segredo prévio K a qual A utilizará para esconder P, produzindo o *texto cifrado* C, e B utilizará para encontrar P a partir de C. O segredo K é chamado de *chave*. A Figura 2.1 ilustra a troca de informações entre A e B.

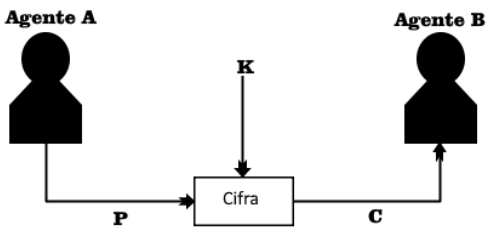


Figura 2.1: Agente A utiliza a chave K para esconder (cifrar) o texto claro P, transformando-o no texto cifrado C, e B utiliza K para obter P a partir de C.

Dessa forma, caso algum outro agente capture a mensagem C, ele não será capaz de obter o texto claro pois ele não conhece a chave previamente combinada de maneira segura.

A cifra simétrica (ou criptossistema simétrico) é basicamente o método que recebe P e K e retorna C, bem como o inverso, o método que recebe K e C e retorna P. Formalmente, temos a seguinte definição:

Uma *cifra simétrica* (ou *criptossistema simétrico*) é composta pela tupla $(\mathcal{P}, \mathcal{C}, \mathcal{K})$ e as funções E e D, onde:

1. $\mathcal{P}$ é um conjunto finito de todos os textos claros possíveis.
2. $\mathcal{C}$ é um conjunto finito de todos os textos cifrados possíveis.
3. $\mathcal{K}$, chamado de *espaço de chave*, é um conjunto finito de todas as chaves possíveis.
4. E é uma função do tipo $E : (\mathcal{P}, \mathcal{K}) \to \mathcal{C}$ e D é do tipo $D : (\mathcal{C}, \mathcal{K}) \to \mathcal{P}$, tais que $D(E(P, K), K) = P$, para todo $P \in \mathcal{P}$ e $K \in \mathcal{K}$.

Este tipo de cifra é muito utilizado no dia-a-dia dos usuários, muitas vezes de maneira transparente. Cifras simétricas são utilizadas de maneira a garantir a confidencialidade entre A e B.

Todo tipo de protocolo seguro na internet utiliza alguma forma de criptografia simétrica, como o S/MIME, protocolo de transferência de e-mail, o SSL/TLS, protocolo para a camada de aplicação da internet usualmente utilizado em conjunto com HTTPS, IPSec, para dar segurança para os protocolos IPv4 e IPv6, e muitos outros.

Podemos definir agora as *cifras de bloco* no contexto deste trabalho. Simplesmente são cifras simétricas onde $\mathcal{P} = \mathcal{C}$ e um texto que se deseja cifrar deve ser divido em blocos de tamanho fixo. Por exemplo, caso desejemos cifrar a frase "O HABITO FAZ O MONGE" (assumindo que todas as letras são maiúsculas e não há acento), utilizando uma cifra de bloco de tamanho 4 caracteres, ciframos separadamente: "O HA", "BITO", " FAZ", " O M", "ONGE". Este tipo de cifra, por cifrar sempre bloco a bloco, tem crescimento linear no tempo de cifração em relação ao tamanho do texto claro, uma vantagem de velocidade que faz com que ela seja muito atraente em contextos práticos.

A Seção 2.2 fala sobre os pontos de design de cifras de bloco mais utilizados com foco nas cifras estudadas para este trabalho.

2.2 Estrutura

Esta seção tem o objetivo de definir e identificar as estruturas base de cifras de bloco modernas que são importantes para a compreensão da terminologia utilizada nos capítulos seguintes.

Primeiramente, podemos definir uma *cifra de bloco* como sendo uma cifra simétrica onde $P \in \mathcal{P}$ e $C \in \mathcal{C}$ são strings de bits de tamanho fixo p e c respectivamente. No geral cada cifra será também identificada pelo tamanho da chave $K \in \mathcal{K}$ que também é uma *string* de bits de tamanho fixo k. Logo temos 2^p possíveis textos claros, 2^c possíveis textos cifrados bem como 2^k chaves diferentes. Via de regra, o tamanho do bloco se refere à p e $c = p$.

Existem cifras com tamanho fixo de chave em sua descrição, como o DES que é uma cifra com blocos de tamanho 64 e chave de tamanho 56. Outras cifras, como os finalistas do concurso AES, têm blocos de tamanho fixo, mas aceitam mais de um tamanho de chave. Por exemplo, o Rijndael é uma cifra com blocos de tamanho 128, porém aceita chaves de 128, 192 e 256 bits. Nesse caso, nos referimos a uma cifra explicitando seu tamanho de chave, chamando a versão do Rijndael com chave de 128 bits de Rijndael-128.

As cifras de bloco modernas são geralmente do tipo *iterativa*. Isso significa que a cifra é formada pela composição de uma função menor por um certo número de vezes. Essa função menor é chamada *função de rodada*, e cada aplicação da função é uma *rodada*. A função de rodada tem como entrada um *estado interno* da cifra (inicialmente se trata do texto claro para a primeira rodada, passando a ser a saída da rodada anterior nas rodadas seguintes) e uma *chave de rodada* ou *subchave*. As chaves de cada uma das rodadas são geradas a partir da chave secreta, através do *gerador de subchaves*.

As funções de rodada são formadas por operações que adicionam *confusão e difusão* ao processo de encriptação. De acordo com Katz *et al.* [27], a confusão tenta tornar a relação entre a chave e o texto cifrado o mais complexo possível, enquanto a difusão rearranja e distribui os bits da mensagem, evitando que redundâncias do texto claro alcancem o texto cifrado.

Um utensílio muito utilizado em cifras modernas, empregado por exemplo no AES, é o *whitening*. *Whitening* se trata do XOR de uma subchave ao estado interno independente da rodada. Geralmente é empregado ao texto claro antes da primeira rodada ou após o término da última rodada para gerar o texto cifrado.

Por exemplo, a Figura 2.2 mostra a aplicação desse conceito na cifra AES-128, que é composta por 10 rodadas, *i.e.*, 10 repetições da função de rodada, e 11 subchaves, uma para cada rodada e uma para o *whitening* inicial.

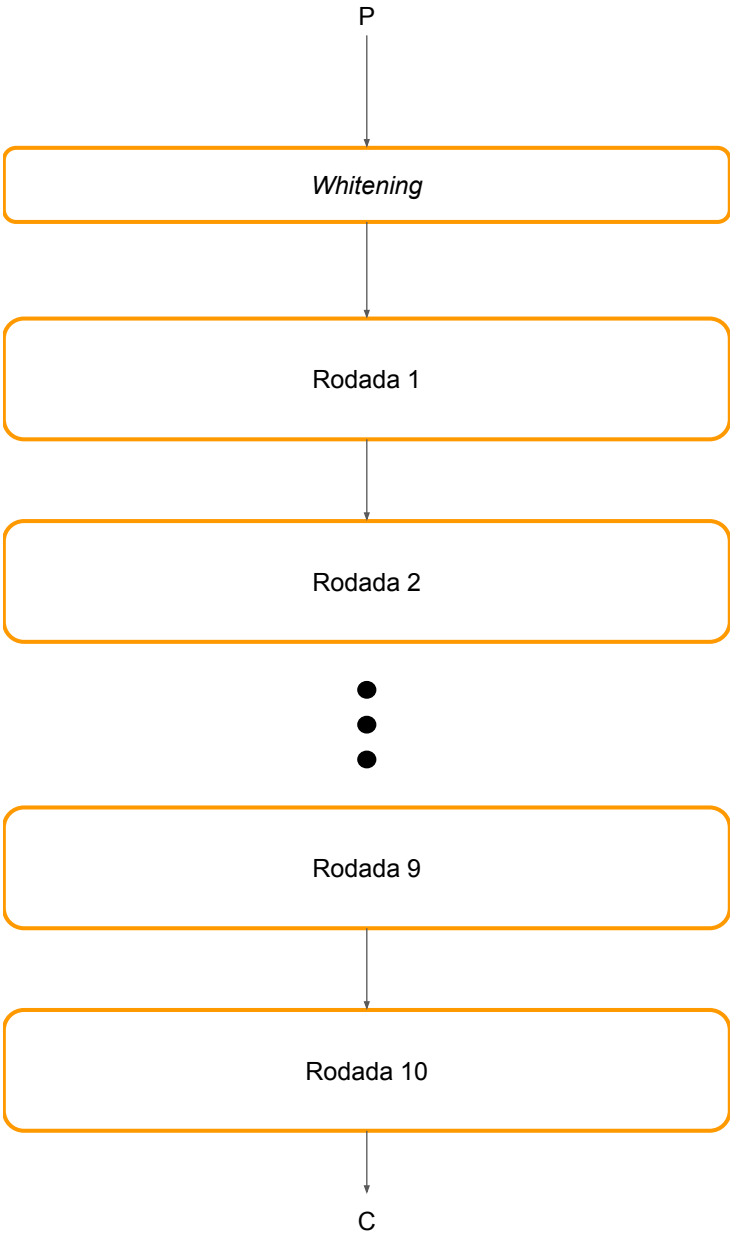


Figura 2.2: As 10 rodadas do AES-128 de maneira simplificada

Formalmente, uma cifra de bloco B é definida pela 4-tupla (p, c, k, r) e pelas funções $E : (\{0,1\}^p, \{0,1\}^k) \to \{0,1\}^c$ e $D : (\{0,1\}^c, \{0,1\}^k) \to \{0,1\}^p$, onde $D(E(P,K),K) = P$ para todo texto claro P e chave K. Os valores de p, c e k são os tamanhos em bits do texto claro, do texto cifrado e da chave, respectivamente. E é a função de encriptação e D a de decriptação. Comumente $p = c$, e nesse caso dizemos que o *tamanho do bloco* é p.

Dadas as definições e a terminologia que serão utilizadas, é válido citar os dois tipos mais comuns de cifras de bloco: as *Substitution Permutation Networks (SPNs)* e as cifras *Feistel*. Figuras 2.3 mostram a estrutura típica destes dois tipos de cifra. Os mais famosos representantes são o AES e o DES para as SPNs e cifras Feistel, respectivamente.

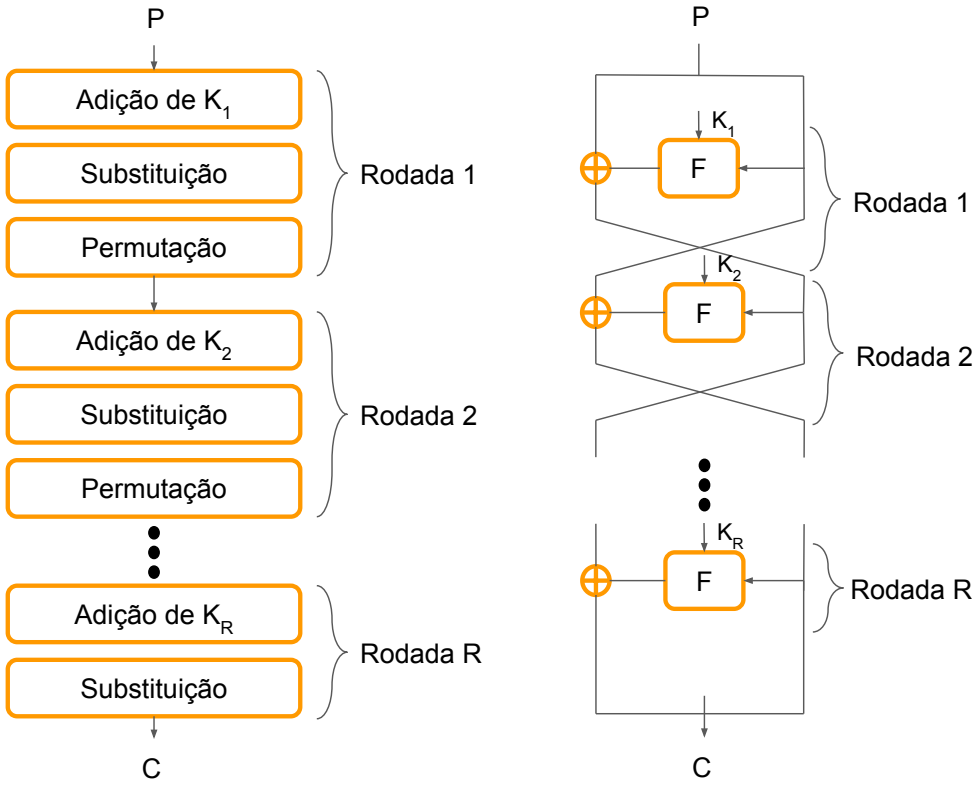


Figura 2.3: Típica estrutura das SPNs (esquerda) e cifras Feistel (direita) de R rodadas

Em uma SPN, cada rodada da encriptação se trata de três passos: XOR com a subchave, fase de substituição e fase de permutação. O primeiro passo é simplesmente o XOR do estado recebido como entrada com a subchave da rodada. O segundo passo é a fase de substituição, onde o estado atual consulta uma ou mais tabelas (conhecidas como *S-boxes*) e substitui o valor do estado pelo valor na tabela. Na fase de permutação, bits do estado atual trocam de posição com outros bits do estado. Essa é a estrutura mais básica, podendo haver outros passos, geralmente lineares, sendo apenas a fase de substituição a fase não-linear. A última rodada substitui a fase de permutação por mais um XOR com a subchave. Isso faz com que a decriptação seja igual a encriptação, desde que se utilize a inversa das S-boxes e sejam feitas algumas modificações nas subchaves [38].

Nas cifras Feistel, o texto claro é inicialmente dividido em dois. Uma das metades passa pela *Função F*. Essa função pode ser construída de várias maneiras, inclusive como uma pequena SPN, e utiliza a chave da rodada. A saída da função F executa um XOR com a outra metade. Em seguida, essa metade é invertida com a metade anterior (a que

foi entrada da função F) para finalizar a rodada. Neste tipo a encriptação e decriptação são quase idênticas, sendo necessário somente inversão na ordem das subchaves [27].

Pelo fato deste tipo de cifra não ser baseado em problemas matemáticos complexos para garantir sua segurança (como é o caso das cifras de chave pública), mas sim na concatenação de funções fracas para tornar a cifra forte, a *criptanálise* se faz necessária para procurar falhas de segurança nelas. O Capítulo 3 foca em descrever superficialmente os tipos de ataques úteis para a compreensão da Criptanálise Biclique e o Capítulo 4, com o objetivo de apresentar os conceitos necessários, aprofunda somente os pontos mais importantes.

Capítulo 3

Criptanálise de Cifras de Bloco

A Criptanálise, no contexto de cifras de bloco, é considerada como a tentativa de se encontrar a chave de um criptossistema conhecendo somente os textos claros, textos cifrados e o criptossistema em si. Este ramo da criptologia tem grande importância fornecendo *certificados de fraqueza* para cifras que forem teoricamente quebradas. Uma cifra é dita *teoricamente quebrada* quando é encontrado um ataque que tenha complexidade de tempo menor que uma busca exaustiva (ou força bruta). *Complexidade de tempo*, nesse caso, se refere à quantidade de encriptações de textos claros usados pelo ataque.

Existem vários modelos de ataque na área da criptanálise. Os modelos são listados a seguir, partindo do modelo em que o atacante tem menos informação para o modelo em que ele tem mais informação:

- Ataque com apenas textos cifrados (*ciphertext only attack*)

  Neste ataque, o adversário tem conhecimento apenas de um ou mais textos cifrados. Ataques deste tipo são eficientes na prática.

- Ataque de texto claro conhecido (*known plaintext attack*)

  Neste ataque, o adversário tem conhecimento de um ou mais textos claros e seus correspondentes textos cifrados. O maior exemplo deste ataque é a busca exaustiva, que utilizada apenas um par.

- Ataque de textos claro escolhido (*chosen plaintext attack*)

  Neste ataque, assume-se que o adversário obteve acesso temporário ao mecanismo de encriptação, e por isso pode decidir quais textos claros serão encriptados e obter seus correspondentes textos cifrados.

- Ataque de textos cifrado escolhido (*chosen ciphertext attack*)

  Neste ataque, assume-se que o adversário obteve acesso temporário ao mecanismo de decriptação, e por isso pode decidir quais textos cifrados serão decriptados e obter seus correspondentes textos claros.

Dessa forma, dependendo do ataque feito à cifra, ainda que seja mais rápida que a força bruta, não necessariamente causará impacto prático na cifra, principalmente porque o ataque pode precisar de um volume muito grande de pares de texto claro/cifrado.

Por exemplo, a Criptanálise Diferencial do DES [6] é um ataque de texto claro escolhido. Esse ataque fornece um certificado de que o DES não é seguro em um cenário em que o atacante obteve acesso ao mecanismo de encriptação por tempo o suficiente para produzir 2^{47} textos claros escolhidos, pois nesse caso ele consegue quebrar o DES com 2^{37} execuções da cifra, muito abaixo da busca exaustiva (2^{56}). Este cenário não é muito prático, e portanto este ataque não comprometeu seu uso.

Por outro lado, o ataque *Meet-in-the-Middle*, que é explicado mais a diante, feito ao 3DES [18] é um ataque de texto claro conhecido que somente precisa de um texto claro e seu correspondente cifrado para diminuir a complexidade de recuperação da chave secreta de 2^{168} computações do 3DES para 2^{112}, usando cerca de 2^{56} bytes de memória. Nesse caso, apesar de ser necessária grande quantidade de memória, o impacto prático é evidente, pois o modelo de ataque utilizado é razoável na prática e ele consegue reduzir a complexidade por um fator de $2^{168}/2^{112} = 2^{56}$. Por isso a segurança do 3DES é considerada equivalente a 2^{112}.

Como mostrado nos exemplos anteriores, existem três propriedades importantes associadas à um ataque: as complexidades de tempo, memória e dados.

A complexidade de tempo, como explicitado no início do capítulo, se refere à quantas encriptações da cifra atacada são necessárias para o ataque. A complexidade de memória (ou de espaço) se refere à quantidade de textos claros/cifrados que precisam ser mantidos em memória. Em relação aos dados, trata-se da quantidade de pares de textos claros/cifrados que estão disponíveis para o ataque, ainda que não estejam armazenados. Neste texto, sempre que não for explicitado a que tipo de complexidade que se refere, a complexidade é de tempo.

Temos agora informação suficiente para descrever um método criptanalítico para cifras de bloco com essas informações, por exemplo, a Criptanálise Linear do DES [35], uma cifra com blocos de 64 bits e chave de 56 bits, é um ataque de texto claro conhecido, que

tem complexidade de tempo e de dados de 2^{43} e memória insignificante. Nesse caso, o ataque é $2^{56}/2^{43} = 2^{13}$ vezes mais rápido que força bruta.

As seções seguintes visam introduzir os métodos de criptanálise que a Criptanálise Biclique, o verdadeiro foco deste trabalho, se baseia, de maneira a facilitar seu entendimento.

3.1 Criptanálise Diferencial

A Criptanálise Diferencial (CD) foi criada por Biham e Shamir em 1991 [5] sendo o primeiro ataque a ser bem-sucedido sobre a versão completa do DES [6] em 1992. Este ataque, apesar de precisar de 2^{47} textos claros escolhidos e seus respectivos cifrados, utiliza uma quantidade insignificante de memória e 2^{37} de tempo. Como o DES foi publicado como padrão em 1975 e efetivado em 1977 [21], foram necessários 15 anos de pesquisa para que o DES fosse teoricamente quebrado pela CD. Atualmente, qualquer nova cifra publicada deve ser imune à CD, a chamada "segurança comprovável", desde o concurso AES [37].

Além de sua importância histórica sendo bem-sucedido no ataque ao DES, a CD se tornou base para vários outros ataques além de ter tido inúmeras variações. Alguns exemplos de variações mais bem-sucedidas são Diferencial Impossível [4], Diferencial Truncada ou Parcial [31], Diferencial Linear [33], entre outros. Algumas novas técnicas que se baseiam na CD são Ataque Square [13], Bumerangue [40], entre outros incluindo a Criptanálise Biclique.

A Criptanálise Diferencial baseia-se na análise de como a *diferença* Δ_P entre um par de textos claros afeta a diferença Δ_C dos correspondentes textos cifrados . A diferença, em geral, é um XOR entre o par de textos. As *diferenças* $\Delta_P = P \oplus P'$ e $\Delta_C = C \oplus C'$ formam a *diferencial* $\Delta_P \xrightarrow[E]{K} \Delta_C$, onde E é a cifra e K a chave. Além de analisar os textos claros e cifrados, é possível também analisar a diferença entre os estados intermediários da cifra, ou seja, analisar como a diferença nos textos claros afeta os estados intermediários após a primeira rodada, depois como a diferença após a primeira rodada afeta a segunda e assim por diante. Observar apenas os textos claros e cifrados se trata de utilizar uma *diferencial*, enquanto observar as diferenças nos estados internos se trata de usar *características* ou *trilhas*.

3.2 Ataque *Meet-in-the-Middle*

O Ataque *Meet-in-the-Middle* (MITM) foi criado por Diffie e Hellman em 1977 [18] como um argumento contra o 3DES, já que este diminui a segurança da chave de 168 bits para 112 bits. Haja visto que o maior empecilho à praticidade dos métodos de criptanálise modernos está na grande quantidade de pares de texto claro/cifrado necessários para a realização dos ataques, o MITM poderia ser considerado prático neste sentido, já que precisa de poucos pares (sua versão mais simples utiliza apenas um). Porém, o que impede a aplicação deste método em muitas cifras é a restrição que ele impõe à chave que não é facilmente cumprível como veremos mais a frente nesta seção.

Além da aplicação no 3DES, recentemente foi mostrado que este ataque pode ser utilizado em várias cifras. Em geral, com sucesso apenas em versões com quantidade reduzida de rodadas, em vez da cifra completa, como por exemplo o IDEA [16], DES [19], AES [15, 17], KTANTAN [9], entre outras.

O método em si precisa de um par de texto claro/cifrado e escolher um ponto intermediário v na cifra tal que para cada chave, uma parte dela interfira somente nas primeiras rodadas da cifra até v e o resto dos bits da chave interfira somente nas rodadas seguintes. Então, encripta-se o texto claro até v de todas as formas possíveis e decripta-se o texto cifrado de todas as formas possíveis até o ponto intermediário. A chave secreta faz com que esse ponto seja igual para ambos. Portanto, as chaves candidatas são as que tem resultados coincidentes em v.

Formalmente temos o seguinte para o caso mais simples deste tipo de ataque. Sejam a cifra $E = g \circ f$, P um texto claro, $C = E_K(P)$ o respectivo texto cifrado e K a chave secreta de k bits. Assuma que $K = (K^1, K^2)$ tal que, $E_K(P) = g_{K^2}(f_{K^1}(P))$, onde os tamanhos em bits de K^1 e K^2 são k_1 e k_2 respectivamente, e $k_1 + k_2 = k$. Então, calculamos todos os valores de $f_{K_i^1}(P)$ e $g^{-1}_{K_j^2}(C)$, onde K_i^1 e K_j^2 são as diversas possibilidades para K^1 e K^2 respectivamente. A partir dai, buscamos por um par (i, j) tal que

$$f_{K_i^1}(P) = g^{-1}_{K_j^2}(C), \tag{3.1}$$

para $0 \leq i \leq 2^{k_1} - 1$ e $0 \leq j \leq 2^{k_2} - 1$. A chave secreta será portanto $K = (K_i^1, K_j^2)$ encontrada em tempo da ordem de $2^{k_1} + 2^{k_2}$ (utilizando tabelas *hash* [27]) em vez da busca exaustiva que levaria $2^{k_1} \times 2^{k_2}$.

A cifra 3DES foi atacada por este método mais simples do MITM, visto que o 3DES se trata da composição de três DES e a chave secreta é $K = (K^1, K^2, K^3)$, onde $k_1 =$

$k_2 = k_3 = 56$ e $k = 168$. Sendo assim nosso $g_{K^1,K^2} = DES_{K^1}(DES_{K^2})$ e $f_{K^3} = DES_{K^3}$. Fazendo como mostrado anteriormente, encontraremos a chave secreta em tempo $2^{112} + 2^{56} \approx 2^{112}$, sendo este valor muito abaixo da busca exaustiva 2^{168}.

O ataque MITM necessita que a restrição acima ($K = (K^1, K^2)$ tal que, $E_K(P) = g_{K^2}(f_{K^1}(P))$) seja satisfeita. Fica claro, portanto, que o maior empecilho para a aplicação deste ataque é encontrar dois espaços de chave com esta propriedade. O Capítulo 4 demonstra como a Criptanálise Biclique consegue contornar este problema.

3.3 Chaves Relacionadas

Um ataque de chaves relacionadas (*related-keys*) pode ser definido, de acordo com Kelsey *et al.* [28], como o ataque onde o adversário conhece as encriptações de um dado texto claro P utilizando uma chave secreta K, e de outras chaves derivadas de K, $K' = f(K)$, onde f é o relacionamento entre elas. Existem dois tipos de ataques a partir daí, *relacionamento de chaves conhecido ou escolhido*, porém em ambos os casos as chaves K' e K em si não são conhecidas.

Este tipo de ataque é conhecido como muito poderoso desde sua primeira utilização por Biham [2] para atacar as cifras LOKI [10] e LUCIFER [20], porém as condições necessárias para um ataque genérico de chaves relacionadas tornam ele extremamente impraticável. Este tipo de ataque teórico é muito utilizado atualmente tendo atacado as cifras XTEA e GOST [32], o AES [7] e outros.

Entretanto, sua eficiência na prática não pôde ser ignorada quando em 2001 Fluhrer *et al.* [23] utilizou ataques de chaves relacionadas para atacar o protocolo *WEP*, antigo padrão para redes sem fio. Esse ataque obrigou a revisão do protocolo o que culminou na sua alteração pelos protocolos *WPA* e *WPA2*.

Entre as diferentes maneiras de se definir o relacionamento entre as chaves, existe o uso da criptanálise diferencial, onde o relacionamento é dado por uma *diferencial de chaves relacionadas*. Nesse caso, dada a diferencial de chaves relacionadas Δ_K, temos que $K \oplus K' = \Delta_K$. Porém, caso queiramos um conjunto com k chaves relacionadas à chave secreta K, basta que criemos k diferenciais Δ_i. Dessa forma basta fazer $K_i = K \oplus \Delta_i$ com $0 \leq i < k$ para obter as k chaves relacionadas.

A Criptanálise Biclique utiliza este tipo de relacionamento para suas chaves, como o Capítulo 4 mostra. Este ataque demonstra uma forma de se utilizar o ataque de chaves

relacionadas em um modelo de *chave única*, onde o atacante deseja recuperar a chave secreta e onde apenas a encriptação e decriptação com esta chave são permitidas.

Capítulo 4

Criptanálise Biclique

A Criptanálise Biclique foi criada por Bogdanov *et al.* [8] baseada na utilização de bicliques como estrutura inicial em um ataque às funções hash SKEIN-256 e SHA-2 [30]. Bogdanov *et al.* utilizaram esse novo método de criptanálise para produzir os primeiros ataques mais rápidos que uma busca exaustiva pela chave às versões completas do AES, tanto com chaves de 128, 192 e 256 bits.

Após o sucesso inicial desse ataque muitas outras cifras também foram atacadas, como as cifras IDEA [29], ARIA [11], TWINE [12], PRESENT [1, 26], HIGHT [25] e SQUARE [34]. É válido constatar que esses ataques foram feitos no espaço de sete anos já que o ataque original ao AES foi feito em 2011. Esse grande uso mostra o enorme potencial da Criptanálise Biclique.

Esse ataque se baseia em combinar uma estrutura chamada biclique, que será explicada mais a frente, com o MITM, utilizando a biclique para reduzir o número de rodadas da cifra em que o MITM será aplicado. Ao criar uma biclique em m rodadas de uma cifra de r rodadas (sejam as m primeiras rodadas ou as m últimas), se torna possível utilizar O MITM nas $r-m$ rodadas restantes. Porém, para contornar a restrição de chave do MITM, Bogdanov *et al.* criaram a técnica *Matching* com Pré-computações (MP). Esta técnica pré-computa e armazena os estados internos e subchaves das $r-m$ rodadas para algumas chaves, de modo que possam ser reutilizadas na computação das outras chaves. Maiores detalhes sobre bicliques e sobre a MP são dados nas seções 4.2 e 4.4, respectivamente.

4.1 Algoritmo

O ataque Biclique segue uma estrutura bem simples de 3 passos: construção da estrutura inicial, obtenção de textos e MITM. Porém, antes desses passos é necessário que o adversário particione o espaço de chave em grupos com 2^{2d} chaves para algum d. Cada grupo de chaves é associado a uma matriz K de tamanho $2^d \times 2^d$, onde cada elemento $K[i, j]$ da matriz corresponde a uma chave do grupo. Se assumirmos que o espaço da chave é de 2^k possibilidades, então teremos 2^{k-2d} grupos.

É preciso que a cifra E que está sob ataque seja composta de três subcifras $E = f \circ g \circ h$ e deve-se escolher entre as subcifras h ou f para aplicar a estrutura inicial. A partir daí, os três passos são aplicados a cada grupo de chaves até que a correta seja encontrada.

Os passos são descritos abaixo. Para a demonstração do algoritmo do ataque suponha que o atacante tenha escolhido a subcifra f para aplicar a estrutura inicial. O algoritmo seria o mesmo para h, porém simétrico.

1. **Construção da estrutura inicial.** Construa uma estrutura inicial com 2^d estados internos S_j e 2^d textos cifrados C_i que respeitem a seguinte restrição:

$$\forall i, j : S_j \xrightarrow[f]{K[i,j]} C_i. \tag{4.1}$$

   Uma possível estrutura que corresponde a essa descrição é a Biclique. As bicliques são explicadas na Seção 4.2.

2. **Obtenção de textos.** Para cada um dos C_i obtenha os correspondentes P_i utilizando o oráculo de decriptação (o oráculo conhece a chave secreta).

$$\forall i : C_i \xrightarrow[E^{-1}]{\text{oráculo de decriptação}} P_i. \tag{4.2}$$

3. **MITM.** Teste para cada chave $K[i, j]$ do grupo se

$$\exists i, j : P_i \xrightarrow[g \circ h]{K[i,j]} S_j. \tag{4.3}$$

   Se $K[i, j]$ é a chave secreta, então ela mapeia P_i para S_j. Logo, se existe i, j que satisfaça 4.3, então $K[i, j]$ é uma candidata à chave. A técnica utilizada pelos autores originais para testar o Passo 3 é chamada *Matching* com Pré-computações e é descrita na Seção 4.4.

O artigo original utiliza a biclique como estrutura para o Passo 1, o que originou o nome do ataque. Para o Passo 3, qualquer tipo de técnica que verifique a condição 4.3 pode ser utilizada, porém Bogdanov *et al.* em seu artigo original criaram o método MP.

4.2 Bicliques

Uma Biclique é uma estrutura vinda da teoria dos grafos. A biclique é um grafo, no qual os vértices (ou nós) podem ser particionados em dois conjuntos, de forma que, não há arestas entre dois vértices do mesmo conjunto e há arestas entre todos os pares de vértices de conjuntos distintos.

Nas bicliques utilizadas na Criptanálise, os vértices estão associados a estados internos da cifra ou textos claros/cifrados e as arestas são as chaves utilizadas para se obter o estado interno a partir do texto claro/cifrado. As bicliques do ataque original possuíam as partições de mesmo tamanho, 2^d. Este parâmetro d é chamado de *dimensão* da biclique. Artigos posteriores já utilizam bicliques com partições de tamanhos distintos, como Ghosh [24] com as estrelas e Tao [39] utilizando uma biclique com partições de tamanho 2^8 e 2^{16} para melhorar o ataque original ao AES.

Quando mostramos o algoritmo do ataque utilizamos a biclique ao final da cifra, porém é possível aplicá-la às rodadas iniciais, nesse caso alterando o uso de textos cifrados por textos claros.

Seja f a subcifra que mapeia um estado interno S da cifra para o texto cifrado C usando a chave K, ou seja, $f_K(S) = C$. A Biclique de dimensão d aplicada à subcifra f é a 3-tupla $\{\{S_j\}, \{C_i\}, \{K[i,j]\}\}$ onde $0 \leq i, j \leq 2^d - 1$ e

$$\forall i, j : f_{K[i,j]}(S_j) = C_i. \tag{4.4}$$

Essa é de fato a definição exata da estrutura necessária para o Passo 1 do algoritmo básico do ataque. O obstáculo para o uso da biclique está em como ela deve ser construída, já que as partições devem obedecer a condição 4.4. A maneira mais intuitiva de se abordar o problema é fixar os conjuntos $\{S_j\}$ e $\{C_i\}$ e derivar a partir daí as 2^{2d} chaves correspondentes. Outra forma seria fixar o conjunto $\{S_j\}$ e as chaves $K[i,j]$ e derivar o conjunto $\{C_i\}$ deles. O problema é que ambos os métodos custarão pelo menos 2^{2d} aplicações da função f. Porém, Bogdanov *et al.* encontraram uma forma capaz de construir a biclique com apenas 2^{d+1} aplicações de f utilizando diferenciais de chaves relacionadas.

4.3 Diferenciais de Chaves Relacionadas

Para encontrar bicliques com complexidade menor que a abordagem mais direta (onde derivamos as 2^{2d} chaves a partir dos 2^d textos cifrados e estados internos) devemos fazer o oposto: escolher as chaves e derivar os textos cifrados e estados internos a partir delas. Para isso, é necessário que as chaves da biclique sejam escolhidas segundo diferenciais especificas. Neste trabalho somente trabalhamos com *diferenciais de chaves relacionadas independentes* e por isso as chamamos somente de diferenciais de chaves relacionadas.

Seja $K[0,0]$ a *chave base* que mapeia o estado interno S_0 para o texto cifrado C_0. Essa computação é chamada de *computação base* e é denotada por

$$S_0 \xrightarrow[f]{K[0,0]} C_0.$$

Definimos a seguir os conjuntos de 2^d diferenciais, as *diferenciais-*Δ_i e *diferenciais-*∇_j, a partir de diferenciais de chaves relacionadas.

A diferencial que pertence ao conjunto diferenciais-Δ_i mapeia a diferença de entrada 0 para a diferença de saída Δ_i usando a diferença de chave Δ_i^K, onde $\Delta_0 = \Delta_0^K = 0$.

$$0 \xrightarrow[f]{\Delta_i^K} \Delta_i, \text{com } \Delta_0 = \Delta_0^K = 0 \tag{4.5}$$

Já a diferencial que pertence ao conjunto diferenciais-∇_j mapeia a diferença de entrada ∇_j para a diferença de saída 0 usando a diferença de chave ∇_j^K, onde $\nabla_0 = \nabla_0^K = 0$.

$$\nabla_j \xrightarrow[f]{\nabla_j^K} 0, \text{com } \nabla_0 = \nabla_0^K = 0 \tag{4.6}$$

Se nenhum dos dois conjuntos compartilharem componentes não-lineares (no caso de cifras de bloco modernas normalmente se tratam das *S-boxes*) então é possível combinar ambos os conjuntos em um novo conjunto chamado *diferenciais-*(Δ_i, ∇_j)

$$\nabla_j \xrightarrow[f]{\nabla_j^K \oplus \Delta_i^K} \Delta_i. \tag{4.7}$$

Não compartilhar componentes não-lineares significa que nenhuma parte de um estado interno ou subchave de f afetados pelas diferenciais-Δ_i são também afetados pelas diferenciais-∇_j e vice-versa.

Por definição, a computação base $(S_0, C_0, K[0,0])$ conforma com as diferenciais com-

binadas, logo é possível substituí-las nessas diferenciais, obtendo

$$S_0 \oplus \nabla_j \xrightarrow[f]{K[0,0]\oplus\nabla_j^K\oplus\Delta_i^K} C_0 \oplus \Delta_i. \tag{4.8}$$

Podemos agora chamar $S_j = S_0 \oplus \nabla_j$, $C_i = C_0 \oplus \Delta_i$ e $K[i,j] = K[0,0] \oplus \Delta_i^K \oplus \nabla_j^K$, *i.e.*, a definição de uma biclique de dimensão d. Construir a biclique desta forma requer apenas 2^{d+1} computações de f, já que podemos apenas escolher as diferenciais de chave Δ_i^K e ∇_j^K e gerar separadamente as diferenciais-Δ_i e -∇_j.

4.4 *Matching* com Pré-computações

Como dito anteriormente neste capítulo, qualquer técnica de MITM pode ser utilizada para o Passo 3 do algoritmo. Contudo, qualquer técnica direta desse tipo precisa satisfazer a restrição da chave (essa restrição é descrita na Seção 3.2) a fim de que o ataque seja eficiente.

O método *Matching* com Pré-computações (MP) busca fazer o MITM onde as variáveis intermediárias v são apenas parte de um estado interno em vez de ser um estado interno completo. Isso implica que, vindo de ambos os lados do ataque, o cálculo de v não necessita que toda a subcifra seja calculada e sim apenas as partes que influenciam v. O mesmo é válido para as chaves a serem utilizadas.

Para tanto, o MP faz a pré-computação e o armazenamento em memória das computações completas de v (ou seja, todos os estados internos e subchaves) para algumas das chaves, e somente recomputa as partes diferentes das já armazenadas para o restante das chaves.

Seja a cifra $E = f \circ g \circ h$ atacada pela Criptanálise Biclique onde a biclique está construída na subcifra f. O adversário pré-computa e armazena as 2^d *computações completas* de h e g, *i.e.*, armazena todos os estados internos e subchaves de h e de g, até a variável v, variando somente i para a subcifra h e somente j para subcifra g, como

$$\forall i : P_i \xrightarrow[h]{K[i,0]} v_i^1 \quad \text{e} \quad \forall j : v_j^2 \xleftarrow[g]{K[0,j]} S_j. \tag{4.9}$$

A seguir resta apenas recomputar as partes que diferem dos valorem armazenados. O custo deste método depende das propriedades de difusão da cifra em questão, tanto em relação à geração das chaves quanto dos estados internos, podendo variar bastante de

cifra para cifra.

4.5 Complexidades

Como dito no Capítulo 3, existem três propriedades importantes associadas à um ataque: as complexidades de tempo, memória e dados. Esta seção é dedicada à análise desses três tipos de complexidade.

Primeiramente, a complexidade de memória, nesse caso, é afetada apenas por guardar os 2^d estados internos S_j e textos cifrados C_i, e também as 2^d computações completas das subcifras g e h. Isso inclui armazenar cada estado interno e cada subchave utilizada em g e em h. Dessa forma, é possível notar que o armazenamento das pré-computações é muito mais custoso e domina a complexidade. Portanto, a complexidade de memória neste ataque depende do tamanho dos estados internos, do tamanho das subchaves e da quantidade de rodadas em g e h da cifra atacada. Se chamarmos a soma dos tamanhos em bits dos estados e subchaves de g e h de S_{bits}, temos que a complexidade de memória é $2^d(S_{bits})$.

Em seguida observamos a complexidade de dados. Na Criptanálise Biclique, o que define a quantidade de pares de textos necessários para o ataque são as diferenciais-Δ_i. Assuma que a biclique foi construída nas últimas rodadas da cifra atacada. Seja b a quantidade de bits de C_0 afetados pelas diferenciais-Δ_i de uma determinada biclique. A quantidade de pares necessários para o ataque será 2^b. Nesse caso, o ataque será de texto cifrado escolhido, pois serão necessários os respectivos textos claros de todos os textos cifrados que se diferenciam pelas diferenciais-Δ_i somente, *i.e.*, se as diferenciais-Δ_i afetam apenas b bits, então os outros bits dos textos cifrados são os mesmos de C_0. Em suma, o ataque precisa de um par de texto claro para cada um dos 2^b possíveis textos cifrados. Caso a biclique fosse construída nas primeiras rodadas, seria um ataque de texto claro escolhido.

Por fim, o foco principal das contribuições deste trabalho, a complexidade de tempo. A complexidade de tempo deste ataque se trata da complexidade de tempo C_{iter} de uma iteração do ataque vezes a quantidade de iterações I, obtemos $C_{tempo} = IC_{iter}$. Seja k o tamanho em bits da chave e d a dimensão da biclique em questão. Portanto , $I = 2^{k-2d}$, já que esta é a quantidade de diferentes grupos de chave como mostrado na Seção 4.1. A complexidade de uma iteração é a soma do tempo gasto construindo a biclique $C_{biclique}$ com o tempo gasto na MP C_{MP}. Obtemos a equação $C_{tempo} = 2^{k-2d}(C_{biclique} + C_{MP})$. A Seção

4.3 conclui que a complexidade da construção de uma biclique utilizando diferenciais de chaves relacionadas é 2^{d+1} computações de f (sejam r_f o número de rodadas cobertas por f e r o total de rodadas da cifra), enquanto a Seção 4.4 mostra que a complexidade C_{MP} se dá pela soma do tempo gasto na pré-computação $C_{precomp}$, na recomputação C_{recomp} e na resolução dos falso positivos C_{falpos}. Ao fim obtemos a equação

$$C_{tempo} = 2^{k-2d}(C_{biclique} + C_{precomp} + C_{recomp} + C_{falpos}). \tag{4.10}$$

Sabemos que $C_{precomp} = 2^d((r_g + r_h)/r)$, onde r_g é a quantidade de rodadas cobertas por g e r_h por h pois na pré-computação são feitas 2^d computações de g e h. C_{falpos} dependerá do tamanho em bits do v escolhido para a MP e da dimensão da biclique, sendo aproximadamente $C_{falpos} = 2^{2d}/2^{|v|}$. Por fim, C_{recomp}, o maior influenciador na complexidade final do ataque, será menor que 2^{2d} computações de g, sendo proporcionalmente mais próximo deste valor quanto mais difusas forem a cifra e seu gerador de subchaves. De maneira mais completa temos

$$C_{tempo} = 2^{k-2d}(2^{d+1}(r_f/r) + 2^d(r_g/r) + 2^2 d/2^{|v|} + C_{recomp}). \tag{4.11}$$

O cálculo de C_{recomp} é comumente feito pela razão entre a quantidade de *S-boxes* recalculadas pela quantidade de *S-boxes* normalmente necessárias para calcular a cifra. Isso se dá porque a consulta às *S-boxes* representa o custo dominante no processo de computação das cifras de bloco de maneira geral.

Capítulo 5

Advanced Encryption Standard (AES)

5.1 Descrição

A criptanálise Biclique original do AES foi feita por Bogdanov *et al.* [8] e, portanto, utilizaremos aqui não somente os parâmetros do ataque como também notação para representar o AES utilizada no artigo original.

O AES-128 tem estados internos de 128 bits, bem como sua chave secreta. Tanto os estados internos da cifra quanto as subchaves são representados por matrizes 4 x 4 bytes, onde a ordenação dos bytes é dada abaixo:

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

A função de encriptação do AES-128 é

$$E_K = AK_{10} \circ SR \circ SB \circ \bigcirc_{i=1}^{9}(R_i) \circ AK_0(P),$$

onde AK_i é a função AddRoundKey com a subchave K^i e

$$R_i = AK_i \circ MC \circ SR \circ SB,$$

onde MC é a função MixColumns, SR é a função ShiftRows e SB é a função SubBytes [14]. As subchaves K^i são geradas a partir de $K^0 = K$. A única parte não linear da cifra é a função SubBytes que é utilizada tanto na encriptação quanto no gerador de subchaves.

Por fim, cada estado interno será numerado pós AK e pós SB, SR, MC, portanto

$P = \#0$, após AK_0 temos $\#1$, depois de SB, SR, MC temos $\#2$ e ao final de R_1 temos $\#3$ e assim por diante.

A descrição detalhada de todas as funções e do gerador de subchaves (gerador de subchaves) se encontra na Seção 5.3.

5.2 O Ataque

Para o ataque, Bogdanov *et al.* [8] construíram uma biclique de dimensão $d = 8$ utilizando como chave base a chave K^8. Portanto o espaço da chave deve ser dividido em $2^{k-2d} = 2^{128-2*8} = 2^{112}$ grupos da seguinte forma.

K_0^8	K_4^8	K_8^8	0
0	K_5^8	K_9^8	K_{13}^8
K_2^8	K_6^8	K_{10}^8	K_{14}^8
K_3^8	K_7^8	K_{11}^8	K_{15}^8

Onde os bytes 1 e 12 de K^8 são zerados e todos os outros 14 podem variar. As Δ_i^K e ∇_j^K são do tipo:

0	0	i	i
j	0	j	0
0	0	0	0
0	0	0	0

Além disso, o E_K do AES-128 foi dividido em $E_K = f_K \circ t_K \circ r_K$, onde $r_K(P) = \#5$, $t_K(\#5) = \#15$ e $f_K(\#15) = C$ para um dado texto claro P e seu correspondente texto cifrado C. O ataque procede da seguinte forma para cada grupo de chaves:

Passo 1: Construção da Biclique: Primeiramente escolhemos um C_0 para a computação base. Este C_0 será utilizado por todo o ataque, alterando apenas a chave base $K[0,0]$ que será uma por grupo de chaves, e S_0 é calculado a partir de ambos. Fazemos $f_{K[0,0]}^{-1}(C_0) = S_0$. A partir de S_0 e $K[0,0]$, calculamos todos os C_i ao fazer $f_{K[0,0]\oplus\Delta_i^K}(S_0) = C_i$ e a partir de C_0 e $K[0,0]$ calculamos todos os S_j ao fazer $f_{K[0,0]\oplus\nabla_j^K}^{-1}(C_0) = S_j$.

Passo 2: Geração dos textos claros: No artigo original é assumido que todos os possíveis 2^{88} pares estão a disposição para serem consultados e são considerados complexidade

de dados, porém na implementação os pares necessários devem ser calculados utilizando um oráculo de decriptação no Passo 2.

Para efeitos de comparação, implementamos o passo 3 de duas formas distintas: a) simplesmente calculando os 2^{2d} estados a partir dos P_i e comparando com S_j; b) utilizando MP.

Passo 3 a): MITM: Nesse caso simplesmente checamos se

$$\exists i, j : \ t_{K[i,j]}(r_{K[i,j]}(P_i)) = S_j.$$

Caso exista temos um candidato com grande probabilidade de ser a chave correta. Caso contrário passamos para o próximo grupo de chaves.

Passo 3 b): MP: Nesse caso, escolhemos $v = \#5_{12}$, ou seja, a variável intermediária é o byte 12 do estado $\#5$. Escolhido v fazemos a Pré-computação, que computa e armazena em memória os estados internos, as subchaves e os v necessários para a fase de Recomputação. Na Recomputação computamos apenas as partes necessárias da cifra para calcular v (*i.e.*, as partes que ainda não foram calculadas na Pré-computação) tanto adiante ($r_{K[i,j]}(P_i) = v$), quanto na direção oposta ($v = t^{-1}_{K[i,j]}(S_j)$).

O ataque utilizando o passo 3.a) tem complexidade de tempo:

$$C_{total} = 2^{112}(2^7 + 2^{15,58}) = 2^{127,59} \tag{5.1}$$

computações do AES-128, onde considera-se que a complexidade de $C_{biclique} = 2^9(2, 5/10) = 2^7$ já que a biclique cobre aproximadamente duas rodadas e meia do AES, e $C_{MITM} = 2^{16}(7, 5/10) = 2^{15,58}$. No caso do ataque com MP a complexidade é:

$$C_{total} = 2^{112}(2^8 + 2^{14,14} + 2^8) = 2^{126,18} \tag{5.2}$$

computações do AES-128, onde a complexidade de $C_{biclique} + C_{precomp} \leq 2^8$ e o número médio de falsos positivos é também 2^8 e $C_{recomp} = 2^{16} * (3,475/12,5) = 2^{14,14}$, onde $12,5$ é o total de chamadas à função SB que o AES-128 faz e $3,475$ é número de vezes que SB precisa ser recomputada, de acordo com o artigo original. As ilustrações da biclique e das recomputações do ataque original estão no Apêndice B.

Além das complexidades de tempo, também é apresentado um limitante superior para o uso de memória que é 2^8 computações completas de $t_K \circ r_K$ do estágio de Pré-computação bem como a complexidade de dados é igual a 2^{88} devido ao fato de serem possíveis apenas

2^{88} diferentes textos cifrados para o ataque.

Nossa implementação visa comparar os resultados práticos com os resultados teóricos possibilitando a validação das complexidades tanto de tempo quanto de memória. Na prática levamos em consideração o passo 2 na complexidade, fazendo com que ela se torne $2^{126,20}$ para o caso da Biclique com MP e se mantem $2^{127,59}$ para a Biclique com MITM.

5.3 Implementação

A implementação foi feita em C na plataforma Windows e compilado com o GCC Mingw. Todos os testes foram feitos utilizando um processador AMD FX-8320 $3.5GHz$. Por questões de viabilidade do tempo de ataque, a maior quantidade de chaves buscadas foi de 2^{32}, ou seja, 12 bytes da chave eram conhecidos e portanto não precisavam ser testados. Como os ataques de força bruta com 12 bytes fixos levaram em média 4,4 horas para serem concluídos, o ataque com 11 bytes fixos levaria cerca de 1126 horas ou, de maneira equivalente, quase 47 dias. Devido à falta de tempo, sua execução se tornou inviável. A implementação está publicada no github no sítio *https://github.com/Gabriel-Cardoso-de-Carvalho/CRYPTANALYSIS-AES128*.

As subseções seguintes detalham a implementação enquanto os resultados e as comparações com as complexidades teóricas são feitos na seção 5.4.

5.3.1 O AES

5.3.1.1 SB (*SubBytes*)

SB é a parte não linear da cifra. Se trata unicamente de uma *lookup table*, *i.e.*, uma tabela de consulta. SB^{-1} é a tabela inversa de SB. A tabela de SB se encontra no Apêndice A.

$$\mathrm{B} = \begin{array}{|c|c|c|c|} \hline B_0 & B_4 & B_8 & B_{12} \\ \hline B_1 & B_5 & B_9 & B_{13} \\ \hline B_2 & B_6 & B_{10} & B_{14} \\ \hline B_3 & B_7 & B_{11} & B_{15} \\ \hline \end{array}, \qquad \mathrm{SB(B)} = \begin{array}{|c|c|c|c|} \hline SB(B_0) & SB(B_4) & SB(B_8) & SB(B_{12}) \\ \hline SB(B_1) & SB(B_5) & SB(B_9) & SB(B_{13}) \\ \hline SB(B_2) & SB(B_6) & SB(B_{10}) & SB(B_{14}) \\ \hline SB(B_3) & SB(B_7) & SB(B_{11}) & SB(B_{15}) \\ \hline \end{array}$$

5.3.1.2 SR (*ShiftRows*)

SR é uma permutação onde a linha i $(0 \leq i \leq 3)$ rotaciona i vezes para a esquerda. Na implementação, por motivos de otimização, é feito o swap direto em vez de repetir a mesma função de rotação i vezes. Sua inversa SR^{-1} simplesmente rotaciona para a direita ao invés da esquerda.

$$\mathrm{B} = \begin{array}{|c|c|c|c|} \hline B_0 & B_4 & B_8 & B_{12} \\ \hline B_1 & B_5 & B_9 & B_{13} \\ \hline B_2 & B_6 & B_{10} & B_{14} \\ \hline B_3 & B_7 & B_{11} & B_{15} \\ \hline \end{array}, \qquad \mathrm{SR(B)} = \begin{array}{|c|c|c|c|} \hline B_0 & B_4 & B_8 & B_{12} \\ \hline B_5 & B_9 & B_{13} & B_1 \\ \hline B_{10} & B_{14} & B_2 & B_6 \\ \hline B_{15} & B_3 & B_7 & B_{11} \\ \hline \end{array}$$

5.3.1.3 MC (*MixColumns*)

MC é uma transformação linear que é aplicada a cada coluna da matriz individualmente, resultando em uma nova coluna onde cada byte depende de todos os bytes da coluna antiga. Isso é feito através da multiplicação no corpo $GF(2^8)$ de cada coluna pela matriz M. A inversa MC^{-1} de MC é a mesma função utilizando a matriz inversa M^{-1}.

Na implementação utilizamos *lookup tables* para cada multiplicação possível (por 2, 3, 9, 11, 13 e 14) de forma a uniformizar a complexidade tanto de MC quanto MC^{-1}.

$$M = \begin{array}{|c|c|c|c|} \hline 2 & 3 & 1 & 1 \\ \hline 1 & 2 & 3 & 1 \\ \hline 1 & 1 & 2 & 3 \\ \hline 3 & 1 & 1 & 2 \\ \hline \end{array}, \qquad M^{-1} = \begin{array}{|c|c|c|c|} \hline 14 & 13 & 11 & 9 \\ \hline 9 & 14 & 11 & 13 \\ \hline 13 & 9 & 14 & 11 \\ \hline 11 & 13 & 9 & 14 \\ \hline \end{array}$$

5.3.1.4 Geração das subchaves

A função AK_i simplesmente faz o XOR dos bytes da chave K^i com os bytes do estado interno. Temos um total de 11 chaves K^i $(0 \leq i \leq 10)$ e se faz necessária a implementação de mais de um gerador de subchaves. Precisamos de:

- O original, que gera as 11 subchaves a partir da chave secreta K (para o AES-128 completo);
- Um gerador que gere as chaves K^9 e K^{10} a partir de K^8 (para a construção da biclique);

- Um gerador que gere as subchaves K^0, K^1, ..., K^7 a partir de K^8 de maneira inversa (para MP e MITM);

O gerador de subchaves funciona da seguinte maneira. Seja $K^i = (W_0^i, W_1^i, W_2^i, W_3^i)$, onde W_j^i representa a coluna j da subchave K^i e $K^0 = K$. Consequentemente temos,

$$K^{i+1} = (W_0^{i+1}, W_1^{i+1}, W_2^{i+1}, W_3^{i+1}),$$

onde

$$W_j^{i+1} = W_j^i \oplus W_{j-1}^{i+1} \text{ para } 1 \leq j \leq 3$$

e

$$W_0^{i+1} = W_0^i \oplus ROT(SB(W_3^i)) \oplus RCON_i \text{ para } j = 0,$$

onde ROT é uma função que rotaciona os bytes de uma coluna para cima uma vez e $RCON_i$ são i constantes.

A função ROT é implementada de forma semelhante ao SR, porém apenas para uma coluna, enquanto $RCON_i$ são constantes tabeladas. As tabelas se encontram no Apêndice A. Nas outras colunas são aplicadas apenas XOR's entre seus valores.

5.3.2 Algoritmo do Ataque

O pseudo-código dos ataques segue abaixo no algoritmo 1. Todos os três ataques seguem a mesma estrutura, alterando somente a função ATAQUE(...) dependendo do ataque em questão.

A função ATAQUE(...) é substituída por uma das funções: $Brute\_Force(P, K', C_0)$, $Biclique\_MITM(K, K', C_0)$ e $Biclique\_MP(K, K', C_0)$ que são respectivamente os ataques de força bruta, biclique usando MITM básico e biclique utilizando MP. A força bruta precisa somente de um par de textos claro/cifrado para teste das chaves, enquanto a criptanálise biclique vai gerar vários pares à partir de um C_0 e da chave secreta K utilizada pelo oráculo de decriptação. Por isso o C_0 é parâmetro de todas as funções enquanto P é apenas no caso da força bruta.

A função $Proxima\_chave(K', i)$ gera a próxima chave que será testada (no caso da força bruta) ou a chave base do próximo grupo de chaves que será testado (no caso das bicliques). O *overhead* causado por ela é mínimo, se tratando do incremento em 1 de um byte (inicialmente zerado) até que o mesmo se torne 0 novamente, caso em que o próximo byte é incrementado e assim por diante. A função fixa_bytes(K) fixa os bytes de K que

são conhecidos em K', permitindo que somente os bytes desconhecidos variem, e a função gera_estado() produz um estado interno com 16 bytes aleatórios.

Dados: Número de bytes conhecidos f, Chave secreta K
Resultado: Retorna quando encontrar a chave
$K' \leftarrow$ fixa_bytes(K);
$C_0 \leftarrow$ gera_estado();
$total \leftarrow 2^{112-(8*f)}$;
se *força_bruta* **então**
  $P \leftarrow E^{-1}(K, C_0)$;
fim
para $i \leftarrow 0$ **até** $total$ **faça**
  se $ATAQUE(...)$ **então**
    escreve("Sucesso");
    retorna
  fim
  $K' \leftarrow$Proxima_chave(K',i);
fim
escreve("Fracasso");
retorna

Algoritmo 1: Estrutura da implementação dos ataques.

A função $Brute\_Force(P, K', C_0)$ é simples: fazemos um estado temporário T receber a computação $E_K(P) = T$ e comparamos T com C_0. Se forem iguais retorna 1, senão retorna 0.

A implementação de ATAQUE(...) para os ataques com bicliques segue o algoritmo 2, para ambas as implementações com exceção do PASSO_3 que é diferente para cada caso, sendo eles $MITM(K', P_i, S_j)$ e $MP(K', P_i, S_j)$ para a biclique que utiliza MITM e para a biclique que utiliza MP, respectivamente. Ambos os casos são explicados na Subseção 5.3.3.

Dados: Chave secreta K, Chave base K' e texto cifrado base C_0
Resultado: Retorna 1 se a chave correta for encontrada e 0 caso contrário
C_i, S_j, P_i, onde $0 \leq i, j < 2^8$;
$C_i, S_j \leftarrow$ constroi_biclique(K',C_0);
$P_i \leftarrow$ gera_P_i(C_i,K);
retorna PASSO_3(K',P_i,S_j);

Algoritmo 2: Estrutura da implementação da criptanálise biclique.

A função $constroi\_biclique(K', C_0)$ computa $S_0 = f^{-1}_{K'}(C_0)$, criando a computação base. Seja $K[i,j] = K' \oplus \Delta^K_i \oplus \nabla^K_j$. Depois ela faz $S_j = f^{-1}_{K[0,j]}(C_0)$ e $C_i = f_{K[i,0]}(S_0)$ como mostra o algoritmo 3. Já a função $gera\_P_i(C_i, K)$ simplesmente decripta os 2^8 textos cifrados C_i, usando $E^{-1}(C_i) = P_i$.

Dados: Chave base K' e texto cifrado baseC_0
Resultado: Retorna os vetores C_i e S_j com 2^8 posições cada
$S_0 \leftarrow f^{-1}_{K'}(C_0)$;
para $i, j \leftarrow 0, 0$ **até** $2^8, 2^8$ **faça**
  $C_i \leftarrow f_{K[i,0]}(S_0)$;
  $S_j \leftarrow f^{-1}_{K[0,j]}(C_0)$;
fim
retorna C_i, S_j;

Algoritmo 3: Função $constroi\_biclique(K', C_0)$.

5.3.3 Biclique com MITM e com MP

A função $MITM(K', P_i, S_j)$, uma das duas possibilidades do PASSO_3(K',P_i,S_j), é simplesmente uma busca exaustiva entre P_i e S_j em vez de ser entre P_i e C_i, *i.e.* em vez de testar $E_{K[i,j]}(P_i) = C_i$, a função testa se $t_{K[i,j]}(r_{K[i,j]}(P_i)) = S_j$. Dessa forma, fazemos $S_{temp} = t_{K[i,j]}(r_{K[i,j]}(P_i))$ e checamos se $S_{temp} = S_j$ para cada um dos 2^{16} possíveis pares (i, j). Caso seja verdade para algum par retorna 1, caso contrário retorna 0.

No segundo caso, a função $MP(K', P_i, S_j)$ chama duas funções, primeiro a função Pré-computação(K', P_i, S_j) e retorna o resultado da função Recomputação(K', P_i, S_j).

A Pré-computação é mostrada no algoritmo 4. Se trata de fazer 2^8 chamadas às funções r e t^{-1}, de maneira a computar e salvar em variáveis globais (representadas por

RK_i^r e RK_j^t) todas subchaves usadas nas computações de r e t^{-1} com as 2^8 chaves $K[i,0]$ e $K[0,j]$; bem como salvar todos os estados internos dessas computações (representadas por precomp_i^r e precomp_j^t) relevantes para a fase de Recomputação.

Dados: Chave base K', vetor de textos claros P_i e vetor de estados S_j
$S_0 \leftarrow f_{K'}^{-1}(C_0)$;
para $i, j \leftarrow 0, 0$ **até** $2^8, 2^8$ **faça**
  $RK_i^r, \text{precomp}_i^r \leftarrow r_{K[i,0]}(P_i)$;
  $RK_j^t, \text{precomp}_j^t \leftarrow t_{K[0,j]}^{-1}(S_j)$;
fim

Algoritmo 4: Função Pré-computação(K', P_i, S_j)

Utilizando as chaves e computações já calculadas, é feita a Recomputação somente das partes que são afetadas pela diferença entre as chaves pré-computadas utilizadas em r e t^{-1} com as que faltam ser computadas. Para r temos que computar as partes afetadas pela diferença entre as chaves $K[i,0]$ e $K[i,j]$, enquanto para t^{-1} a diferença entre $K[0,j]$ e $K[i,j]$.

Portanto, primeiramente devemos recalcular as partes afetadas das chaves $K[i,j]$ a partir de RK_i^r e RK_j^t para finalmente recalcular as partes afetadas dos estados internos de r e t^{-1}, a partir de precomp_i^r e precomp_j^t, de modo a alcançar a variável $v = \#5_{12}$ e testar se $r_{K[i,j]}(P_i) = t_{K[i,j]}^{-1}(S_j)$. Se a expressão for verdadeira, $K[i,j]$ é uma chave candidata e portanto testamos se $t_{K[i,j]}(r_{K[i,j]}(P_i)) = S_j$ é verdadeira. Caso seja, $K[i,j]$ é com grande probabilidade a chave secreta.

5.4 Resultados

Considerando a inviabilidade de testar a criptanálise Biclique no AES com a chave completa desconhecida, para fazer testes comparativos com a busca exaustiva pela chave, partimos do pressuposto que conhecemos alguns bytes da chave. Foram feitas 5 buscas exaustivas com 12 bytes conhecidos da chave, já que este ataque é o mais demorado. Para a Criptanálise Biclique em ambas as formas foram feitos 10 ataques com 12 bytes da chave conhecidos. Foram feitos 100 experimentos para as versões com 13 e 14 bytes conhecidos. A tabela abaixo mostra os resultados dos experimentos em relação à complexidade de tempo. Os experimentos foram feitos assumindo que a chave correta é sempre a última a ser buscada, pois dessa forma podemos comparar o tempo, haja visto que todos os ataques

aqui apresentados sempre encontram a chave correta.

	12 Bytes	13 Bytes	14 Bytes
Busca Exaustiva	4,41h	61,40s	265ms
Biclique MITM	3,48h	47,75s	190ms
Biclique MP	1,98h	27,90s	115ms

Tabela 5.1: A tabela mostra a quantidade de tempo médio necessário para completar os experimentos para o ataque com 12, 13 e 14 bytes fixos de chave.

Primeiramente olhamos para o caso em que 12 bytes são conhecidos. Observamos que a Busca Exaustiva equivale a $2^{128-96} = 2^{32}$ computações do AES-128, pois conhecemos 96 bits da chave. Consequentemente, 2^{32} computações do AES-128 levam em média 4,41h com desvio padrão 0,12h e variância 0,014h. Proporcionalmente a Biclique MITM equivale a $2^{31,65}$ em média e a Biclique MP equivale a $2^{30,84}$ computações em média[1]. Comparando com o resultado teórico, tivemos um aumento de $2^{31,65}/2^{31,59} = 2^{0,06} = 1,04$ vezes para o ataque Biclique com MITM, ou seja, 4% a mais que o esperado. No caso da Biclique com MP, tivemos um aumento de $2^{30,84}/2^{30,20} = 2^{0,64} = 1,56$ vezes, ou seja, 56% mais lento que o esperado.

Vemos também que a escalabilidade em relação à quantidade de bytes conhecidos acontece de maneira bastante direta onde o aumento de tempo entre um ataque com x bytes conhecidos é cerca de 256 vezes maior que o tempo do ataque com $x + 1$ bytes conhecidos para todos os casos.

Portanto, podemos notar que o *overhead* adicionado pela construção da biclique é baixo, aproximadamente 4% a mais, visto que a biclique com MITM também faz a construção. Dessa forma, o *overhead* restante é devido à MP e é bem alto, ocasionando um aumento de cerca de 52%, após descontar o *overhead* da construção da biclique. Por outro lado, continua sendo mais de duas vezes mais rápido que a Busca Exaustiva.

Em relação ao uso de memória, de acordo com os dados do processo informados pelo Windows, a busca exaustiva utiliza cerca de 2.380KB, enquanto as Criptanálises Biclique gastam 2.730KB e 2.880KB aproximadamente, para os ataques com MITM e com MP respectivamente. Ao final, temos um gasto menor que 3MB de memória, o que é irrisório comparado à memória disponível nas máquinas atuais.

[1] A variância e o desvio padrão de todos os outros experimentos são insignificantes

5.5 Análise da Implementação do Ataque ao AES

Esta Seção apresentou a implementação da Criptanálise Biclique e da força bruta pura no AES-128 com até 12 bytes fixos da chave para diminuir o custo computacional da força bruta. A partir daí comparamos os resultados de tempo e memória com os resultados teóricos apresentados pelos autores originais do ataque [8] de maneira a validá-los e compreender o impacto na complexidade de tempo do ataque na prática.

De fato, como esperado, o *overhead* associado à Criptanálise Biclique, principalmente na fase de Recomputação, aumenta bastante a complexidade de tempo em relação à estimativa teórica. Quanto à complexidade de memória, ela é fixa, quase em sua totalidade associada à fase de Pré-computação, já que todos os cálculos de chaves e a computação completa de 2^9 computações de g devem ser guardados em memória. Por outro lado, o total de uso de memória é de aproximadamente 3MB, o que é bem baixo para os padrões atuais de tamanho de memória RAM.

Entretanto, a Criptanálise Biclique continua mais de duas vezes mais rápida que a força bruta pura. Dessa forma, o ataque permanece sólido mesmo no contexto prático.

Futuramente, além de estudar maneiras de otimizar o código para diminuir o *overhead*, melhorias podem ser feitas com a paralelização dessa implementação em CPU e GPU, de forma a possibilitar o ataque com menos bytes conhecidos da chave, além da generalização dessa implementação de forma a criar um *framework* capaz de aplicar o ataque à cifras de bloco genéricas.

Capítulo 6

Serpent

Neste capítulo são apresentados os dois ataques que propomos para a cifra *Serpent*, um utilizando uma biclique de dimensão 4 e outro de dimensão 8. A de menor dimensão consegue cobrir quase 4 rodadas completas e por isso conseguimos obter complexidade de tempo menor que o outro ataque (que cobre quase 3 rodadas completas). Por outro lado, a biclique de maior dimensão precisa de muito menos pares de texto claro/cifrado para que o ataque seja aplicado.

Começamos o capítulo com a descrição da cifra, depois sequencialmente apresentamos os dois ataques e ao final comparamos os dois.

6.1 Descrição da Cifra *Serpent*

Serpent é uma SPN de 32 rodadas. Cada estado interno e cada subchave tem 128 bits, divididos em 4 palavras de 32 bits, enquanto a chave secreta pode ter 128, 192 ou 256 bits [3]. Chamamos as versões da cifra com chaves de 128, 192 e 256 bits respectivamente de *Serpent*-128, *Serpent*-192 e *Serpent*-256.

Cada rodada i da cifra consiste de 3 camadas: a adição da chave da rodada (ou subchave da rodada) chamada AK_i, a Fase de Substituição SB_i e a transformação linear L. A exceção é a última rodada que substitui a transformação linear por mais uma adição da chave. As rodadas variam de 0 e 31, e temos 33 subchaves derivadas da chave secreta numeradas de K^0 a K^{32}.

Existem também duas permutações: uma inicial feita antes da primeira rodada e uma final após a última. Porém elas não adicionam nenhum tipo de segurança na cifra e podem ser ignoradas para propósitos de criptanálise [3].

É chamado de estado interno todo resultado de alguma operação da cifra, bem como o texto claro P e o texto cifrado C. Cada estado interno é representado por $\#j$, onde $0 \leq j \leq 96$, $\#0 = P$ e $\#96 = C$. Cada estado $\#j$ pode ser graficamente representado por uma matriz 4×8 de quadrados, onde cada quadrado representa meio byte e cada linha representa uma palavra de 32 bits. Chamamos cada meio byte de *half byte*. Os *half bytes* são numerados do mais à esquerda para o mais à direita e então para baixo para a próxima palavra. As palavras são numeradas de cima para baixo. O h-ésimo *half byte* do estado S é denotado por S_h. A seguir temos a representação gráfica de um estado interno da cifra.

W_0	h_0	h_1	h_2	h_3	h_4	h_5	h_6	h_7
W_1	h_8	h_9	h_{10}	h_{11}	h_{12}	h_{13}	h_{14}	h_{15}
W_2	h_{16}	h_{17}	h_{18}	h_{19}	h_{20}	h_{21}	h_{22}	h_{23}
W_3	h_{24}	h_{25}	h_{26}	h_{27}	h_{28}	h_{29}	h_{30}	h_{31}

A fase de substituição SB_i consulta uma de 32 diferentes *S-boxes* para cada palavra, usando esta *S-box* para cada *half byte* da palavra em paralelo, dependendo da rodada. A rodada i utiliza a *S-box* B_i. As *S-boxes* estão descritas no Apêndice C.

A Transformação Linear L se trata da aplicação de uma série de rotações, *shifts* e XORs entre as palavras do estado interno atual. Sejam W_0, W_1, W_2 e W_3 as 4 palavras de um estado S. As seguintes operações são efetuadas na ordem em que aparecem:

1. $W_0 <<< 13$. A palavra W_0 é rotacionada 13 bits para a esquerda.
2. $W_2 = W_2 <<< 3$. A palavra W_2 é rotacionada 3 bits para a esquerda.
3. $W_1 = W_1 \oplus W_0 \oplus W_2$. É feito o XOR das palavras W_1, W_0 e W_2.
4. $W_3 = W_3 \oplus W_2 \oplus (W_0 << 3)$. É feito o XOR das palavras W_3, W_2 e a palavra W_2 com o *shift* de 3 bits para a esquerda.
5. $W_1 = W_1 <<< 1$. A palavra W_1 é rotacionada 1 bit para a esquerda.
6. $W_3 = W_3 <<< 7$. A palavra W_3 é rotacionada 7 bits para a esquerda.
7. $W_0 = W_0 \oplus W_1 \oplus W_3$. É feito o XOR das palavras W_0, W_1 e W_3.
8. $W_2 = W_2 \oplus W_3 \oplus (W_1 << 7)$. É feito o XOR das palavras W_2, W_3 e a palavra W_1 com o *shift* de 7 bits para a esquerda.

9. $W_0 = W_0 <<< 5$. A palavra W_0 é rotacionada 5 bit para a esquerda.

10. $W_2 = W_2 <<< 22$. A palavra W_2 é rotacionada 22 bits para a esquerda.

Enfim, temos o gerador de subchaves. Ele recebe como entrada a chave secreta de 128, 192 ou 256 bits (dependendo da versão a ser utilizada) e gera 33 subchaves de 128 bits. A entrada do gerador é sempre uma chave de 256 bits. Portanto, no caso das chaves menores, o bit seguinte é setado em um e todos os outros em zero, por exemplo, no caso da chave de 128 bits temos o bit 128 sendo 1 e todos os bits de 129 a 255 sendo 0. A entrada tem 8 palavras indexadas de w_{-8} a w_{-1}. Então, é calculada a pré-chave, 132 palavras indexadas de w_0 a w_{131}, da seguinte forma

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus i \oplus \phi) <<< 11 \tag{6.1}$$

onde ϕ é a parte fracionária da proporção áurea $(\sqrt{5}+1)/2$ ou $0x9e3779b9$ em hexadecimal.

A partir da pré-chave, geramos as 133 palavras das subchaves. A palavra $k_i = SBox_{(3-(i \mod 33)) \mod 32}(w_i)$, por exemplo, $k_{100} = SBox_2(w_{100})$. Finalmente, a subchave K^i é formada pelas palavras $k_{4i}, k_{4i+1}, k_{4i+2}$ e k_{4i+3}, e $0 \leq i \leq 32$.

Informações mais detalhadas sobre o *design* da cifra *Serpent* podem ser encontradas na proposta original [3].

6.2 O Ataque com Biclique de dimensão 4

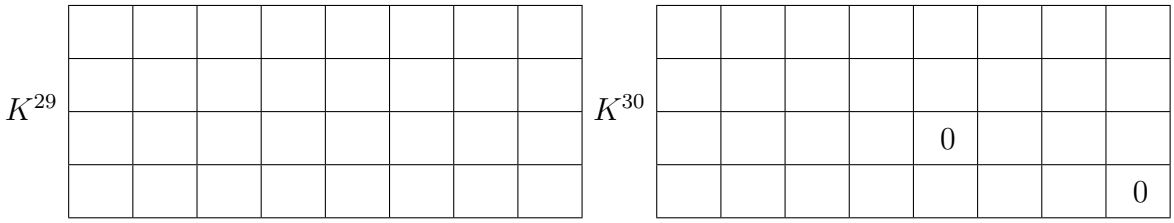
6.2.1 Particionamento das Chaves

O primeiro passo para o ataque é particionar as chaves em grupos. Como dito na Seção 4.1, a quantidade de grupos de chaves é igual a 2^{k-2d}, onde d é a dimensão da biclique usada e k a quantidade de bits da chave. Conforme será mostrado na Seção 6.2.2, a biclique do nosso ataque tem dimensão 4. Este ataque é sobre *Serpent*-256, logo as 2^{256} chaves possíveis serão particionadas em 2^{248} grupos de chaves. Cada grupo é representado por uma chave base.

Cada chave base define um grupo de 2^8 chaves, e cada iteração do ataque irá testar um desses grupos.

Neste caso, usamos as subchaves K^{29} e K^{30} para definir esses grupos como mostrado abaixo, já que duas subchaves em sequência são suficientes para gerar todas as outras. Os

half bytes K_{20}^{30} e K_{31}^{30} são setados em zero, enquanto todos os outros *half bytes* de ambos K^{29} e K^{30} irão variar a cada iteração gerando uma nova chave base.

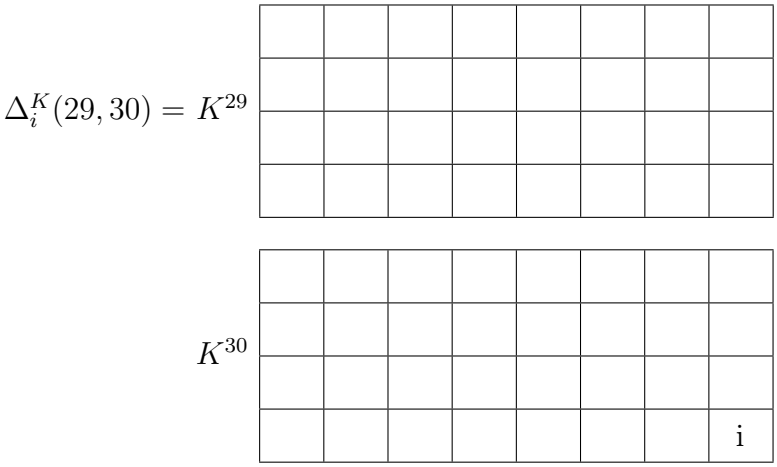


As 2^8 chaves do grupo são testadas graças a este particionamento. Isso acontece pois o *half byte* K_{31}^{30} é diretamente dependente da diferença de chave Δ_i^K utilizada no ataque, e K_{20}^{30} é indiretamente dependente da diferença de chave ∇_j^K, o que faz com que estes *half bytes* variem por todas as possibilidades. A chave base de um grupo é chamada $K[0,0]$ e umahttps://www.overleaf.com/project/5be42f21656b5a6b48cc5c15 chave deste grupo é $K[i,j] = K[0,0] \oplus \Delta_i^K \oplus \nabla_j^K$.

6.2.2 Biclique de dimensão 4 Cobrindo 4 rodadas

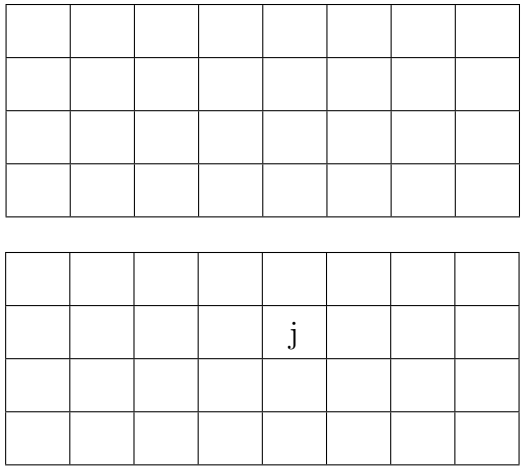
Esta biclique cobre as últimas 4 rodadas da cifra com exceção da adição da chave K^{28}, ou seja, do estado #85 até #96. Basicamente são 4 rodadas menos uma aplicação da transformação linear L.

Primeiro o Adversário escolhe $C_0 = 0$ e calcula $S_0 = f_{K[0,0]}^{-1}(C_0)$, onde $K[0,0]$ é a chave base do grupo a ser testado. Então, as diferenciais-Δ_i são computadas usando as diferenciais de chave Δ_i^K nas subchaves K^{29} e K^{30}. As diferenciais-∇_j são computadas usando as diferenciais de chave ∇_j^K nas subchaves K^{31} e K^{32}. As diferenciais de chave (demonstradas abaixo) foram escolhidas de maneira que as diferenciais-Δ_i e -∇_j sejam independentes.



$\nabla_j^K(31, 32) = K^{31}$

K^{32}



A Figura 6.1 mostra a estrutura básica para construir nossa biclique de dimensão 4 com 2^5 computações de f. Se trata de uma biclique de dimensão 4 porque os estados internos de ambas as diferenciais-Δ_i e -∇_j não compartilham componentes não lineares, o que no caso da cifra *Serpent* são as *S-Boxes*, e tem dimensão quatro porque existem 2^4 possíveis C_i e S_j.

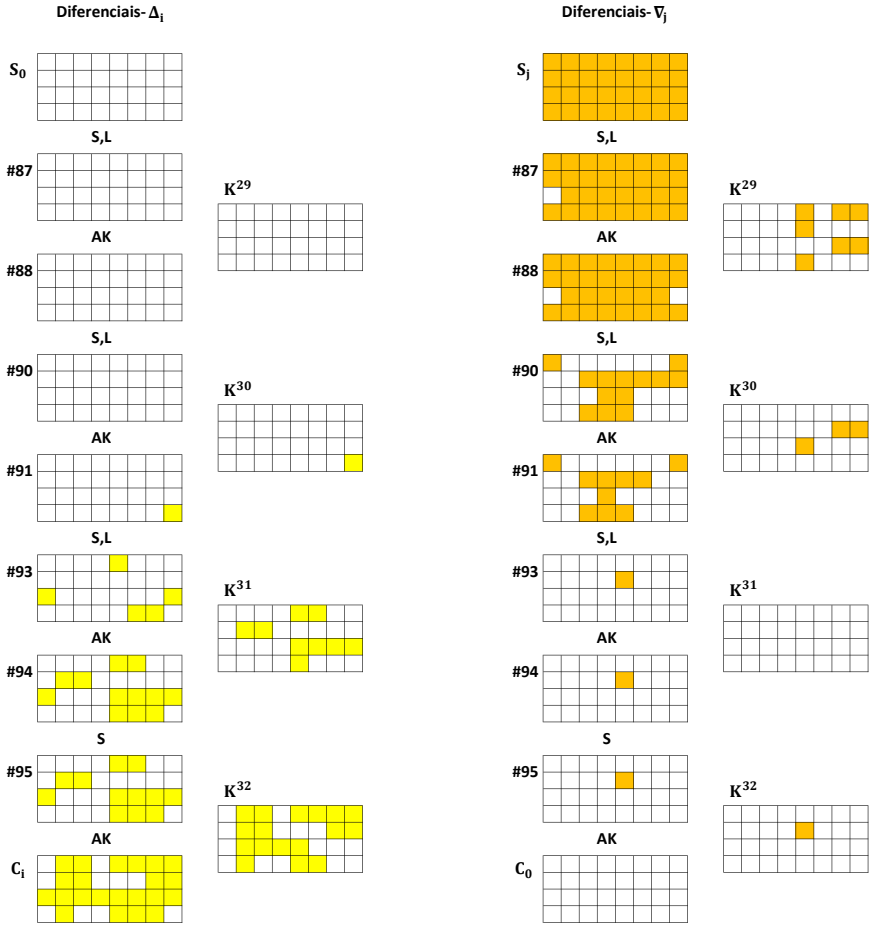


Figura 6.1: Diferenciais-Δ_i e -∇_j na biclique de dimensão 4

Observando C_i é possível notar que apenas 22 *half bytes* são afetados e, portanto, apenas 2^{88} pares de textos claros/cifrados são necessários para o ataque, pois existem apenas 2^{88} possibilidades de textos cifrados.

6.2.3 MP Sobre 28 Rodadas

Nesta parte do ataque, nós finalmente checamos se a chave secreta pertence ao grupo $\{K[i,j]\}$, *i.e.*, se $K_{secreta} \in \{K[i,j]\}$. Fazemos 2^5 pré-computações de v, que definimos como sendo os *half bytes* $\#6_6$ e $\#6_7$, e o armazenamos, junto com todos os estados internos e subchaves envolvidos nessas pré-computações. Então, fazemos

$$P_i \xrightarrow[h]{K[i,j]} v^1_{i,j},$$

$$v^2_{i,j} \xleftarrow[g]{K[0,j]} S_j$$

para todo i e j, recomputando somente as partes que diferem das guardadas em memória. Se $v^1_{i,j} = v^2_{i,j}$, então $K[i,j]$ é um candidato à chave secreta.

Recomputação adiante

Aqui observamos a diferença entre a computação de $P_i \xrightarrow{K[i,j]} v$ e os valores pré-computados $P_i \xrightarrow{K[i,0]} v$, dada pela influência de ∇^K_j nas subchaves K^0 e K^1.

Somente 18 *half bytes* de K^0 são influenciados por ∇^K_j, mas 3 deles não afetam v e portanto podem ser ignorados. Embora K^1 tenha 20 de seus *half bytes* afetados por ∇^K_j, apenas 3 deles afetam v. Figura 6.2(a) mostra os *half bytes* que afetam a computação de v. Portanto, apenas 24 consultas a *S-boxes* são necessárias para os estados internos mais 18 *half bytes* das chaves K^0 e K^1, totalizando 42 *S-boxes*.

Recomputação oposta

De maneira similar à recomputação adiante, olhamos para a diferença entre a computação de $v \xleftarrow{K[i,j]} S_j$ e os pré-computados $v \xleftarrow{K[0,j]} S_j$, dada pela influência de Δ^K_i nas subchaves entre K^2 e K^{28}.

A Figura 6.2(b) mostra que K^{28} afeta 4 *half bytes* de $\#84$, que por sua vez faz com que 15 consultas às *S-boxes* sejam necessárias no estado $\#83$. Após K^{27}, são necessárias 30 consultas no estado $\#80$. De $\#77$ até $\#10$, todas as Fases de Substituição devem ser

recomputadas. Porém, no estado interno #11, apenas 15 *half bytes* afetam v, e assim apenas 15 *S-boxes* precisam ser consultadas para este estado. A mesma quantidade vale para a subchave K^3.

Por fim, apenas 2 consultas a mais são necessárias para #6 e K^2 cada, totalizando 766 *S-boxes* para todos os estados internos e 386 para as subchaves.

6.2.4 Complexidades

Primeiramente temos

$$C_{total} = 2^{k-2d}(C_{biclique} + C_{precomp} + C_{recomp} + C_{falpos}).$$

Sabemos que haverá em média apenas um falso positivo por iteração ($C_{falpos} = 2^{2d}/2^{|v|}$). Também sabemos que $C_{biclique} = 2^5 * (4/32) = 2^2$ e $C_{precomp} = 2^4 * (28/32) = 2^{3,81}$. Resta encontrar C_{recomp}. Para isso, devemos contar cada consulta a uma *S-box* feita na fase de recomputação e comparar com o número total de consultas envolvidas na computação de toda a cifra. Dessa forma, saberemos qual a porcentagem da cifra é computada por iteração, observando somente o número de *S-boxes* consultadas devido ao fato de esta ser a operação mais custosa da cifra.

Uma vez que não é necessário observar os *half bytes* que não influenciam a recomputação (devido ao fato de que o gerador de subchaves somente consulta as *S-boxes* após calcular todas as palavras das subchaves), o total de consultas a *S-boxes* feitas pelo gerador de subchaves é 390, enquanto os estados internos, como mostrados na figura 6.2, fazem 803 consultas, totalizando 1194 *S-boxes*. Como não precisamos recomputar os valores já calculados na fase de pré-computação, e cifra completa consulta 2080 *S-boxes*, então $C_{recomp} = (2^8 - 2^5) * (1194/2080) = 2^{7,01}$.

Ao final, obtemos aproximadamente

$$C_{total} = 2^{248}(2^2 + 2^{3,81} + 2^{7,01} + 1) = 2^{255,21}.$$

computações da *Serpent*-256.

Em termos de memória, o ataque é limitado superiormente por 2^4 computações de g e h, visto que g e h juntos são muito maiores que f e, portanto, é necessário muito mais memória para armazenar todos os estados internos e subchaves deles do que armazenar os 2^5 estados necessários para a biclique.

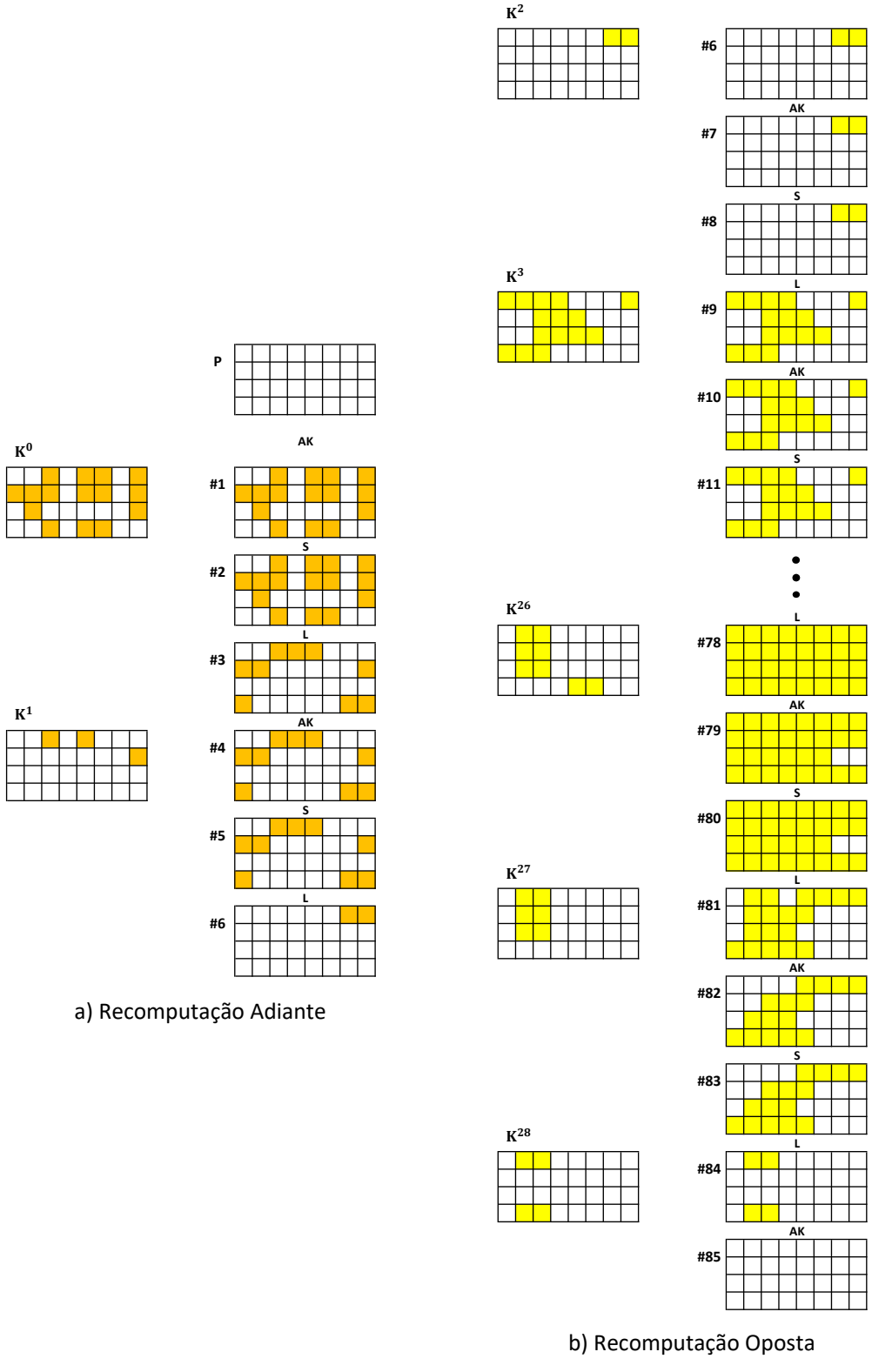


Figura 6.2: (a) Ilustra a recomputação adiante e (b) a recomputação oposta para a biclique de dimensão 4

6.3 O Ataque com Biclique de dimensão 8

6.3.1 Particionamento das Chaves

Como o ataque mostrado anteriormente devemos particionar as chaves primeiramente. Este ataque utiliza uma biclique de dimensão 8 sobre *Serpent*-256, logo as 2^{256} chaves possíveis serão particionadas em 2^{240} grupos de chaves. Portanto, cada chave base define um grupo de 2^{16} chaves.

Neste caso, usamos as subchaves K^{30} e K^{31} para definir esses grupos como mostrado abaixo, já que duas subchaves em sequência são suficientes para gerar todas as outras. Os *half bytes* K^{31}_{11}, K^{31}_{12}, K^{31}_{30} e K^{31}_{31} são setados em zero, enquanto todos os outros *half bytes* de ambos K^{30} e K^{31} irão variar a cada iteração gerando uma nova chave base.

K^{30}

K^{31}

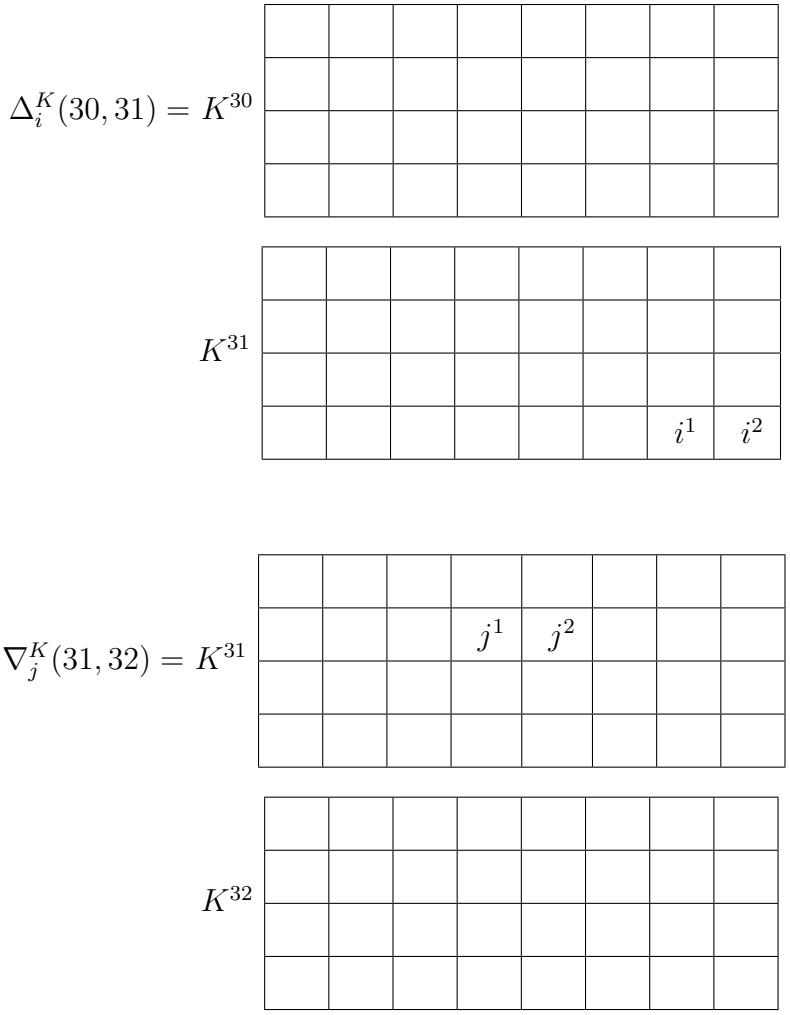
			0	0			
						0	0

As 2^{16} chaves do grupo são testadas graças a este particionamento. Isso acontece pois os *half bytes* K^{31}_{30} e K^{31}_{31} são diretamente dependentes da diferença de chave Δ^K_i utilizada no ataque, e K^{31}_{11} e K^{31}_{12} são diretamente dependentes da diferença de chave ∇^K_j, o que faz com que estes *half bytes* variem por todas as possibilidades. A chave base de um grupo é chamada $K[0,0]$ e uma chave deste grupo é $K[i,j] = K[0,0] \oplus \Delta^K_i \oplus \nabla^K_j$.

6.3.2 Biclique de dimensão 8 Cobrindo 3 rodadas

Esta biclique cobre as últimas 3 rodadas da cifra com exceção da adição da chave K^{29}, ou seja, do estado #88 até #96. Basicamente são 3 rodadas menos uma aplicação da transformação linear L.

Primeiro o Adversário escolhe $C_0 = 0$ e calcula $S_0 = f^{-1}_{K[0,0]}(C_0)$, onde $K[0,0]$ é a chave base do grupo a ser testado. Então, as diferenciais-Δ_i são computadas usando as diferenciais de chave Δ^K_i nas subchaves K^{30} e K^{31}. As diferenciais-∇_j são computadas usando as diferenciais de chave ∇^K_j nas subchaves K^{31} e K^{32}. As diferenciais de chave foram escolhidas de maneira que as diferenciais-Δ_i e -∇_j sejam independentes. Δ^K_i e ∇^K_j estão demonstradas abaixo, onde i_1 se refere aos 4 bits mais à esquerda de i e i_2 aos bits restantes. O mesmo é válido para j_1 e j_2.



A Figura 6.3 mostra a estrutura básica para construir nossa biclique de dimensão 8 com 2^9 computações de f. Se trata de uma biclique de dimensão 8 porque os estados internos de ambas as diferenciais-Δ_i e -∇_j não compartilham componentes não lineares, e tem dimensão oito porque existem 2^8 possíveis C_i e S_j.

Observando C_i, é possível notar que 15 *half bytes* são afetados e, portanto, apenas 2^{60} pares de textos claros/cifrados são necessários para o ataque, pois existem apenas 2^{60} possibilidades de textos cifrados.

6.3.3 MP Sobre 29 Rodadas

Finalmente, checamos se a chave secreta pertence ao grupo $\{K[i,j]\}$, *i.e.* se $K_{secreta} \in \{K[i,j]\}$. Fazemos 2^9 pré-computações de v, que definimos como sendo os *half bytes* $\#6_6$ e $\#6_7$ (como no ataque anterior), e os armazenamos, junto com todos os estados internos e subchaves envolvidos nessas pré-computações. Então, fazemos

$$P_i \xrightarrow[h]{K[i,j]} v_{i,j}^1 \quad \text{e}$$

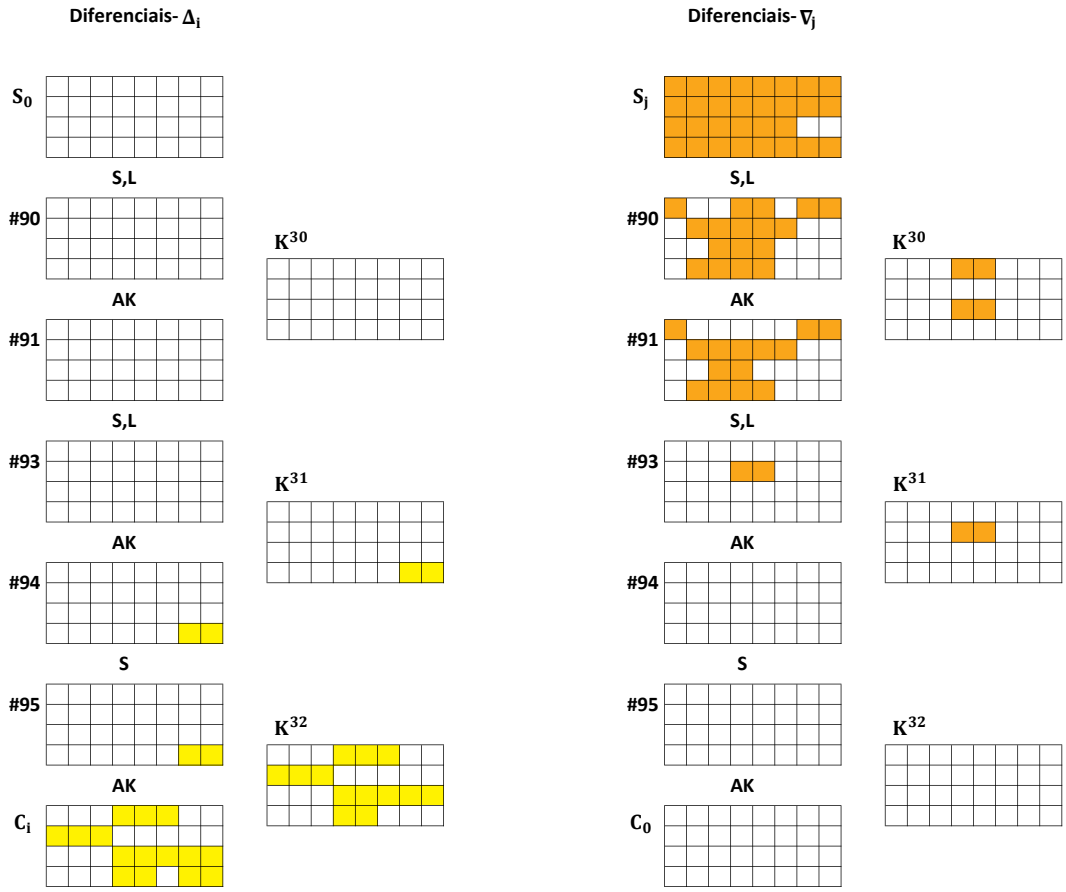


Figura 6.3: Diferenciais-Δ_i e -∇_j na biclique de dimensão 8.

$$v_{i,j}^2 \xleftarrow[g]{K[0,j]} S_j$$

para todo i e j, recomputando somente as partes que diferem das guardadas em memória. Se $v_{i,j}^1 = v_{i,j}^2$, então $K[i,j]$ é um candidato à chave secreta.

Recomputação adiante

Aqui observamos a diferença entre a computação de $P_i \xrightarrow{K[i,j]} v$ e os valores pré-computados $P_i \xrightarrow{K[i,0]} v$, dada pela influência de ∇_j^K nas subchaves K^0 e K^1.

Temos que 22 *half bytes* de K^0 e K^1 são influenciados por ∇_j^K, porém 4 *half bytes* de K^0 e 15 *half bytes* de K^1 não afetam v e portanto podem ser ignorados. A Figura 6.4(a) mostra os *half bytes* que afetam a computação de v. Portanto, temos um total de 25 consultas a *S-boxes* para as chaves, e para os estados internos temos mais 27, totalizando 52 consultas a *S-boxes*.

Recomputação oposta

De maneira similar à recomputação adiante, olhamos para a diferença entre a computação de $v \xleftarrow{K[i,j]} S_j$ e os pré computados $v \xleftarrow{K[0,j]} S_j$, dada pela influência de Δ_i^K nas subchaves entre K^2 e K^{29}.

A Figura 6.4(b) mostra que K^{29} afeta 6 *half bytes* de #87, que por sua vez faz com que 15 consultas às *S-boxes* sejam necessárias após a transformação L no estado #86. Todas as Fases de Substituição devem ser recomputadas a partir dai, do estado #83 ao estado #10. Porém, no estado interno #11, apenas 15 *half bytes* afetam v, e assim apenas 15 *S-boxes* precisam ser consultadas para este estado. A mesma quantidade vale para a subchave K^3.

Por fim, apenas 2 consultas a mais são necessárias para #8 e K^2 cada, totalizando 802 *S-boxes* para todos os estados internos e 558 para as subchaves.

6.3.4 Complexidades

Como anteriormente, temos

$$C_{total} = 2^{k-2d}(C_{biclique} + C_{precomp} + C_{recomp} + C_{falpos}).$$

Em média, teremos $C_{falpos} = 2^{16}/2^8 = 2^8$ falso positivos por iteração e cada um deles deverá ser testado. Também sabemos que $C_{biclique} = 2^9 * (3/32) = 2^{5,58}$ e $C_{precomp} = 2^8 * (29/32) = 2^{7,86}$. Resta encontrar C_{recomp} que, como anteriormente, contamos cada consulta a uma *S-box* feita na fase de recomputação e comparar com o número total de consultas envolvidas na computação de toda a cifra. Dessa forma, saberemos qual a porcentagem da cifra é computada por iteração, observando somente o número de *S-boxes* consultadas devido ao fato de esta ser a operação mais custosa da cifra.

O total de consultas a *S-boxes* feitas pelo gerador de subchaves é 583, enquanto os estados internos, como mostrados na figura 6.4, fazem 829 consultas, totalizando 1412 *S-boxes* consultadas. Como não precisamos recomputar os valores já calculados na fase de pré-computação, e a cifra completa consulta 2080 *S-boxes*, então $C_{recomp} = (2^{16} - 2^9) * (1412/2080) = 2^{15,43}$.

Ao final, obtemos aproximadamente

$$C_{total} = 2^{240}(2^{5,58} + 2^{7,86} + 2^{15,43} + 2^8) = 2^{255,45}.$$

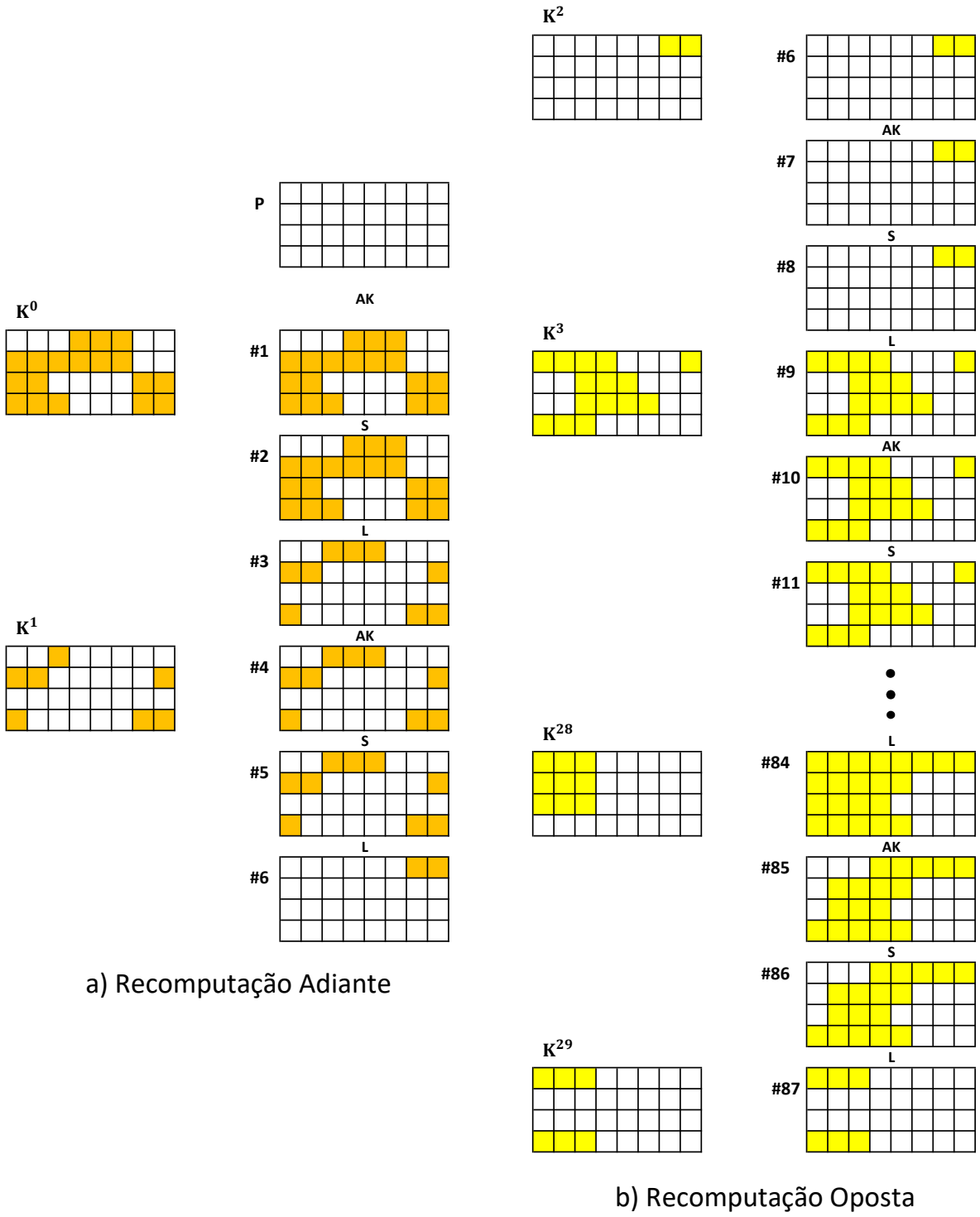


Figura 6.4: (a) Ilustra a recomputação adiante e (b) a recomputação oposta para a biclique de dimensão 8.

computações da cifra *Serpent*-256.

Em termos de memória, o ataque é limitado superiormente por 2^8 computações de g e h, visto que g e h juntos são muito maiores que f e, portanto, é necessário muito mais memória para armazenar todos os estados internos e subchaves deles do que armazenar os 2^9 estados necessários para a biclique.

6.4 Análise dos Ataques à cifra *Serpent*

A Criptanálise Biclique é uma técnica muito poderosa para cifras de bloco e a cifra *Serpent* não é uma exceção. Os ataques aqui apresentados têm complexidade de tempo aproximada de $2^{255,21}$ e $2^{255,44}$ para o *Serpent*-256 para as bicliques de dimensão 4 e 8 respectivamente. Para a biclique menor, a complexidade de memória é menor que 2^4 computações completas do *Serpent*-256 (isso é, todos os 96 estados internos e 33 subchaves) e usa 2^{88} textos cifrados escolhidos a seus correspondentes textos claros. Para a segunda biclique temos que a complexidade de memória é menor que 2^8 computações completas do *Serpent*-256 e usa 2^{60} textos cifrados escolhidos a seus correspondentes textos claros. Logo, vemos que os dois ataques têm suas vantagens, sendo a biclique menor a que produziu o ataque com menor complexidade de tempo, porém a maior biclique utiliza uma quantidade muito menor de textos cifrados escolhidos para realizar o ataque.

Isso mostra que cifras de bloco do tipo SPN com difusão baixa no gerador de chaves normalmente são vulneráveis à Criptanálise Biclique, já que a cifra Serpent que, por vinte anos, nunca sofreu um ataque que fosse eficiente nem para uma versão com metade das rodadas da versão completa, ainda que marginalmente, foi vencida pela Criptanálise Biclique.

Quanto às versões *Serpent*-128 e *Serpent*-192, a viabilidade do ataque ainda não foi confirmada devido ao fato de serem necessárias duas subchaves consecutivas para gerar todas as 33 subchaves necessárias para a cifra, forçando a Criptanálise Biclique a custar próximo de $2^{255,5}$ computações, muito maior do que a busca exaustiva para essas versões. Apesar disso, a criação de bicliques nas primeiras rodadas é possível já que podemos utilizar a própria chave secreta como chave base. A diferença será somente na quantidade de rodadas que serão alcançadas, o que deve ser estudado.

A possibilidade de existirem bicliques melhores que as aqui apresentadas também é real, já que não podemos provar que estas bicliques sejam ótimas tanto quanto à quantidade de rodadas cobertas quanto à dimensão das bicliques.

Capítulo 7

Conclusões

Este trabalho teve como objetivo estudar e aplicar a Criptanálise Biclique em cifras de bloco consideradas seguras. As cifras candidatas para estudo foram dois dos finalistas do concurso *Advanced Encryption Standard*, as cifras *Rijndael* e *Serpent* (os outros finalistas são *Twofish*, RC6 e *MARS*.).

Para o *Rijndael*, fizemos uma implementação do ataque original feito por Bogdanov *et al.* [8]. Este ataque é meramente teórico e, portanto, não leva em consideração os pequenos *overheads* associados à implementação na prática. Esta implementação teve por objetivo testar, de maneira tão prática quanto computacionalmente viável, se as complexidades descritas no artigo original são válidas na prática, ou seja, testar se o *overhead* adicionado pelo ataque tem algum efeito notável na complexidade de tempo. Como mostrado na Tabela 5.4, e na Seção 5.4, o *overhead* é significativo na aplicação da MP, com aumento de cerca de 56% na complexidade, porém o ataque continua mais de duas vezes mais rápido que uma busca exaustiva.

No caso da cifra *Serpent*, fizemos dois ataques completos à sua versão de chave com 256 bits usando o método básico que atacou o *Rijndael*, pois ambas as cifras são semelhantes em vários sentidos. Com isso, foi possível criar os primeiros ataques com complexidade de tempo menor que uma busca exaustiva por todas as chaves, apesar de, devido à grande quantidade de textos cifrados escolhidos necessários (2^{88} e 2^{60}), estes ataques serem menos práticos que a força bruta. Estes ataques utilizam uma biclique de dimensão 4 cobrindo as últimas 4 rodadas da cifra e uma biclique de dimensão 8 cobrindo as últimas 3 rodadas da cifra e têm complexidade de tempo de $2^{255,21}$ e $2^{255,45}$ respectivamente.

Trabalhos futuros envolvem a aplicação da Criptanálise Biclique e variações nas outras cifras do concurso AES, melhoria do ataque atual ao *Serpent* (usando bicliques melhores

ou variações mais recentes da Criptanálise Biclique) e o aprimoramento da implementação usada no *Rijndael*, generalizando-o para permitir sua aplicação à outras cifras e paralelização em CPU e GPU para oferecer maior confiabilidade nos resultados.

Referências

[1] Abed, F.; Forler, C.; List, E.; Lucks, S.; Wenzel, J. Biclique cryptanalysis of the PRESENT and LED lightweight ciphers. *IACR Cryptology ePrint Archive 2012* (2012), 591.

[2] Biham, E. New types of cryptanalytic attacks using related keys. *Journal of Cryptology 7*, 4 (1994), 229–246.

[3] Biham, E.; Anderson, R.; Knudsen, L. Serpent: A new block cipher proposal. In *International Workshop on Fast Software Encryption* (1998), Springer, pp. 222–238.

[4] Biham, E.; Biryukov, A.; Shamir, A. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In *International Conference on the Theory and Applications of Cryptographic Techniques* (1999), Springer, pp. 12–23.

[5] Biham, E.; Shamir, A. Differential cryptanalysis of DES-like cryptosystems. *Journal of CRYPTOLOGY 4*, 1 (1991), 3–72.

[6] Biham, E.; Shamir, A. Differential cryptanalysis of the full 16-round DES. In *Annual International Cryptology Conference* (1992), Springer, pp. 487–496.

[7] Biryukov, A.; Khovratovich, D. Related-key cryptanalysis of the full AES-192 and AES-256. In *International Conference on the Theory and Application of Cryptology and Information Security* (2009), Springer, pp. 1–18.

[8] Bogdanov, A.; Khovratovich, D.; Rechberger, C. Biclique cryptanalysis of the full AES. In *International Conference on the Theory and Application of Cryptology and Information Security* (2011), Springer, pp. 344–371.

[9] Bogdanov, A.; Rechberger, C. A 3-subset meet-in-the-middle attack: cryptanalysis of the lightweight block cipher KTANTAN. In *International Workshop on Selected Areas in Cryptography* (2010), Springer, pp. 229–240.

[10] Brown, L.; Pieprzyk, J.; Seberry, J. LOKI—a cryptographic primitive for authentication and secrecy applications. In *International Conference on Cryptology* (1990), Springer, pp. 229–236.

[11] Chen, S.-z.; Xu, T.-m. Biclique key recovery for ARIA-256. *IET Information Security 8*, 5 (2014), 259–264.

[12] Çoban, M.; Karakoç, F.; Boztaş, Ö. Biclique cryptanalysis of TWINE. In *International Conference on Cryptology and Network Security* (2012), Springer, pp. 43–55.

[13] Daemen, J.; Knudsen, L.; Rijmen, V. The block cipher Square. In *International Workshop on Fast Software Encryption* (1997), Springer, pp. 149–165.

[14] Daemen, J.; Rijmen, V. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[15] Demirci, H.; Selçuk, A. A. A meet-in-the-middle attack on 8-round AES. In *International Workshop on Fast Software Encryption* (2008), Springer, pp. 116–126.

[16] Demirci, H.; Selšuk, A. A.; TŘre, E. A new meet-in-the-middle attack on the IDEA block cipher. In *International Workshop on Selected Areas in Cryptography* (2003), Springer, pp. 117–129.

[17] Demirci, H.; Taşkin, İ.; Çoban, M.; Baysal, A. Improved meet-in-the-middle attacks on AES. In *International Conference on Cryptology in India* (2009), Springer, pp. 144–156.

[18] Diffie, W.; Hellman, M. E. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *Computer 10*, 6 (1977), 74–84.

[19] Dunkelman, O.; Sekar, G.; Preneel, B. Improved meet-in-the-middle attacks on reduced-round DES. In *International Conference on Cryptology in India* (2007), Springer, pp. 86–100.

[20] Feistel, H. Cryptography and computer privacy. *Scientific american 228*, 5 (1973), 15–23.

[21] FIPS. Data Encryption Standard (DES). *PUB 46-3* (1999).

[22] FIPS. Specification for the ADVANCED ENCRYPTION STANDARD (AES). *PUB 197* (2001).

[23] Fluhrer, S.; Mantin, I.; Shamir, A. Weaknesses in the key scheduling algorithm of RC4. In *International Workshop on Selected Areas in Cryptography* (2001), Springer, pp. 1–24.

[24] Ghosh, M.; Sanadhya, S. K.; Chang, D. *Analysis of block cipher constructions against biclique and multiset attacks*. Tese de Doutorado, Indraprastha Institute of Information Technology, Delhi (IIIT-Delhi), 2016.

[25] Hong, D.; Koo, B.; Kwon, D. Biclique attack on the full HIGHT. In *International Conference on Information Security and Cryptology* (2011), Springer, pp. 365–374.

[26] Jeong, K.; Kang, H.; Lee, C.; Sung, J.; Hong, S. Biclique Cryptanalysis of Lightweight Block Ciphers PRESENT, Piccolo and LED. *IACR Cryptology ePrint Archive 2012* (2012), 621.

[27] Katz, J.; Menezes, A. J.; Van Oorschot, P. C.; Vanstone, S. A. *Handbook of applied cryptography*. CRC press, 1996.

[28] Kelsey, J.; Schneier, B.; Wagner, D. Key-schedule cryptanalysis of IDEA, G-DES, GOST, Safer, and triple-DES. In *Annual International Cryptology Conference* (1996), Springer, pp. 237–251.

[29] Khovratovich, D.; Leurent, G.; Rechberger, C. Narrow-Bicliques: cryptanalysis of full IDEA. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2012), Springer, pp. 392–410.

[30] Khovratovich, D.; Rechberger, C.; Savelieva, A. Bicliques for preimages: attacks on Skein-512 and the SHA-2 family. In *Fast Software Encryption* (2012), Springer, pp. 244–263.

[31] Knudsen, L. R. Truncated and higher order differentials. In *International Workshop on Fast Software Encryption* (1994), Springer, pp. 196–211.

[32] Ko, Y.; Hong, S.; Lee, W.; Lee, S.; Kang, J.-S. Related key differential attacks on 27 rounds of XTEA and full-round GOST. In *International Workshop on Fast Software Encryption* (2004), Springer, pp. 299–316.

[33] Langford, S. K.; Hellman, M. E. Differential-linear cryptanalysis. In *Annual International Cryptology Conference* (1994), Springer, pp. 17–25.

[34] Mala, H. Biclique-based cryptanalysis of the block cipher SQUARE. *IET Information Security 8*, 3 (2014), 207–212.

[35] Matsui, M. The first experimental cryptanalysis of the Data Encryption Standard. In *Annual International Cryptology Conference* (1994), Springer, pp. 1–11.

[36] Nechvatal, J.; Barker, E.; Bassham, L.; Burr, W.; Dworkin, M.; Foti, J.; Roback, E. Report on the development of the Advanced Encryption Standard (AES). *Journal of Research of the National Institute of Standards and Technology 106*, 3 (2001), 511.

[37] Standaert, F.-X.; Piret, G.; Quisquater, J.-J. Cryptanalysis of block ciphers: A survey. *UCL Crypto Group* (2003).

[38] Stinson, D. R. *Cryptography: theory and practice*. CRC press, 2005.

[39] Tao, B.; Wu, H. Improving the biclique cryptanalysis of AES. In *Australasian Conference on Information Security and Privacy* (2015), Springer, pp. 39–56.

[40] Wagner, D. The boomerang attack. In *International Workshop on Fast Software Encryption* (1999), Springer, pp. 156–170.

APÊNDICE A – Tabelas da cifra AES

63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tabela A.1: *S-box* utilizada pelo AES

i	1	2	3	4	5	6	7	8	9	10
	01	02	04	08	10	20	40	80	1B	36
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00

Tabela A.2: Constantes RCON utilizadas no gerador de subchaves do AES

APÊNDICE B – Biclique e Recomputação do ataque original do AES

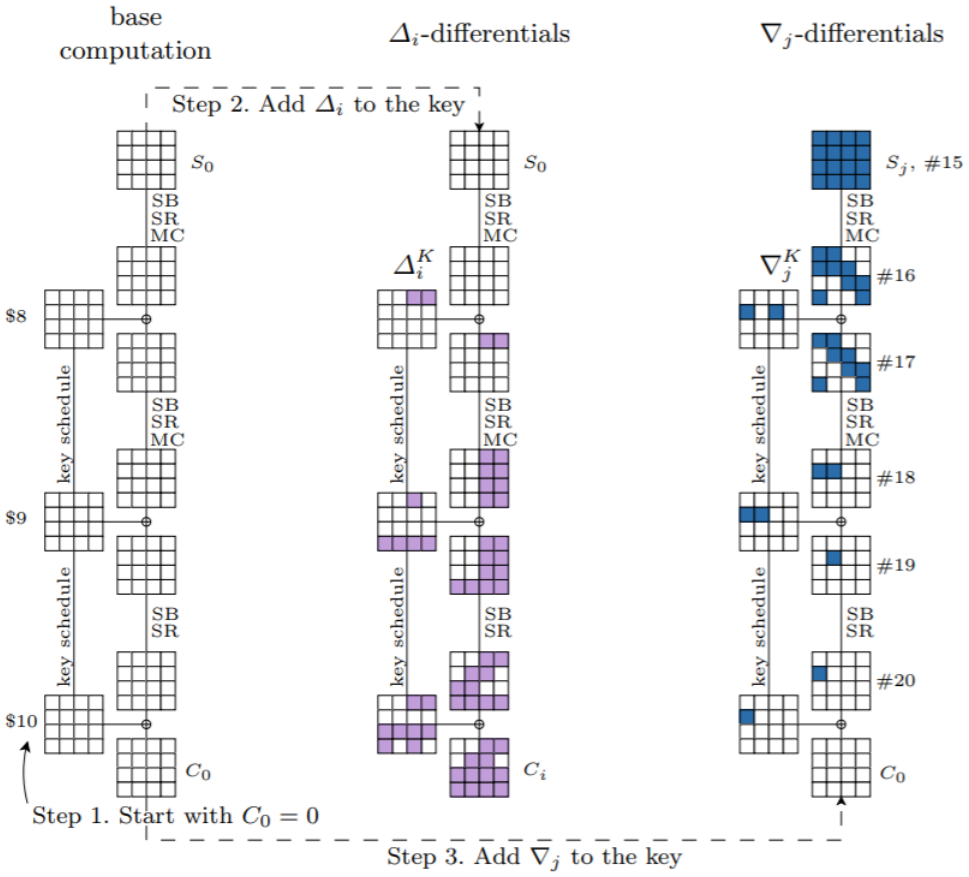


Figura B.1: Biclique do ataque original ao AES [8]

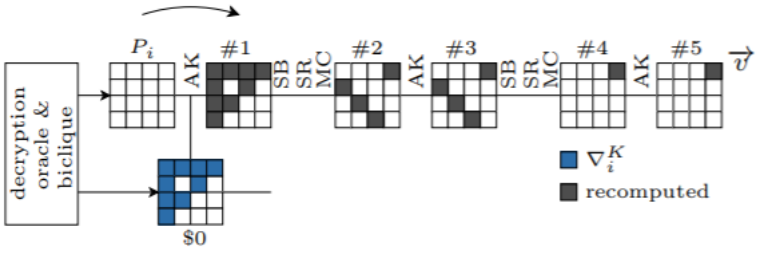


Figura B.2: Recomputação adiante do ataque original ao AES [8]

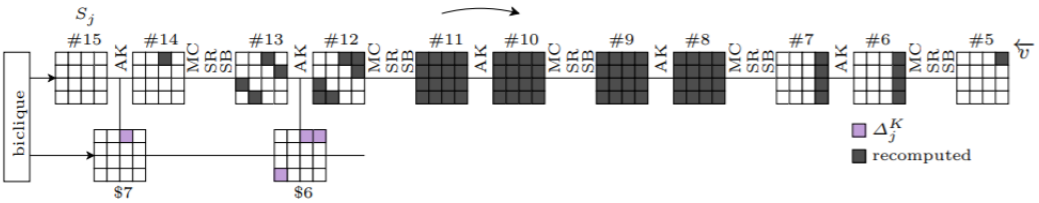


Figura B.3: Recomputação oposta do ataque original ao AES [8]

APÊNDICE C – S-boxes da cifra Serpent

B_0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
B_1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
B_2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
B_3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
B_4	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
B_5	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
B_6	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
B_7	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
B_8	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
B_9	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
B_{10}	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
B_{11}	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
B_{12}	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
B_{13}	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
B_{14}	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
B_{15}	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
B_{16}	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
B_{17}	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
B_{18}	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
B_{19}	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
B_{20}	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
B_{21}	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
B_{22}	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
B_{23}	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
B_{24}	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
B_{25}	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
B_{26}	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
B_{27}	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
B_{28}	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
B_{29}	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
B_{30}	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
B_{31}	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Tabela C.1: *S-boxes* utilizadas pela cifra *Serpent*