MARCOS ANTONIO SIMPLICIO JUNIOR

**ALGORITMOS CRIPTOGRÁFICOS PARA REDES
DE SENSORES**

Dissertação apresentada à Escola Politécnica
da Universidade de São Paulo para obtenção
do Título de Mestre em Engenharia Elétrica.

São Paulo
2008 - (Edição Original)
2008 - (Edição Revisada)

MARCOS ANTONIO SIMPLICIO JUNIOR

**ALGORITMOS CRIPTOGRÁFICOS PARA REDES
DE SENSORES**

Dissertação apresentada à Escola Politécnica
da Universidade de São Paulo para obtenção
do Título de Mestre em Engenharia Elétrica.

Área de Concentração:

Sistemas Digitais

Orientador:

Paulo Sérgio Licciardi M. Barreto

São Paulo
2008 - (Edição Original)
2008 - (Edição Revisada)

FICHA CATALOGRÁFICA

Simplicio Junior, Marcos Antonio

Algoritmos criptográficos para redes de sensores. São Paulo, 2008 - (Edição Original)

2008 - (Edição Revisada) .

184 p.

Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

1. Segurança 2. Criptografia. 3. Redes de Sensores. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

AGRADECIMENTOS

Primeiramente, às duas mulheres mais importantes na minha vida: minha mãe, Maria Suely M. Simplício, e minha noiva e futura esposa, Débora Arantes (Simplício). A primeira, agradeço por superar, com garra e determinação, a tarefa de criar dois filhos após o falecimento prematuro de meu pai. A segunda, agradeço por me aturar nas melhores e piores horas e por continuar a fazê-lo dia após dia com muito carinho. Agradeço imensamente a ambas pela força e paciência em todos os momentos.

Além disto, não poderia deixar de agradecer a todos que me apoiaram e me ajudaram a realizar o sonho de chegar à faculdade e obter o grau de mestre. Dentre as diversas pessoas que fazem parte de uma grande lista, gostaria de agradecer em especial à professora e amiga Elisabeth M. do Lago Ferreira, quem me fez acreditar que isto seria possível, e ao meu orientador e amigo Paulo S. L. M. Barreto, quem ajudou a a tornar isto possível.

Ao mesmo tempo, agradeço profundamente ao FDTE (Fundação Para o Desenvolvimento Tecnológico da Engenharia) e a todo o pessoal do LARC, não apenas pelo auxílio financeiro mas também pelas inspiradas e bem humoradas discussões sobre os projetos do laboratório.

A estas e outras tantas pessoas importantes ao longo destes 18 anos de vida, meus sinceros agradecimentos.

RESUMO

É crescente a necessidade de prover segurança às informações trocadas nos mais diversos tipos de redes. No entanto, redes amplamente dependentes de dispositivos com recursos limitados (como sensores, *tokens* e *smart cards*) apresentam um desafio importante: a reduzida disponibilidade de memória, capacidade de processamento e (principalmente) energia dos mesmos dificulta a utilização de alguns dos principais algoritmos criptográficos considerados seguros atualmente. É neste contexto que se insere o presente documento, que não apenas apresenta uma pesquisa envolvendo projeto e análise de algoritmos criptográficos, mas também descreve um novo algoritmo simétrico denominado CURUPIRA. Esta cifra de bloco baseia-se na metodologia conhecida como Estratégia de Trilha Larga e foi projetada especialmente para ambientes onde existe escassez de recursos.

O CURUPIRA possui estrutura involutiva, o que significa que os processos de encriptação e decríptação diferem apenas na seqüência da geração de chaves, dispensando a necessidade de algoritmos distintos para cada uma destas operações. Além disto, são propostas duas formas diferentes para seu algoritmo de geração de chaves, cada qual mais focada em segurança ou em desempenho. Entretanto, ambas as formas caracterizam-se pela possibilidade de computação das sub-chaves de *round* no momento de sua utilização, em qualquer ordem, garantindo uma operação com reduzido uso de memória RAM.

Palavras-chave: Criptografia Simétrica. Cifras de Bloco. Cifras Involutivas. Estratégia de Trilha Larga. Redes de Sensores. Redes com recursos limitados.

ABSTRACT

The need for security is a great concern in any modern network. However, networks that are highly dependent of constrained devices (such as sensors, tokens and smart cards) impose a difficult challenge: their reduced availability of memory, processing power and (more importantly) energy hinders the deployment of many important cryptographic algorithms known to be secure. In this context, this document not only presents the research involving the design and analysis of cryptographic algorithms, but also proposes a new symmetric block cipher named CURUPIRA.

The CURUPIRA follows the methodology known as the Wide Trail Strategy and was specially developed having constrained platforms in mind. It displays an involutinal structure, which means that the encryption and decryption processes differ only in the key schedule and, thus, there is no need to implement them separately. Also, two distinct scheduling algorithms are proposed, whose main focus are either on tight security or improved performance. In spite of this difference, both of them allow the keys to be computed on-the-fly, in any desired order, assuring a reduced consumption of RAM memory during their operation.

Keywords: Symmetric Cryptography. Block Ciphers. Involutinal Ciphers. Wide Trail Strategy. Sensor Networks. Constrained Networks.

RÉSUMÉ

Le besoin d'assurer la sécurité des données est un sujet de grande importance dans les réseaux modernes. Cependant, réseaux fortement dépendent de dispositifs de capacité réduite (comme des senseurs, badges et cartes à puce) impose un grand défi : leur limitation en termes de disponibilité de mémoire, vitesse du traitement des données et (surtout) énergie entrave l'utilisation de plusieurs algorithmes de chiffrement modernes considérés solides. Dans ce contexte, ce document comprend une étude concernant le projet et l'analyse de primitives cryptographiques et propose aussi un nouvel algorithme de chiffrement par bloc nommé CURUPIRA.

Le CURUPIRA suit la méthodologie de projet connue sous le nom de "Wide Trail Strategy" et a été spécialement développé pour des plateformes ayant ressources limitées. Il possède une structure involutive, de façon que les processus de chiffrement et déchiffrement ne diffèrent que par l'ordre d'application des sous-clés et, donc, il n'y a pas besoin de les implémenter séparément. En plus, deux algorithmes pour la génération des sous-clés sont proposés, un plus concerné avec la sécurité et l'autre plus centré sur la vitesse. Toutefois, ces deux algorithmes permettent que les sous-clés soient générées au moment d'utilisation, dans un ordre arbitraire, en épargnant de la mémoire RAM.

Mots-clés: Cryptographie Symétrique. Chiffrement par blocs. Comportement Involutif. Wide Trail Strategy. Réseaux de Senseurs. Réseaux de ressources limitées.

SUMÁRIO

Lista de Figuras

Lista de Tabelas

1	Introdução	17
1.1	Justificativa	18
1.2	Objetivos	19
1.3	Metodologia	19
1.4	Contribuições originais	20
1.5	Organização	20
2	Conceitos de Criptografia	23
2.1	Segurança na Comunicação	23
2.2	Breve história da criptografia	24
2.3	Dois métodos para trocar informações	27
2.3.1	Criptografia Simétrica	27
2.3.2	Criptografia Assimétrica	28
2.3.3	Simetria × Assimetria	29
2.4	Resumo	31
3	Tipos de Cifras simétricas	33

3.1	Cifras de Fluxo	33
3.1.1	Cifras de Fluxo Síncronas	34
3.1.2	Cifras de Fluxo Auto-síncronas	35
3.1.3	Utilização e Segurança	36
3.2	Cifras de Bloco	38
3.2.1	Modos de Operação	39
3.2.1.1	ECB (Electronic Code Book)	40
3.2.1.2	CBC (Cipher Block Chaining)	41
3.2.1.3	CFB (Cipher Feedback)	42
3.2.1.4	OFB (Output Feedback)	43
3.2.1.5	CTR (Counter Mode)	44
3.2.2	Ciphertext Stealing	45
3.2.3	Utilização e Segurança	47
3.3	Resumo	48
4	Cifras de Bloco e Criptoanálise	49
4.1	Redes de Substituição-Permutação	49
4.2	Cifras de Feistel	52
4.3	Criptoanálise Moderna	54
4.3.1	Ataque de Força Bruta	57
4.3.2	Criptoanálise Linear	57
4.3.2.1	O lema de <i>Piling-Up</i>	60

4.3.2.2	Complexidade do Ataque	62
4.3.3	Criptanálise Diferencial	63
4.3.3.1	Probabilidade de uma característica diferencial	65
4.3.3.2	Complexidade do ataque	65
4.3.3.3	Outras modalidades de Ataque Diferencial	66
4.3.4	Ataque de Interpolação	67
4.3.5	Ataque de Saturação	68
4.3.6	Ataque Gilbert-Minier	70
4.3.7	Ataque de Extensão Geral	70
4.3.8	Ataques Algébricos	71
4.3.9	Outros Ataques	72
4.4	Resumo	73
5	Estratégia de Trilha Larga	75
5.1	Introdução	75
5.2	Segurança contra Ataques Lineares	78
5.3	Segurança contra Ataques Diferenciais	80
5.4	Peso de uma trilha	81
5.4.1	O conceito de “Trilhas Largas”	82
5.5	Fator de ramificação	84
5.5.1	Propriedades	85
5.5.2	Difusão associada a 2 rounds de uma cifra	85

5.6	Combinação de segurança e eficiência	86
5.6.1	A transformação de difusão θ	86
5.6.2	A transformação linear Θ	87
5.6.3	Construção eficiente de Θ	87
5.6.4	Construção eficiente da função de round	88
5.7	Códigos MDS	89
5.8	Resumo	91
6	O Protocolo TinySec para redes de sensores	93
6.1	Projeto do Protocolo	94
6.2	Modos de operação do protocolo e Estrutura de pacotes	95
6.3	Criptografia	96
6.3.1	Formato do campo IV	97
6.3.2	Algoritmo criptográfico	97
6.4	Críticas	98
7	O CURUPIRA	101
7.1	Características Gerais	101
7.2	Preliminares Matemáticas e Notação	103
7.2.1	Corpos Finitos	103
7.2.1.1	Adição	104
7.2.1.2	Multiplicação	104
7.2.2	Matrizes em $GF(2^m)$	106

7.2.3	Propriedades Criptográficas	107
7.2.4	Outras notações utilizadas	109
7.3	Estrutura do CURUPIRA	109
7.3.1	Elementos da Encriptação e Decriptação	109
7.3.1.1	A camada não-linear γ	109
7.3.1.2	A camada de permutação π	111
7.3.1.3	A camada de difusão linear θ	111
7.3.1.4	A adição de chave $\sigma[k]$	112
7.3.2	Algoritmo de evolução das chaves - versão 1	112
7.3.2.1	Representação das chaves	112
7.3.2.2	Constantes de escalonamento	112
7.3.2.3	O escalonamento de chaves ψ_s	113
7.3.2.4	A seleção de chaves ϕ_r	115
7.3.3	Algoritmo de evolução das chaves - versão 2	116
7.3.3.1	Representação das chaves	116
7.3.3.2	Constantes de escalonamento	117
7.3.3.3	O escalonamento de chaves Υ_s	117
7.3.3.4	A seleção de chaves ϕ_r^*	121
7.3.4	A cifra completa	121
7.3.5	A cifra inversa	122
7.4	Análise de Segurança	123
7.4.1	Criptanálise Diferencial e Linear	124

7.4.2	Criptoanálise Diferencial Truncada	124
7.4.3	Ataques por interpolação	125
7.4.4	Chaves fracas e Chaves relacionadas	125
7.4.5	Ataque de Saturação	126
7.4.6	O ataque Gilbert-Minier	127
7.4.7	Ataque de extensão geral	127
7.4.8	Ataques algébricos e outros ataques	128
7.4.9	Conclusões sobre segurança	128
7.5	Implementação e desempenho	129
7.5.1	Considerações preliminares	132
7.6	Resumo	133
8	Testes e comparação com outras cifras	135
8.1	Características das implementações	136
8.2	Resultados obtidos: 8 bits	140
8.3	Resultados obtidos: Simulador de Sensor	146
8.4	Resultados obtidos: 32 bits	148
8.5	Considerações finais sobre desempenho	152
9	Conclusões	155
9.1	Trabalhos futuros	157
	Referências	161

Apêndice A - Projeto das matrizes D e E	169
A.1 Matriz D : Comportamento Involutivo	169
A.2 Matriz E : Escalonamento Cíclico	172
Apêndice B - Ataque de Saturação aplicado ao CURUPIRA	175
B.1 Comportamento para 3 rounds	175
B.2 O ataque básico de 4 rounds	177
B.3 Adicionando um round no final	178
B.4 Adicionando um round no início	179
B.5 Soma-parcial aplicada a 6 rounds	180
B.6 Soma-parcial aplicada a 7 rounds	182
Apêndice C - Detalhes adicionais da S-Box	183
C.1 Formas de implementação	183
C.2 Segurança	183

LISTA DE FIGURAS

1	Criptografia Simétrica	28
2	Criptografia Assimétrica	29
3	Operação de uma Cifra de Fluxo Síncrona	36
4	Operação de uma Cifra de Fluxo Auto-síncrona	37
5	Encriptação e Decriptação usando o Modo de Operação ECB	40
6	Encriptação e Decriptação usando o Modo de Operação CBC	41
7	Encriptação e Decriptação usando o Modo de Operação CFB	43
8	Encriptação e Decriptação usando o Modo de Operação OFB	44
9	Encriptação e Decriptação usando o Modo de Operação CTR	45
10	Ciphertext Stealing para os modos ECB e CBC	47
11	Dois modos de incorporar as chaves de round em uma SPN	51
12	Estrutura de Feistel	53
13	Cifra simplificada: uma S-Box e 2 camadas de aplicação de chave	59
14	Exemplo de Trilha Diferencial	76
15	SPN no contexto da Estratégia de Trilha Larga	78
16	Exemplo de Trilha “Larga”	83
17	Esquema ilustrativo de códigos MDS, evidenciando a matriz geratriz	90
18	Formato dos pacotes TinySec e TinyOS	95
19	Desempenho x Número de Rounds para o CURUPIRA _c -1 (PIC)	143

20	Desempenho - CURUPIRA Completo (PIC)	145
21	Desempenho - CURUPIRA ₉₆ (PIC)	146
22	Desempenho - CURUPIRA Completo (Simulador)	147
23	Desempenho - CURUPIRA ₉₆ (Simulador)	148
24	Desempenho x Número de Rounds no CURUPIRA (32 bits)	149
25	Comparação entre velocidades de encriptação/decriptação (32 bits) . .	150
26	Comparação entre velocidades de escalonamento de chaves (32 bits) .	151
27	Comportamento do CURUPIRA para 3 rounds	176
28	Ataque de saturação a 4 rounds do CURUPIRA	177
29	Ataque de saturação a 5 rounds do CURUPIRA	178
30	Ataque de saturação a 6 rounds do CURUPIRA	180

LISTA DE TABELAS

1	CURUPIRA: as mini-boxes P e Q	110
2	CURUPIRA - número de rounds permitido para cada tamanho de chave .	122
3	CURUPIRA: complexidade de ataques de saturação	127
4	Ocupação de memória no (PIC)	141
5	Ocupação de memória no Simulador de Sensor	146
6	A S-Box completa CURUPIRA	184

1 INTRODUÇÃO

Redes sem-fio consistem em uma coleção de nós móveis ou fixos sem conexão física entre si, formando uma topologia dinâmica. Cenários comuns de aplicação incluem redes de telefonia celular e redes de sensores para monitoração de ambientes (neste último caso, os nós são comumente conhecidos como *motes*).

Genericamente, estas redes podem ser classificadas de duas formas: redes “infra-estruturadas” ou redes “*ad-hoc*”. Em redes sem fio infra-estruturadas, os nós têm acesso direto a Pontos de Acesso que são, geralmente, ligados a uma administração central em uma rede fixa com taxas de transmissão superiores às da própria rede sem fio (redes Ethernet, por exemplo). Neste caso, toda a comunicação é feita através deste Ponto de Acesso. Redes *ad-hoc*, por outro lado, são independentes de qualquer infra-estrutura ou administração central. Desta forma, os nós que as compõem desempenham também o papel de roteadores, repassando pacotes de nós vizinhos que não sejam capazes de se comunicar diretamente em razão do alcance limitado de suas interfaces sem-fio.

Redes sem fio são atraentes não apenas pelo seu baixo custo de instalação, principalmente quando comparado com redes cabeadas tradicionais, mas também pela sua grande flexibilidade e escalabilidade. Entretanto, elas possuem algumas peculiaridades importantes. Os nós que fazem parte dela muitas vezes apresentam diversas restrições pouco comuns em computadores modernos, como reduzida disponibilidade de memória e energia, além de baixa capacidade de processamento. Deste modo, é comum

encontrar redes cujos sensores apresentam de 8 a 120 KB de memória de código e 512 a 4096 bytes de memória RAM, bem como velocidade de processamento variando entre 4 e 16 MHz (LAW; DOUMEN; HARTEL, 2006; HILL et al., 2000). Redes *ad-hoc* envolvendo dispositivos móveis, bem como tokens e smart cards, impõem restrições adicionais com relação a tamanho máximo de código e ocupação de banda. Além disto, as mensagens trocadas entre os nós (ou entre os nós e centrais de processamento) costumam ser pequenas, principalmente no caso de redes de sensores, sendo comum que os pacotes apresentem até 30 bytes de comprimento (KARENOS; KALOGERAKI, 2006).

Em razão do constante desenvolvimento e aumento no número de aplicações, bem como da facilidade de acesso ao meio de comunicação utilizado (o ar), fica cada vez mais patente a necessidade de proteção da informação em redes de sensores e redes sem fio de uma forma geral. Contudo, a maioria dos algoritmos criptográficos (também denominados *cifras*) considerados robustos atualmente não responde de forma adequada às necessidades de redes com limitações como as apresentadas. Isto acontece porque os mesmos foram desenvolvidos para propósitos gerais, adaptando-se a diversos cenários, mas não necessariamente de forma otimizada para cada um deles.

1.1 Justificativa

Recentemente, têm sido desenvolvidos diversos estudos voltados à segurança de redes sem fio. O protocolo da camada de enlace TinySec (KARLOF; SASTRY; WAGNER, 2004), desenvolvido especificamente para redes de sensores, é uma das soluções mais populares. De maneira geral, este protocolo se mostra eficiente no fornecimento de acesso ao meio, bem como para garantir a integridade e confidencialidade das mensagens. Todavia, cabe uma crítica à escolha de seu algoritmo criptográfico: por padrão, o TinySec adota o Skipjack (NSA, 1998), uma cifra de bloco que não apenas usa chaves criptográficas consideravelmente pequenas (80 bits), mas também apresenta uma baixa margem de segurança, posto que 31 de seus 32 passos podem ser cripto-analisados

com sucesso (BIHAM; BIRYUKOV; SHAMIR, 1999). Como alternativa, os autores recomendam o algoritmo RC5 (RIVEST, 1995) que, apesar de apresentar um desempenho superior ao Skipjack quando suas sub-chaves de round são pré-computadas, tem sua utilização dificultada pela existência de patentes. Por esta razão, a segurança a longo termo de sistemas baseados neste algoritmo é tema de preocupação.

1.2 Objetivos

Este trabalho visa especificamente ao desenvolvimento de algoritmos criptográficos dedicados a redes com recursos limitados de uma forma geral, dando foco especial a redes de sensores. Desta forma, busca-se desenvolver uma cifra capaz de oferecer maior segurança a um custo (de processamento, memória, energia, etc.) menor ou semelhante àquele obtido com os algoritmos disponíveis atualmente para estas redes. A relação entre o desempenho da cifra proposta e de outros algoritmos importantes, em especial o Skipjack, deve ser comprovada em algumas plataformas consideradas relevantes, especialmente naquelas que apresentam severas restrições de recursos.

1.3 Metodologia

Tendo em vista as restrições intrínsecas a redes móveis e de sensores, a metodologia adotada neste trabalho consiste primeiramente no estudo de técnicas de projeto de cifras de bloco seguras e eficientes, formando assim um arcabouço para a construção de uma cifra adequada a este contexto. Em especial, são analisados os conceitos envolvidos na *Estratégia de Trilha Larga*, metodologia comprovadamente efetiva contra modernas técnicas de ataque (cf. ref.chp:wts) e que dá margem a otimizações de suas operações básicas conforme as necessidades da aplicação.

Fazem parte deste trabalho a especificação, o desenvolvimento, a otimização e a análise de desempenho da cifra proposta, dando-se foco a plataformas com recursos

restritos. No que diz respeito à análise de segurança, algumas das mais efetivas técnicas de ataque a cifras de bloco são analisadas, verificando sua viabilidade contra o algoritmo desenvolvido. Além disto, para que seja possível realizar uma análise completa de seu desempenho, é feita uma análise comparativa entre a cifra desenvolvida neste projeto e outros algoritmos muitas vezes considerados adequados para o mesmo contexto. Finalmente, deve ser adotada uma metodologia de programação bastante similar para os blocos construtivos de todas as cifras testadas, de forma a garantir uma comparação justa entre os algoritmos.

1.4 Contribuições originais

A principal contribuição deste trabalho é a cifra denominada CURUPIRA, especialmente desenvolvida para redes com recursos limitados e seguindo uma metodologia de segurança comprovada. Em razão de seu comportamento involutivo, a mesma exige a implementação apenas do algoritmo de encriptação, que confunde-se com o algoritmo de deciptação. Esta é uma vantagem interessante para qualquer cifra simétrica, assumindo um papel ainda mais importante quando se consideram dispositivos com reduzido espaço de memória para código. Além disto, são propostos dois algoritmos de escalonamentos de chaves, ambos facilmente inversíveis, eliminando-se a necessidade do pré-processamento das sub-chaves de round e, portanto, reduzindo-se a ocupação de memória RAM durante seu processamento. Por fim, as operações utilizadas pela cifra são bastante simples (basicamente OU-exclusivo e deslocamento de bits) e eficientemente implementáveis em plataformas limitadas, como dispositivos equipados com processadores de 8 bits.

1.5 Organização

Este documento é dividido em 9 capítulos. O Capítulo 2 descreve alguns dos conceitos fundamentais de criptografia. Esta discussão inicial é complementada pelo Capítulo 3, que aborda a criptografia simétrica e evidencia as características distintivas entre cifras de bloco e cifras de fluxo. O Capítulo 4 mostra algumas das principais técnicas construtivas de cifras de bloco, que são o foco deste trabalho, juntamente com as mais efetivas técnicas de criptoanálise conhecidas atualmente. Já o Capítulo 5 trata da Estratégia de Trilha Larga, a base para o desenvolvimento de diversas cifras com segurança quantitativamente mensurável, enquanto o Capítulo 6 descreve de forma sucinta o protocolo TinySec. Em seguida, o Capítulo 7 descreve em detalhes o algoritmo adotado pelo CURUPIRA, apresentando também uma análise de segurança e considerações preliminares sobre o desempenho do mesmo. No Capítulo 8 a eficiência deste algoritmo é posta à prova, tanto em plataformas com recursos restritos como em dispositivos poderosos, comparando-se sua ocupação de memória e velocidade com aquelas do Skipjack e do AES. Finalmente, o Capítulo 9 consiste nas conclusões do trabalho.

2 CONCEITOS DE CRIPTOGRAFIA

Para que seja possível compreender as considerações apresentadas neste documento, é necessário ter claros alguns conceitos básicos de Criptografia e sua aplicação. Tendo isto em mente, este capítulo discute brevemente a história da Criptografia, abordando desde seus primórdios até algumas das avançadas técnicas existentes atualmente. Desta forma, deve ficar clara a diferença entre criptografia simétrica e assimétrica, bem como a forma pela qual a utilização destas técnicas é capaz de garantir o estabelecimento de uma comunicação segura.

2.1 Segurança na Comunicação

O termo Criptografia (do grego: *kryptós*, “escondido”, e *gráphein*, “escrever”) é historicamente associado à arte de “esconder informações”, termo que pode ser interpretado como a capacidade de fornecer confidencialidade à informação. Sucintamente, pode-se então definir um algoritmo criptográfico (ou *cifra*) como uma função reversível que transforma *textos claros* (também denominados *mensagens claras*) P em *textos cifrados* (ou *mensagens cifradas*) C e vice-versa, utilizando no processo uma *chave criptográfica* K . Portanto, se Alice quer enviar uma mensagem a Bob (seguindo a tradição de nomes comumente adotados no domínio da Criptografia) de forma segura, mesmo que o canal de comunicação usado seja caracteristicamente inseguro como é o caso da Internet, ela irá encriptar sua mensagem usando uma *função de encriptação* E e uma chave de encriptação K_e . Para ser capaz de decifrar e ler a mensagem de Alice,

Bob irá utilizar uma *função de decifração* D e também uma chave de decifração K_d . Um terceiro indivíduo que não tenha acesso a estas chaves deve então ser incapaz de espreitar o diálogo.

Apesar da confidencialidade ser um dos principais domínios de aplicação da Criptografia, atualmente ela apresenta um campo mais amplo, respondendo às seguintes necessidades:

- **Confidencialidade:** garantia de que as mensagens trocadas poderão ser compreendidas somente pelos usuários desejados, de tal forma que apenas eles sejam capazes de extrair a informação nelas contida.
- **Integridade:** possibilidade de verificar a consistência da informação contida em uma mensagem. Tal serviço não garante que as mensagens não sejam alteradas durante a transmissão, mas sim que a ocorrência da alteração possa ser detectada.
- **Autenticação:** possibilidade de comprovar a identidade de um indivíduo que participa da comunicação.
- **Irretratabilidade:** garantia de que nem o remetente nem o destinatário de uma determinada mensagem possam negar sua transmissão, recepção ou posse.

2.2 Breve história da criptografia

A história da criptografia está intimamente ligada àquela da escrita. Os primeiros traços de sua utilização datam de 2000 A.C., no Egito. Durante séculos, a criptografia se baseou em dois grandes princípios:

- **Substituição:** Em um alfabeto, pode-se trocar um conjunto de letras entre si. O mais famoso exemplo desta técnica de encriptação é provavelmente o alfabeto de César, que consiste na substituição de cada letra pela K -ésima letra sucessiva

em um alfabeto cíclico (a letra “z” sendo seguida pela letra “a”), havendo assim um número de possibilidades de substituição igual ao número de letras do alfabeto. Uma evolução desta última, conhecida como Cifra de Vigenère, consiste na utilização de um conjunto de valores repetidos ciclicamente ao invés de um único valor K ; com duas chaves K_1 e K_2 , por exemplo, as letras localizadas em uma posição ímpar seriam substituídas pela letra localizada K_1 posições adiante, enquanto K_2 seria usada nas posições pares.

- **Permutação:** modifica-se a ordem das letras do texto claro. O mais antigo exemplo de utilização desta técnica é atribuído aos Espartanos, na antiga Grécia, que utilizavam-se de um bastão conhecido pelo nome de *skytale* ou *scytale*. O processo de encriptação consistia em enrolar uma faixa de pergaminho no skytale, escrevendo então a mensagem clara sobre este pergaminho. Para decifrar o texto, bastava enrolar a faixa em um outro skytale de mesmo tamanho, alinhando o texto de forma a torná-lo legível (KELLY, 1998).

No contexto dos documentos digitais, as “palavras” de uma mensagem foram substituídas pelos seus bits ou bytes. Entretanto, as técnicas de substituição e permutação continuam perfeitamente válidas. É assim que, em um trabalho de 1949 na área da teoria da informação, o engenheiro e matemático Claude Shannon (SHANNON, 1949) discorreu sobre a fundamental importância dos mesmos, em termos dos conceitos denominados *confusão* e *difusão*, respectivamente. A confusão refere-se à inserção de não-linearidade na cifra, aumentando então a complexidade da estrutura ligando os textos claro e cifrado. Já o segundo implica que cada bit do texto cifrado seja dependente do maior número possível de bits do texto claro e da chave utilizada, levando à dissipação de qualquer informação estatisticamente mensurável porventura presente nos textos claros. No mesmo trabalho, Shannon enuncia também o que ele chama de *Segredo Perfeito*: a probabilidade de se obter um texto claro a partir de quaisquer conjuntos de textos cifrados deve ser a mesma, ou seja, a dificuldade de se obter textos

claros deve ser independente do número de textos cifrados de que dispõe um atacante. Ele também mostra que a combinação da mensagem original com uma chave completamente aleatória de mesmo tamanho satisfaz este requisito, resultando em uma cifra de segurança máxima, denominada *One-Time Pad*. No entanto, como as chaves só podem ser utilizadas uma única vez e são potencialmente grandes, a utilização de tal cifra na maioria das aplicações práticas é inviável (veja seção 3.1).

Com o desenvolvimento dos computadores e de redes de telecomunicações ao longo das últimas décadas, a criptografia deixou de ser restrita aos meios diplomático e militar, permitindo a criação de diversas cifras e também maneiras de ataques às mesmas. É desta forma que, em 1975, foi promovido um concurso semi-público para a escolha de padrão criptográfico destinado a aplicações civis. O vencedor do concurso foi um algoritmo proposto pela IBM e alterado pela Agência de Segurança Nacional americana (*National Security Agency* - NSA) segundo critérios não divulgados na época, dando origem ao DES - *Data Encryption Standard* (Padrão de Encriptação de Dados) (NIST, 1977). O projeto da cifra baseia-se na chamada Estrutura de Feistel (cf. seção 4.2), sendo efetivamente adotado como padrão de encriptação pelo governo americano em 1977. Entretanto, o DES é atualmente considerado obsoleto, principalmente pelo seu reduzido tamanho de chave (56 bits), pouco segura considerando a capacidade computacional disponível atualmente, e pelo seu pequeno tamanho de bloco (64 bits), que pode facilitar ataques futuros.

Apesar de o DES ter sido definitivamente aposentado pelo Instituto Nacional de Padrões e Tecnologia americano (*National Institute of Standards and Technology* - NIST) em 2004, foi criada uma solução paliativa para aproveitar a ampla base existente de implementações desta cifra em hardware e software, conhecida como 3-DES (ou DES Triplo). Ele consiste em efetuar por 3 vezes a encriptação usando o DES simples, com 3 chaves distintas de 56 bits, K_1 , K_2 e K_3 , resultando em uma segurança aproximadamente equivalente à de uma cifra com chave de 112 bits. Existe ainda

uma variante bastante popular que consiste em utilizar a chave K_2 com o algoritmo de decifração, de forma que o 3-DES executa a sequência encriptação-decriptação-encriptação (EDE). Apesar de esta construção não ser diferente em termos de segurança, ela é interessante pela possibilidade de simular o DES simples usando o 3-DES com 3 chaves idênticas.

O padrão de encriptação atual foi definido em outubro do ano 2000, após um novo concurso (desta vez verdadeiramente público) patrocinado pelo NIST, que teve início oficial em junho de 1997. Dentre os 5 finalistas, o algoritmo Rijndael (DAEMEN; RIJMEN, 2002), desenvolvido pelos criptógrafos belgas Joan Daemen e Vincent Rijmen, foi escolhido para ser o novo AES - *Advanced Encryption Standard* (Padrão Avançado de Encriptação) (NIST, 2001a). O Rijndael foi desenvolvido a partir de um outro algoritmo chamado de SQUARE (DAEMEN; KNUDSEN; RIJMEN, 1997), dos mesmos autores, e baseia-se na Estratégia de Trilha Larga (cf. Capítulo 5), método sistemático de implementar eficientemente os princípios de confusão e difusão.

2.3 Dois métodos para trocar informações

Formalmente, os processos de encriptação e decifração podem ser definidos da seguinte forma: seja $\mathcal{P}$ um conjunto de textos claros, $\mathcal{C}$ um conjunto de textos cifrados e $\mathcal{K}$ um conjunto de chaves. Uma função de encriptação E associa a cada par $(P, K_e) \in \mathcal{P} \times \mathcal{K}$ um elemento $C = E(P, K_e) \in \mathcal{C}$. A função de decifração D correspondente, juntamente com uma chave de decifração K_d , associa ao par $(C, K_d) \in \mathcal{C} \times \mathcal{K}$ o mesmo texto $P \in \mathcal{P}$, de forma que $D(E(P, K_e), K_d) = P$.

2.3.1 Criptografia Simétrica

Até o ano de 1976, a única forma conhecida de criptografia era a *criptografia de chave secreta*, ou *criptografia simétrica*: para que dois indivíduos pudessem se comu-

nicar de forma segura, ambos precisavam compartilhar uma mesma chave secreta para encriptação e decrptação, conhecida apenas por eles. Isto gerava problemas principalmente com relação à distribuição destas chaves, que precisavam ser trocadas através de um meio seguro (em pessoa, por exemplo) antes que fosse possível utilizar de qualquer forma de criptografia.

Portanto, se Alice e Bob decidem utilizar um esquema de encriptação simétrica para se comunicar, eles devem partilhar uma mesma chave K , conhecida apenas por eles. Esta chave será usada tanto na operação de encriptação quanto decrptação, ou seja, $K_e = K_d = K$. A figura 1 ilustra o processo. O AES e o DES são dois exemplos de algoritmos que adotam o esquema de criptografia simétrica.

Figura 1: Criptografia Simétrica

2.3.2 Criptografia Assimétrica

Apesar de algumas fontes alegarem que este tipo de criptografia já era conhecido no meio militar antes desta data (WILLIAMSON, 1974), foi apenas em 1976 que os criptógrafos ingleses Diffie e Hellman (DIFFIE; HELLMAN, 1976) apresentaram ao meio civil o esquema conhecido como *criptografia assimétrica* (ou *criptografia de chave pública*). Sua utilização permite que indivíduos estabeleçam uma comunicação segura sem a necessidade de um compartilhando prévio de chave criptográfica.

Portanto, são usadas duas chaves diferentes para encriptação e decrptação: uma chave pública K_u e sua correspondente chave privada K_r . Quando Alice deseja enviar uma mensagem confidencial P a Bob, ela deve encriptar a mensagem usando a chave

pública de Bob, que pode ser encontrada abertamente na Internet ou junto a uma entidade com esta atribuição (Entidade Certificadora), por exemplo. Desta forma, é gerada uma mensagem cifrada C que apenas Bob é capaz de decifrar, posto que ele é o único que conhece sua chave privada. A figura 2 ilustra este processo. Um dos exemplos mais conhecidos deste tipo de cifra é o algoritmo RSA (RIVEST; SHAMIR; ADELMAN, 1977), cuja segurança se baseia na dificuldade computacional de se fatorar números grandes.

Figura 2: Criptografia Assimétrica

2.3.3 Simetria × Assimetria

Existe um grande interesse na utilização de criptografia de chave pública, pela sua capacidade de prover diversas funcionalidades essenciais no domínio da segurança da informação. Dentre as suas aplicações, destacam-se: assinatura digital, autenticação de usuários e distribuição de chaves simétricas.

Uma assinatura digital é uma forma de comprovar a autoria de uma mensagem ou documento. Sucintamente, o procedimento para sua geração consiste no cálculo de um resumo criptográfico de tamanho fixo (também conhecido como **hash**) da mensagem que se deseja assinar. Isto pode ser feito utilizando, por exemplo, uma função da família SHA (NIST, 2002). O **hash** é então encriptado com a chave privada do autor da mensagem. O resultado é a assinatura digital da mesma: como apenas este indivíduo tem acesso à sua chave privada, apenas ele seria capaz de criar tal assinatura, que pode ser verificada por meio de sua chave pública. Assim, caso Bob receba uma mensagem

supostamente assinada por Alice, ele pode decriptá-la com a chave pública de Alice e comparar o resultado com o *hash* da mensagem recebida. Caso os mesmos sejam idênticos, pode-se concluir que aquele documento foi realmente assinado por Alice; caso contrário, é possível que a assinatura não pertença a Alice (por exemplo, Eva pode tê-lo assinado) ou que o documento foi alterado durante a transmissão (acidental ou intencionalmente). O fato do *hash* da mensagem ser assinado ao invés da mensagem completa está relacionado a questões de desempenho: não importando o tamanho da mensagem, a encriptação se dá sobre dados de tamanho fixo e arbitrariamente pequeno.

O processo de autenticação é muito semelhante à assinatura digital, porém possui uma sutil (e importante) diferença: enquanto a assinatura digital prova que um documento pertence a uma determinada pessoa, mesmo muito tempo após a geração do mesmo, o processo de autenticação visa à identificação dos interlocutores antes do início da comunicação. Desta forma, quando Alice assina uma mensagem de apresentação usando sua chave privada, Bob pode ter certeza que é com ela (e não com Eva) que ele está iniciando uma conversa.

Por fim, a distribuição de chaves simétricas significa que estas últimas podem ser compartilhadas por Alice e Bob sem a necessidade de um canal de comunicação seguro: basta Alice enviar a Bob uma mensagem contendo a chave simétrica desejada, encriptada com um algoritmo assimétrico e a chave pública de Bob. Desta forma, Bob será o único capaz de recuperar a chave simétrica enviada por Alice. A razão pela qual o próprio algoritmo de chave pública não é utilizado durante toda a comunicação é simples: desempenho. É fato conhecido que algoritmos assimétricos apresentam um desempenho bem inferior àquele dos algoritmos simétricos. O RSA, por exemplo, é de 100 a 1000 vezes mais lento do que o DES e usa chaves bem maiores, sendo 1024 bits um tamanho bastante comum. Portanto, após as fases iniciais de comunicação usando algoritmos de chave pública, os pares comunicantes geralmente passam a utilizar um algoritmo simétrico com a chave compartilhada.

2.4 Resumo

Este capítulo abordou alguns dos conceitos mais básicos e essenciais relativos à área de estudo da Criptografia. Além de um breve histórico, foram enumerados os serviços básicos aos quais a mesma se presta, a saber: confidencialidade, integridade, autenticação e irretratabilidade. Foi ainda explicada a diferença entre Criptografia Simétrica e Assimétrica, e como as mesmas são utilizadas no sentido de prover estes serviços.

Como o presente estudo tem como objetivo o desenvolvimento de cifras simétricas, com foco especial àquelas destinadas a cenários com recursos restritos, este será o foco no restante deste documento. É desta forma que o Capítulo 3 apresentará uma análise mais detalhada dos algoritmos desta família, esclarecendo a diferença entre Cifras de Fluxo e Cifras de Bloco. Para uma discussão mais profunda sobre criptografia assimétrica, recomenda-se a leitura de (SCHÄFER, 2004) ou (THORSTEINSON; GANESH, 2003).

3 TIPOS DE CIFRAS SIMÉTRICAS

No capítulo anterior, foram apresentados os conceitos básicos referentes a cifras simétricas, que utilizam uma mesma chave para encriptar e decriptar mensagens e costumam ser bem mais rápidas do que cifras assimétricas. Apesar de seu grande número, as cifras simétricas podem ser agrupadas em duas grandes classes principais, denominadas *cifras de fluxo* e *cifras de bloco*. Como as características que as distinguem são importantes em termos de segurança e cenários de aplicação, elas são discutidas a seguir.

3.1 Cifras de Fluxo

Cifras de fluxo (também conhecidas como *cifras de mascaramento*) consistem na geração de uma sequência de bits pseudo-aleatória, dependente da chave utilizada, que é combinada com o texto claro de entrada. A forma mais comum de combinar os bits da mensagem clara e da chave criptográfica é por meio de uma operação de OU-exclusivo (denotado XOR e representado como $\oplus$). Exemplos de algoritmos que se enquadram nesta categoria incluem Helix (FERGUSON et al., 2003), Scream (HALEVI; COPPERSMITH; JUTLA, 2002) e Turing (ROSE; HAWKES, 2002).

Na verdade, assim como ocorre com o One-Time Pad citado na seção 2.2, poderia ser empregada uma chave criptográfica completamente aleatória, não reutilizável e apresentando o mesmo tamanho da mensagem clara. Apesar da quebra deste esquema exigir que todas as chaves possíveis sejam testadas (cf. 4.3.1), sua utilização

seria muito dispendiosa na maioria dos casos reais. Para criptografar o conteúdo de um DVD, por exemplo, seriam necessários dois DVDs: o primeiro para o conteúdo criptografado e o outro apenas para armazenar a chave capaz de decriptar este conteúdo. Por esta razão, na prática é utilizada uma chave inicial de tamanho reduzido (128 bits, por exemplo), a partir da qual são derivadas sub-chaves pseudo-aleatórias em número adequado para cobrir toda a mensagem. Este é o caso da Cifra de Vernam, que faz o XOR de cada bit da mensagem original P com um bit correspondente do fluxo pseudo-aleatório de chaves Φ , o qual é gerado a partir da chave inicial K . O resultado é a mensagem cifrada C . A mesma pode então ser descrita da seguinte maneira (o índice i corresponde ao i -ésimo bit):

$$\text{(criptação)} \ E(P, K) : C_i = P_i \oplus \Phi_i(K), \ i \geq 0$$

$$\text{(decriptação)} \ D(C, K) : P_i = C_i \oplus \Phi_i(K), \ i \geq 0$$

Desta forma, ganha-se em desempenho ao custo de uma redução na margem de segurança: evitam-se a armazenagem e distribuição de chaves muito grandes, mas não é mais possível garantir que tal algoritmo apresente o mesmo grau de segurança absoluta do One-Time Pad.

De acordo com o modo pelo qual o fluxo Φ é gerado, distinguem-se duas categorias de cifras de fluxo: *cifras de fluxo síncronas* e *cifras de fluxo auto-síncronas*.

3.1.1 Cifras de Fluxo Síncronas

Neste caso, o fluxo pseudo-aleatório de chaves é gerado independentemente das mensagens original ou cifrada, sendo construído unicamente a partir de uma chave criptográfica inicial K e de um *vetor de inicialização* IV .

Para que a deciptação seja possível, além de utilizar os mesmos valores de K e IV , emissor e receptor devem estar sincronizados, gerando assim dois fluxos de chaves idênticos. Caso haja inserção ou remoção de bits na mensagem durante a transmis-

são, entretanto, a sincronização é perdida, podendo ser restaurada por técnicas como a utilização de marcadores em espaçamentos regulares da mensagem. Já no caso de ocorrer uma alteração nos bits em trânsito, o erro não se propaga para o restante da mensagem, uma característica interessante no caso de meios onde existe alta taxa de erros. Contudo, isto faz com que esta categoria de cifras seja suscetível aos chamados *ataques ativos*, posto que a inversão intencional de um único bit na mensagem cifrada causará uma inversão na mesma posição da mensagem decifrada correspondente.

A função de encriptação das cifras de fluxo síncronas escreve-se como:

$$\sigma_0 = J(IV, K)$$

$$\sigma_{i+1} = F(\sigma_i, K)$$

$$\Phi_i = G(\sigma_i, K)$$

$$C_i = E(P_i, \Phi_i)$$

Onde σ_0 indica o estado inicial da cifra, gerado a partir de IV e K por meio de uma transformação inicial J , e evoluindo pela aplicação de uma função F . Estes estados são usados para gerar o fluxo de chaves Φ através de uma função G . Este fluxo passa então por uma função de encriptação E (normalmente um simples XOR) na qual é combinado com a mensagem clara para gerar o texto cifrado. Novamente, o índice i indica o i -ésimo bit da mensagem ou chave. O processo é ilustrado na Figura 3.

3.1.2 Cifras de Fluxo Auto-síncronas

Para esta classe de cifras, o fluxo de chaves é gerado tanto a partir de K e IV quanto de um número n_f de grupos de bits (bytes, por exemplo) encriptados anteriormente. Assim, mesmo que haja a perda de sincronismo entre emissor e receptor, este último é capaz de se recuperar após receber um número suficiente de grupos de bits cifrados. Já no caso da alteração de um bit durante a transmissão, até n_f grupos de bits serão afetados (ao invés de um único bit), dificultando possíveis ataques ativos.

Figura 3: Operação de uma Cifra de Fluxo Síncrona

A função de encriptação das cifras de fluxo auto-síncronas pode então ser descrita da seguinte maneira:

$$\sigma_0 = (C_{-n_f}, C_{-n_f+1}, \dots, C_{-1}) = IV$$

$$\sigma_i = (C_{i-n_f}, C_{i-n_f+1}, \dots, C_{i-1})$$

$$\Phi_i = G(\sigma_i, K)$$

$$C_i = E(P_i, \Phi_i)$$

Neste caso, o estado inicial σ_0 é gerado a partir de IV , que é utilizado para alimentar inicialmente o *buffer* de mensagens cifradas. Este *buffer* é usado para gerar o fluxo de chaves Φ de acordo com uma função G e sob a ação da chave K . A função de encriptação E fica então responsável por combinar a mensagem clara com este fluxo de chaves, gerando as mensagens cifradas que são armazenadas no *buffer*, dando origem aos estados σ_i seguintes. Este processo é ilustrado na Figura 4.

3.1.3 Utilização e Segurança

Cifras de fluxo são potencialmente mais rápidas do que cifras de bloco (VENUGO-PALAN et al., 2003). Por esta razão, elas costumam ser bastante utilizadas em sistemas

Figura 4: Operação de uma Cifra de Fluxo Auto-síncrona

que exijam alta velocidade de operação ou nos quais haja escassez de recursos como memória e processamento. Além disto, devido à sua propagação de erros bem controlada, elas podem também ser vantajosas em ambientes nos quais as taxas de erros sejam elevadas.

Todavia, percebe-se que a segurança desta classe de cifras depende fortemente do período do fluxo de chaves gerado (BELLOVIN; BLAZE, 2001). Isto ocorre porque, após um certo número de bits encriptados, invariavelmente haverá a repetição da sequência de bits no fluxo de chaves. Desta forma, um atacante pode recuperar duas mensagens cifradas C^1 e C^2 , às quais foi aplicado uma mesma sequência pseudo-aleatória de chaves Φ , e combiná-las para remover a encriptação:

$$C^1 \oplus C^2 = P^1 \oplus \Phi \oplus P^2 \oplus \Phi = P^1 \oplus P^2$$

Assim, caso a segurança de uma mensagem clara seja comprometida, um atacante pode trivialmente recuperar todas as mensagens claras encriptadas com a mesma chave, mesmo sem conhecer o valor desta última. Esta propriedade faz com que as cifras de fluxo sejam pouco recomendadas para aplicações onde a repetição de chaves

após um curto período seja uma realidade, como costuma ser o caso de redes de sensores (cf. Capítulo 6). Nestes casos, a melhor alternativa é a utilização de cifras de bloco em um modo de operação adequado, conforme discutido a seguir.

3.2 Cifras de Bloco

Uma cifra de bloco opera sobre grupos de bits de tamanho n_b fixo, denominados **blocos**, que passam separadamente por um mesmo processo de encriptação. Desta forma, sob a ação de uma função de encriptação E e de uma chave criptográfica K de n_k bits, blocos da mensagem original P são transformados em blocos cifrados apresentando o mesmo tamanho n_b . Para recuperar a mensagem original, utiliza-se uma função de deciptação D , a mesma chave de encriptação K e a mensagem cifrada C , que também é deciptada bloco a bloco. Exemplos deste tipo de cifra incluem DES, AES e Skipjack.

As operações de encriptação e deciptação podem ser escritas genericamente como:

$$\text{(encriptação)} \ E_k : C = E(P, K)$$

$$\text{(decriptação)} \ D_k : P = D(C, K)$$

Portanto, tem-se que $D_k = E_k^{-1}$, sendo então necessário que a função E_k seja bijetora para todo K .

A maioria das cifras de bloco adotam uma estrutura iterativa (em **rounds**), significando que as operações de encriptação e deciptação são compostas por sub-operações idênticas (**funções de round**), cada qual utilizando um conjunto de bits como chave criptográfica (**sub-chaves** de round). Os blocos de entrada e saída de cada função de round são denominados **estados**. De maneira semelhante ao que ocorre com cifras de fluxo (cf. seção 3.1), é gerada uma seqüência de sub-chaves a partir de uma chave inicial, processo conhecido como **escalonamento de chaves**. No caso de cifras iterativas,

as funções de E e D podem então ser escritas a partir de suas funções de round ρ e de suas sub-chaves K_i , onde o índice i indica o i -ésimo round da cifra. Para 4 rounds, por exemplo, pode-se escrever:

$$\text{(criptação)} E_k : C = E(P, K) = \rho(\rho(\rho(\rho(P, K_1), K_2), K_3), K_4)$$

$$\text{(decriptação)} D_k : P = D(C, K) = \rho^{-1}(\rho^{-1}(\rho^{-1}(\rho^{-1}(C, K_4), K_3), K_2), K_1)$$

A estrutura exata de cada round depende essencialmente do projeto da cifra, sendo fortemente influenciada pelos conceitos de confusão e difusão discutidos no Capítulo 2. Além disto, a estrutura de cifras modernas costuma levar em consideração os diversos tipos de ataques conhecidos, apresentados com maiores detalhes no Capítulo 4.

3.2.1 Modos de Operação

Cifras de bloco podem operar apenas em blocos completos de n_b bits. Portanto, é necessário considerar o que ocorre quando o tamanho da mensagem clara não corresponde exatamente a um múltiplo do tamanho do bloco, uma situação bastante comum.

No caso de uma mensagem clara menor do que o tamanho de um bloco, ela deve passar pelo processo conhecido como *padding*, que visa ao seu preenchimento com um número de bits suficiente para que seu tamanho atinja n_b bits. Uma alternativa comum é o *padding* do tipo PKCS, que acrescenta um total de a bytes com o valor a à mensagem; outra possibilidade, recomendada pelo NIST (NIST, 2001b), acrescenta um único bit ‘1’ seguido de bits ‘0’ suficientes para completar o bloco. Em qualquer caso, há uma indesejada (porém necessária) expansão da mensagem original.

Por outro lado, caso a mensagem seja maior do que n_b bits, diversos mecanismos podem ser utilizados, conhecidos coletivamente como *modos de operação*. Os mais comuns, atualmente padronizados pelo NIST, são o ECB, CBC, CFB, OFB e CTR, descritos a seguir.

3.2.1.1 ECB (Electronic Code Book)

Neste modo de operação, a mensagem é quebrada em diversos blocos de n_b bits, os quais são encriptados de forma totalmente independente. Tal simplicidade faz com que este seja o mais rápido dos modos de operação, possibilitando inclusive o processamento dos blocos seja feito em paralelo, além de garantir uma baixa propagação de erros. Entretanto, isto também impacta a segurança deste modo, pois blocos iguais invariavelmente resultarão em blocos cifrados idênticos. Este problema adquire maior seriedade no caso de redes onde a variabilidade das mensagens é pequena (redes nas quais as mensagens trocadas são apenas do tipo ‘sim’ ou ‘não’, por exemplo), o que permitiria a um atacante reconhecer com alta probabilidade as mensagens claras, bastando observar as mensagens cifradas transmitidas. Desta forma, apesar de prevenir que adversários recuperarem a mensagem original, ele falha em garantir que os mesmos não consigam recuperar qualquer tipo de informação parcial a partir de uma mensagem cifrada, requisito conhecido como segurança semântica (BELLARE; ROGAWAY, 1993). Os processos de encriptação e decifração no modo ECB são ilustrados na Figura 5.

Figura 5: Encriptação e Decifração usando o Modo de Operação ECB

É importante notar que o último dos blocos resultante do processo de quebra não apresentará exatamente n_b bits a não ser que o tamanho da mensagem original seja um múltiplo inteiro de n_b . Para tratar este bloco incompleto, portanto, pode-se adotar um mecanismo de *padding*, ocasionando na expansão da mensagem cifrada resultante.

Alternativamente, uma técnica mais sofisticada conhecida como *Ciphertext Stealing* pode ser empregada. A mesma será discutida em detalhes na seção 3.2.2.

O modo de operação ECB assume, portanto, a forma geral bastante simples a seguir:

$$C_i = E(P_i, K)$$

$$P_i = D(C_i, K)$$

3.2.1.2 CBC (Cipher Block Chaining)

De maneira semelhante ao modo ECB, a mensagem original é quebrada em blocos de n_b bits. Contudo, antes que estes blocos sejam encriptados, eles são combinados com o último bloco já encriptado, por meio de uma operação de XOR. Para o primeiro destes blocos (que não possui, portanto, um antecessor encriptado), a operação de XOR é feita com o vetor de inicialização *IV*, gerado com este propósito específico. O processo é ilustrado na Figura 6.

Figura 6: Encriptação e Decriptação usando o Modo de Operação CBC

Este modo introduz maior variabilidade aos blocos cifrados mesmo quando as mensagens claras são idênticas (usando *IVs* diferentes) ou quando ocorre reutilização de *IVs* (porém as mensagens são distintas), resultando em uma maior grau de segurança. Todavia, como a encriptação deve ser feita sequencialmente, o processamento dos blocos em paralelo fica impossibilitado; já o processo de deciptação pode ser re-

alizado em paralelo, posto que o mesmo depende apenas de dois blocos adjacentes. Percebe-se ainda que a ocorrência de erros em um dos blocos afetará todos os blocos subsequentes. Além disto, como ocorre com o modo ECB, caso o tamanho da mensagem clara não corresponda a um múltiplo do tamanho do bloco, o último bloco da mensagem deve ser tratado por meio de técnicas como *padding* ou *Ciphertext Stealing*.

Formalmente, podem-se exprimir as operações de encriptação e decríptação da seguinte forma (considerando que o primeiro bloco recebe o índice 1):

$$C_i = E(P_i \oplus C_{i-1}, K)$$

$$P_i = D(C_i, K) \oplus C_{i-1}$$

$$C_0 = IV$$

3.2.1.3 CFB (Cipher Feedback)

Este modo de operação, apesar de semelhante ao CBC, faz com que a cifra de bloco se comporte como uma cifra de fluxo auto-síncrona (cf. seção 3.1.2). Assim, as mensagens não precisam ser quebradas em blocos de tamanho fixo, uma vantagem significativa em diversas aplicações. Seu funcionamento se dá da seguinte forma: um bloco inicial de n_b bits (o vetor de inicialização IV) é encriptado normalmente, resultando em um bloco encriptado S ; em seguida, um número $n_p \leq n_b$ de bits de S é selecionado e sofre uma operação de XOR com uma porção de tamanho n_p da mensagem clara, resultando na porção de mensagem cifrada que é a saída do algoritmo, C . O processo é então repetido utilizando-se C como novo vetor de inicialização, até que a mensagem seja encriptada por completo. A encriptação e decríptação com o modo CFB são mostradas na Figura 7.

É interessante notar que este modo de operação requer apenas a implementação do algoritmo de encriptação, já que o mesmo é utilizado tanto para encriptar quanto para decríptar a mensagem. Por outro lado, observa-se que o processo de encriptação é seqüencial, impossibilitando o processamento dos blocos em paralelo. Além disto,

Figura 7: Encrytação e Decrytação usando o Modo de Operação CFB

caso um bloco seja alterado durante a transmissão, este erro se propaga para todos os blocos subseqüentes e, de maneira semelhante às cifras de fluxo, a segurança do modo CFB também depende da não-repetição de vetores de inicialização.

A encrytação e a decrytação no modo CFB podem ser escritas como segue (considerando novamente que o primeiro bloco recebe o índice 1):

$$C_i = E(S_{i-1}, K) \oplus P_i$$

$$P_i = E(P_{i-1}, K) \oplus C_i$$

$$C_0 = IV$$

3.2.1.4 OFB (Output Feedback)

O modo de operação OFB resulta em um comportamento similar ao de uma cifra de fluxo síncrona. De forma semelhante ao modo CFB, um vetor de inicialização IV de n_b bits é encriptado com uma chave criptográfica K , dando origem a um bloco S ; um número $n_p \leq n_b$ de bits de S é então combinado com n_p bits da mensagem original por meio de uma operação de XOR, resultando na mensagem cifrada C . Entretanto, contrariamente ao modo CFB, o bloco S é aquele utilizado como novo IV para o passo seguinte. O processo é então repetido até que toda a mensagem seja encriptada. O modo OFB é ilustrado na Figura 8.

Analogamente ao modo CFB, o uso deste modo de operação dispensa a implemen-

Figura 8: Encriptação e Decifração usando o Modo de Operação OFB

tação do algoritmo de decifração, posto que a cifra subjacente é utilizada apenas no sentido de encriptação. Apesar de sua segurança depender fortemente da não-repetição de vetores de inicialização (assim como o modo CFB e toda a família de cifras de fluxo), este modo apresenta uma característica interessante: como as mensagens claras e cifradas atuam somente na última operação da cifra (o XOR final), toda a sequência de chaves pode ser pré-processada a partir do conhecimento da chave criptográfica K e do vetor de inicialização IV . Desta forma, é possível que a mensagem seja processada em paralelo, bastando para isso gerar o fluxo de chaves e efetuar a encriptação ou decifração assim que os bits da mensagem estejam disponíveis.

As operações de encriptação e decifração assumem a seguinte forma (considerando ainda que o primeiro bloco recebe o índice 1).

$$C_i = P_i \oplus S_i$$

$$P_i = C_i \oplus S_i$$

$$S_i = E(S_{i-1}, K)$$

$$S_0 = IV$$

3.2.1.5 CTR (Counter Mode)

O modo conhecido por CTR é muito parecido com o modo OFB, apresentando também o comportamento de uma cifra de fluxo síncrona. A diferença fica por conta

da entrada que é encriptada e combinada (via operação de XOR) com o texto claro: ao invés de usar a mensagem cifrada anteriormente (como o modo CFB) ou a saída da operação de encriptação (como o OFB), o valor R de um contador é sempre utilizado. Desta forma, não há realimentação no processo, ou seja, cada uma das porções de texto claro (de até n_b bits) é encriptada separada e independentemente. Mais uma vez, apenas o algoritmo de encriptação precisa ser implementado. A Figura 9 ilustra o funcionamento do modo de operação CTR.

Figura 9: Encriptação e Decifração usando o Modo de Operação CTR

As operações de encriptação e decifração podem então ser descritas por:

$$C_i = P_i \oplus S_i$$

$$P_i = C_i \oplus S_i$$

$$S_i = E(Ctr_i, K)$$

$$Ctr_i = Ctr_{i-1} + 1$$

$$Ctr_0 = 0$$

Apesar de o contador sequencial ter sido iniciado em '0', esquemas de contagem diferentes podem ser utilizados, como um contador com inicialização aleatória, por exemplo.

3.2.2 Ciphertext Stealing

Conforme discutido anteriormente, a utilização dos modos de operação ECB (seção 3.2.1.1) e CBC (seção 3.2.1.2) pode acarretar em expansão do texto cifrado caso o tamanho n_p do texto claro subjacente não corresponda exatamente a um múltiplo de n_b , ou seja, caso $(n_p \bmod n_b \neq 0)$. Para resolver este problema de expansão, indesejável na maioria das aplicações, foi desenvolvida a técnica conhecida como *Ciphertext Stealing* (CTS, ou “Roubo de Texto Cifrado”). Sua utilização permite que mensagens de qualquer tamanho sejam encriptadas sem que haja expansão das mesmas, ao custo de um aumento na complexidade dos processos de encriptação e decríptação.

Resumidamente, a técnica consiste em alterar o processamento dos dois últimos blocos do texto claro e na reordenação dos mesmos antes da transmissão, conforme a seqüência:

1. O último bloco do texto claro (possivelmente incompleto) é preenchido com os bits de ordem mais baixa pertencentes ao penúltimo bloco cifrado, “roubando” assim estes bits;
2. O último bloco, agora completo, é encriptado;
3. A ordem dos dois últimos blocos é invertida;
4. O último bloco resultante deste processo é truncado de forma que seu tamanho corresponda àquele do antigo último bloco, removendo-se assim os bits roubados. O resultado é um texto cifrado com o mesmo tamanho do texto claro original.

Portanto, apenas o processamento dos dois últimos blocos do texto é alterado. É importante ressaltar, entretanto, que a técnica só pode ser usada no caso de mensagens cujo tamanho total seja maior do que um bloco. A figura 10 ilustra a técnica para ambos os modos ECB e CBC.

Figura 10: Ciphertext Stealing para os modos ECB e CBC

3.2.3 Utilização e Segurança

Cifras de bloco são os mais proeminentes e importantes elementos em muitos sistemas criptográficos. Individualmente, eles provêm confidencialidade; sua versatilidade permite a construção de geradores de números pseudo-aleatórios (o modo CTR é comumente utilizado para este fim), cifras de fluxo (modos OFB, CFB e CTR) e funções de *hash*. Esta família de cifras é muito utilizada em técnicas de autenticação de mensagens, mecanismos para garantia de integridade, protocolos de autenticação e também esquemas de assinatura digital por chave simétrica. No entanto, é importante ressaltar que no domínio da criptografia não há panacéia: não existe um cifra, seja ela de bloco ou de fluxo, que seja a solução ideal para toda e qualquer aplicação, mesmo que a segurança por ela oferecida seja de alto nível. Na prática, existem diversos fatores (como restrição de recursos em uma plataforma, desempenho, compatibilidade com outros sistemas, etc.) que influenciam na escolha pela cifra mais adequada para cada aplicação.

São conhecidas atualmente técnicas para o projeto de cifras de bloco seguras, o que constitui em um fator decisivo para sua expansão. O mesmo não ocorre com cifras de fluxo, razão pela qual muitas aplicações em que estas últimas seriam adequadas preferem adotar cifras de bloco seguras em modos de operação de fluxo.

3.3 Resumo

Este capítulo apresentou os dois tipos principais de cifras simétricas existentes atualmente: Cifras de Fluxo e Cifras de Bloco. O funcionamento das cifras de fluxo consiste na geração de uma sequência de bits pseudo-aleatória (dependente da chave) que é diretamente combinada com os bits da mensagem, normalmente por meio de operações simples, como XOR bit-a-bit. Elas podem ainda ser classificadas em Síncronas e Auto-Síncronas, dependendo da forma como esta sequência de chaves é gerada. Já as cifras de bloco operam necessariamente sobre blocos de bits (conjuntos de tamanho fixo) e exigem, então, o uso de técnicas especiais para encriptar mensagens cujo tamanho não corresponda exatamente ao tamanho do bloco, como o uso de Modos de Operação (dentre os quais se destacam ECB, CBC, CFB, OFB e CTR), aplicação de *Padding* e *Ciphertext Stealing*.

Foram ainda discutidas algumas das vantagens e desvantagens de cada uma destas famílias de cifras simétricas. Enquanto cifras de fluxo são potencialmente mais rápidas do que cifras de bloco, estas últimas têm a vantagem de serem conhecidos métodos para sua construção segura, algo que não ocorre com a outra família. Faz-se necessário, portanto, um estudo mais aprofundado das metodologias de projeto de cifras de bloco, utilizadas para nortear o desenvolvimento de uma cifra segura e adequada a plataformas com restrição de recursos. Esta análise será apresentada no Capítulo 4, que discute os principais conceitos envolvidos na construção de cifras desta família e mostra algumas das mais proeminentes técnicas de ataque utilizadas contra a mesma.

4 CIFRAS DE BLOCO E CRIPTOANÁLISE

No capítulo anterior, foram discutidas as características que definem uma cifra de bloco, bem como as técnicas envolvidas em sua utilização. Este capítulo complementa a discussão anterior ao apresentar algumas das principais estratégias de projeto e primitivas criptográficas adotadas, essenciais para a obtenção de algoritmos seguros. Ao longo deste capítulo, deve ficar clara a importância dos conceitos de difusão e confusão estabelecidos por Shannon (cf. Capítulo 2), que são a base do projeto de cifras de bloco.

É importante ressaltar, entretanto, que as estratégias mais modernas levam em conta não apenas estes princípios, mas também alguns dos mais efetivos ataques criptoanalíticos conhecidos. Portanto, para um completo entendimento de como estes fatores influenciam na construção de algoritmos seguros, neste capítulo são também descritas técnicas de ataque a cifras de bloco amplamente utilizadas com a intenção de transpor sua segurança. Apesar de alguns dos conceitos utilizados nestes ataques poderem ser corretamente aplicados à criptoanálise de cifras de fluxo, o foco do presente estudo é o desenvolvimento de cifras de bloco e, portanto, será dada uma maior ênfase a estas últimas.

4.1 Redes de Substituição-Permutação

Permutação e a substituição são primitivas básicas utilizadas no projeto de diversos algoritmos criptográficos. Quando se considera o projeto de cifras atuais, estas

operações podem ser melhor formalizadas da seguinte maneira:

- **Substituição:** um conjunto de n_s bits é substituído por outro, em geral (mas não necessariamente) do mesmo tamanho, de acordo com uma tabela que recebe o nome de **S-Box** (*Substitution-Box*, ou “Caixa de Substituição”). Para ser considerada segura, a relação entre os bits de entrada e saída desta S-Box deve ser altamente não-linear. Deste modo, o objetivo da operação de substituição é prover confusão à cifra.
- **Permutação:** conjuntos de n_p bits são embaralhados entre si, como palavras em um anagrama. Esta permutação de posições é responsável pela capacidade de difusão da cifra.

Em geral, as técnicas de substituição e permutação utilizadas sozinhas não podem ser consideradas seguras. Todavia, a intercalação destas duas primitivas de forma iterativa mostra-se bastante interessante, sendo a estratégia adotada pela família de cifras conhecidas genericamente como **SPN** (*Substitution-Permutation Networks*, ou “Redes de Substituição-Permutação”).

Com relação à chave criptográfica, existem dois modos comumente adotados para incorporá-la a uma cifra do tipo SPN:

- O primeiro deles consiste em intercalar camadas de substituição e permutação (**camadas SP**) independentes da chave e camadas de aplicação da sub-chave de round. São também incluídas camadas de adição de sub-chave no início e no final da cifra, pois sem este procedimento seria fácil para um atacante remover a camada SP não “protegida” por uma sub-chave. Um bom exemplo de cifra que adota esta abordagem é o AES. É interessante ressaltar que, ao invés de se utilizar uma chave a cada round, há ainda a possibilidade de aplicá-la apenas em algumas posições específicas da cifra (a cada 2 rounds, por exemplo).

- Uma segunda forma possível é a utilização da chave de round para alterar o funcionamento de uma das operações de substituição ou permutação, ou mesmo de ambas. Uma discussão sobre este tipo de abordagem é mostrada em (KELIHER; MEIJER; TAVARES, 1997). A chave pode ser utilizada, por exemplo, para definir quais S-Boxes serão usadas em cada round, selecionadas dentro de um conjunto de várias S-Boxes possíveis.

A diferença entre estas duas abordagens é ilustrada na Figura 11.

Figura 11: Dois modos de incorporar as chaves de round em uma SPN: as chaves geradas no escalonamento de chaves (KS) são aplicadas de forma independente das camadas SP (esquerda); elas podem também influenciar diretamente nas camadas S e/ou P , por exemplo por meio da seleção das S-Boxes utilizadas (direita). A última operação de permutação é comumente suprimida para tornar a cifra verticalmente simétrica (ignorando as camadas de adição de chave)

A decifração é realizada injetando os bits “ao inverso” na SPN, ou seja, a ordem dos rounds é invertida e em cada um deles é primeiro aplicada a permutação inversa, seguida da S-Box inversa. Portanto, a implementação do algoritmo de decifração exige que as operações de permutação e substituição inversas sejam também implementadas, o que pode chegar a dobrar o custo da cifra em termos de hardware ou número de li-

nhas de código. Para superar este problema, é comum que os projetos de cifras de bloco utilizem técnicas para torná-las *involutivas*, o que significa que as operações de encriptação e deciptação são idênticas, exceto pelo escalonamento de chaves. Esta classe de cifras, também denominadas *E/D similares* (LAI, 1992), têm como vantagem adicional a garantia de que sua segurança é igual em ambos os sentidos, o que previne ataques que exploram de alguma forma a assimetria de cifras. Uma estratégia bastante interessante para a construção de cifras involutivas é a adoção de funções de round compostas apenas por operações auto-inversas, bem como a remoção da operação de permutação na última camada da SPN (operação tracejada na Figura 11), deixando a cifra verticalmente simétrica. A literatura inclui alguns casos de cifras involutivas e análise das mesmas, como em (BARRETO; RIJMEN, 2000a; BARRETO; RIJMEN, 2000b; DAEMEN et al., 2000; YOUSSEF; TAVARES; HEYS, 1996; YOUSSEF; MISTER; TAVARES, 1997). Este também é o caso do próprio CURUPIRA, que será apresentado no Capítulo 7.

4.2 Cifras de Feistel

No início da década de 70, um pesquisador da IBM, Hors Feistel, descreveu uma forma de construir cifras de bloco involutivas, conhecida como *Estrutura de Feistel* (FEISTEL, 1973), que permite a implementação do algoritmo de deciptação mesmo que a cifra envolva operações não-inversíveis. O processo de encriptação com uma cifra desta família é ilustrado na Figura 12.

A estrutura proposta por Feistel consiste basicamente na divisão dos blocos de texto claro P em dois sub-blocos L e R , cada qual com metade do tamanho do bloco original. Cada sub-bloco R passa pela função de encriptação F , juntamente com uma chave de round K_i gerada pelo escalonamento de chaves (operação “KS” na Figura 12). A operação de substituição consiste em combinar o sub-bloco resultante com o sub-bloco L , em geral (mas não necessariamente) por meio de uma operação de XOR. Finalmente, os sub-blocos R e L são permutados. Este processo é repetido em todos os

Figura 12: Estrutura de Feistel

rounds da cifra, exceto no último, que não apresenta a permutação final. Pode-se então escrever:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

É fácil perceber que este esquema é inversível: denotando por σ_i e π_i , respectivamente, as operações de substituição e permutação do i -ésimo round, é possível escrever:

$$\sigma_i^2(L, R) = \sigma_i(L \oplus F(R, K_i), R) = (L \oplus F(R, K_i) \oplus F(R, K_i), R) = (L, R)$$

$$\pi_i^2(L, R) = \pi_i(R, L) = (L, R)$$

Isto significa que as operações σ_i e π_i são involuções, mesmo que a função de encriptação F não o seja. Para que a cifra seja involutiva, basta então alternar as operações de substituição e permutação e garantir a simetria vertical da estrutura completa (razão pela qual a permutação no último round é removida).

A operação de deciptação assume então a forma simples a seguir:

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus (F(R_{i-1}, K_i))$$

Como a operação de XOR é utilizada para combinar o sub-bloco R e a saída da função de encriptação F , as sub-chaves utilizadas na deciptação são exatamente as mesmas do processo de encriptação, porém aplicadas em ordem inversa. Isto pode não ser verdade para cifras que usam uma estrutura de Feistel modificada, nas quais uma operação diferente de XOR é utilizada para combinar estes conjuntos de bits. Isto é o que ocorre, por exemplo, como o TEA (WHEELER; NEEDHAM, 1995), que utiliza soma de inteiros módulo 32 ao invés do XOR e, assim, requer a implementação de uma função de deciptação em separado.

A segurança de uma cifra de Feistel depende essencialmente da segurança da função F e do número de rounds utilizado, que deve ser suficiente para garantir boa confusão e difusão. Existe ainda uma variante da estrutura descrita denominada **Estrutura de Feistel Desbalanceada**, que consiste em utilizar blocos L e R com tamanhos distintos. Análises mostram que esta construção pode realmente ter impacto na segurança da cifra, mas que sua influência pode ser positiva ou negativa (SCHNEIER; KELSEY, 1996). Como exemplo de cifra que adota uma estrutura desbalanceada, pode-se citar o caso do MacGuffin (BLAZE; SCHNEIER, 1995) que, apesar de baseada no DES, é ainda mais vulnerável a ataques diferenciais do que este último (RIJMEN; PRENEEL, 1994).

4.3 Criptoanálise Moderna

A Criptoanálise é a ciência complementar à Criptografia: enquanto esta visa à proteção das informações, aquela tem como objetivo a quebra desta proteção. Desta forma, sempre que um esquema criptográfico é utilizado, é de se esperar que ataques criptoanalíticos serão empregados com o objetivo de transpor sua segurança, permitindo que um atacante recupere textos claros a partir de seus correspondentes textos

cifrados. A ciência que conjuga Criptografia e Criptoanálise é chamada Criptologia.

Antigamente, a segurança de muitas cifras era principalmente baseada no segredo de sua estrutura detalhada, enquanto o conhecimento da chave de encriptação tomava um papel secundário. Este não é mais o caso atualmente: segundo o ***Princípio de Kerckhoff***, a segurança de uma cifra não deve se basear no segredo do projeto do algoritmo, mas deve depender apenas de se manter em segredo a chave criptográfica utilizada.

De uma forma geral, a criptoanálise de uma cifra pode assumir duas formas:

- **Parcial:** O atacante descobre o texto claro correspondente a um ou vários textos cifrados, mas não pode fazê-lo para todos os textos possíveis.
- **Total:** O atacante é capaz de decriptar todo e qualquer texto gerado pela cifra (por meio da recuperação da chave completa utilizada, por exemplo).

Além disto, dependendo da quantidade de informação da qual dispõe um atacante, distinguem-se algumas modalidades de ataques criptoanalíticos:

- **Ataque de texto cifrado puro:** O atacante tem acesso apenas a textos cifrados, ou partes deles, trocados entre pares comunicantes (espionando o tráfego na rede, por exemplo). Este é o tipo de ataque com menores restrições aqui apresentado, pois supõe que o atacante não necessita de nenhuma informação sobre os textos claros correspondentes para ter sucesso.
- **Ataque de texto claro conhecido:** O atacante dispõe de um ou vários pares de textos cifrados e claros correspondentes (ou parte deles). Em alguns casos, este tipo de ataque requer apenas o conhecimento de certas palavras-chave, contexto da mensagem ou mesmo de características léxicas ou sintáticas do idioma utilizado para ser efetivo.

- **Ataque de texto escolhido (claro ou cifrado):** Trata-se de um ataque ativo, em que o atacante é capaz de gerar certa quantidade de textos claros e obter seus correspondentes textos cifrados ou vice-versa. Isto pode ser feito induzindo um dos pares comunicantes a encriptar/decriptar estes textos, ou ainda por meio do acesso à própria cifra (mas não à chave sendo utilizada). Apesar de ser um tipo de ataque mais difícil de realizar na prática, ele fornece condições de avaliar até que ponto a segurança da cifra depende exclusivamente da chave e não de outras informações.
- **Ataque adaptativo de texto escolhido:** Neste caso o atacante não apenas é capaz de escolher quais textos claros ele deseja encriptar, como também pode fazê-lo de maneira interativa, ou seja, ele pode escolher quais textos ele deseja que sejam encriptados a partir do conhecimento dos pares de textos claros/cifrados gerados anteriormente.
- **Ataque de chave relacionada:** O atacante é capaz de observar a operação da cifra usando algumas chaves diferentes, de valores inicialmente desconhecidos, mas relacionadas matematicamente entre si de maneira conhecida. Ele pode saber, por exemplo, que os primeiros 64 bits da chave são sempre idênticos, apesar de não saber a princípio que bits são esses. Esta informação pode ser usada para selecionar textos claros que resultam, com certa probabilidade, em textos cifrados com uma relação também conhecida, permitindo encontrar as chaves corretas. Este modelo é mais factível quando se considera que atualmente grande parte dos processos criptográficos, incluindo a escolha da chave utilizada, é realizado de maneira autônoma. Portanto, caso não haja grande variabilidade no processo de seleção das chaves, esta forma de ataque mostra-se bastante realista.

O objetivo principal da criptoanálise moderna é a descoberta da chave criptográfica utilizada por um determinado algoritmo criptográfico, utilizando a menor quantidade possível de recursos como processamento, memória e número de mensagens

claras/cifradas. Apesar dos detalhes de um ataque criptoanalítico dependerem da cifra em questão, há algumas formas gerais que são amplamente utilizadas. Elas se baseiam na existência de características que distinguem os rounds da cifra de uma função completamente aleatória, permitindo que o resultado das mesmas seja até certo ponto previsível. Como o projeto da maioria dos algoritmos atuais leva em consideração os ataques conhecidos, a análise e a compreensão destas técnicas são de fundamental importância.

4.3.1 Ataque de Força Bruta

O ataque mais simples possível é a *Busca Exaustiva*, também conhecido como *Ataque de Força Bruta*. Trata-se de um ataque do tipo texto-claro conhecido, que consiste em encriptar o texto claro P com todas as chaves possíveis até que o texto cifrado C desejado seja encontrado. A chave K correta é aquela que satisfaz esta condição.

A complexidade de um ataque de força bruta aplicado a uma cifra com chaves de n_k bits é, no pior caso, da ordem de 2^{n_k} encriptações completas. Este é considerado o resultado base na avaliação de um ataque qualquer: para ser considerado eficiente, ele deve ter um desempenho consideravelmente superior àquele apresentado por um simples ataque de Busca Exaustiva.

Da mesma forma que este ataque é fácil de ser implementado, percebe-se que a proteção contra o mesmo é também simples de ser obtida: basta que a chave utilizada pela cifra seja suficientemente longa para que a Busca Exaustiva se torne computacionalmente inviável. Um tamanho comumente utilizado por cifras de bloco comerciais é 128 bits, apesar de cifras voltadas a redes de sensores normalmente apresentarem tamanhos menores.

4.3.2 Criptoanálise Linear

A descoberta da técnica conhecida como Criptoanálise Linear é normalmente atribuída a Mitsuru Matsui (MATSUI; YAMAGISHI, 1992), que inicialmente a utilizou na quebra do algoritmo FEAL (SHIMIZU; MIYAGUCHI, 1987) e mais tarde na criptoanálise do DES (MATSUI, 1993).

Este é um ataque do tipo texto claro conhecido e baseia-se na utilização de aproximações lineares para as porções não-lineares da cifra, normalmente restritas às S-Boxes. Em geral, a linearidade refere-se a operações de XOR, ou seja, se $X = [x_1, x_2 \dots x_{n_s}]$ representa a entrada da S-Box e $Y = [y_1, y_2 \dots y_{n_s}]$ refere-se à sua saída, são construídas relações entre os bits de X e de Y da forma:

$$(x_1 \oplus u_1) \oplus \dots \oplus (x_{n_s} \oplus u_{n_s}) \oplus (y_1 \oplus v_1) \oplus \dots \oplus (y_{n_s} \oplus v_{n_s}) = 0 \quad (4.1)$$

Onde são usados os vetores binários u e v para seleção dos bits que farão parte da equação linear. Para S-boxes de 4 bits, por exemplo, existem ao todo $2^{4+4} = 256$ equações possíveis. Há uma probabilidade associada a cada destas equações lineares: ela pode ser válida para um total de 0 a 2^{n_s} valores distintos de entrada (sendo inválida quando o resultado da combinação linear for igual a '1').

Suponha então que seja aplicada uma chave $K = [k_1, k_2 \dots k_{n_s}]$ aos bits $Z = [z_1, z_2 \dots z_{n_s}]$ anteriores à entrada da S-Box (i.e. $X = Z \oplus K$), de forma que a equação 4.1 seja reescrita como:

$$(z_1 \oplus u_1) \oplus \dots \oplus (z_{n_s} \oplus u_{n_s}) \oplus (y_1 \oplus v_1) \oplus \dots \oplus (y_{n_s} \oplus v_{n_s}) = \sum_{i=1}^{n_s} k_i \oplus u_i \quad (4.2)$$

Neste caso, o efeito da chave é apenas alternar entre si os valores de X , mas não a probabilidade associada à equação 4.1. Portanto, no que diz respeito à probabilidade p_L de que esta equação seja válida, a camada de adição de chave pode ser ignorada.

Suponha então a existência de uma cifra composta por uma única S-Box e duas

aplicações de chave, com a chave K^0 sendo aplicada a Z e a chave K^1 sendo aplicada a Y , dando como resultado W , conforme a Figura 13. Nesta situação, K^1 poderia ser encontrada da seguinte forma: efetua-se uma decifração parcial de diversos valores distintos de W cujos valores de Z são conhecidos, testando neste processo todos os K^1 possíveis; uma análise estatística dos valores de Y obtidos desta maneira irá revelar a chave K^1 , já que a mesma deve fazer com que a equação 4.1 seja válida com uma probabilidade igual a p_L (ou inválida com probabilidade igual a $1 - p_L$). É importante notar que, apesar de o atacante não ter acesso direto aos valores de X , os bits de Z podem ser aplicados diretamente na equação 4.1, pelas razões discutidas acima.

Figura 13: Cifra simplificada: uma S-Box e 2 camadas de aplicação de chave

A criptoanálise linear nada mais é que uma extensão deste exemplo básico: como a cifra envolve algumas S-Boxes e diversos rounds, é necessário encontrar expressões similares à 4.1 apresentando uma alta ou baixa probabilidade de ocorrência. A composição das S-Boxes aproximadas desta maneira dá origem a uma **trilha linear** que se estende por quase toda a cifra (apenas alguns poucos rounds não são cobertos). A probabilidade desta trilha depende do número de S-Boxes que a compõem, denominadas **S-Boxes Ativas**, e da probabilidade associada a estas últimas, conforme será descrito na seção 4.3.2.1. Ao invés de testar todas as chaves possíveis, apenas aquelas aplicadas à saída da trilha devem ser consideradas. Portanto, quanto menor o número de S-Boxes neste ponto, menor o número de chaves testado e menor a porção de chave recuperada.

O processo pode ser repetido diversas vezes, para trilhas cobrindo diferentes S-Boxes na saída, até que a chave completa seja determinada.

É importante ressaltar que quanto maior a probabilidade p_L de que a equação linear seja válida (ou inválida), maior a linearidade da S-Box para os bits que compõem tal equação. Em outras palavras, quanto mais distante de 50% for o valor de p_L encontrado, menor é a resistência da S-Box à criptoanálise linear. A medida deste desvio é denominada **desvio da probabilidade linear** d_L , o qual pode então ser calculado como $d_L = p_L - 1/2$, de forma que $-1/2 \leq d_L \leq 1/2$.

O ataque linear resume-se então em duas fases:

1. Construção de uma trilha linear de alta probabilidade estendendo-se por quase todos os rounds da cifra, seguida do cálculo de sua probabilidade associada. Quanto mais distante de 50% for a probabilidade associada a esta trilha (i.e. quanto maior o valor de seu desvio d_L), melhor pode ser considerada a aproximação e mais efetivo será o ataque.
2. Escolha de uma sub-chave candidata e decifração parcial dos textos cifrados disponíveis usando esta sub-chave. Uma análise estatística revelará a sub-chave correta: aquela que, com probabilidade correspondente à da trilha linear construída, leva ao resultado esperado na sua saída.

Para maiores detalhes e um exemplo bastante ilustrativo do funcionamento do ataque, recomenda-se a leitura de (HEYS, 2002, Seção 3).

4.3.2.1 O lema de *Piling-Up*

Sejam x_1 e x_2 duas variáveis binárias aleatórias. Pode-se escrever uma relação linear entre estas variáveis como $x_1 \oplus x_2 = 0$, o que equivale a dizer que $x_1 = x_2$; analogamente, a expressão $x_1 \oplus x_2 = 1$ equivale à expressão $x_1 \neq x_2$. Pode-se ainda

escrever a probabilidade associada a estas variáveis como:

$$Prob(x_i = a) = \begin{cases} p_a & \text{se } a = 0, \\ 1 - p_a & \text{se } a = 1 \end{cases}, i = 1, 2$$

Supondo x_1 e x_2 independentes, a probabilidade composta é dada simplesmente pela multiplicação das probabilidades simples. Logo, a probabilidade associada à expressão $x_1 \oplus x_2 = 0$ é dada por:

$$\begin{aligned} Prob(x_1 \oplus x_2 = 0) &= Prob(x_1 = x_2) \\ &= Prob(x_1 = 0 \text{ e } x_2 = 0) + Prob(x_1 = 1 \text{ e } x_2 = 1) \\ &= p_1 p_2 + (1 - p_1)(1 - p_2) \end{aligned}$$

Pode-se ainda escrever $p_1 = 1/2 + d_{L1}$ e $p_2 = 1/2 + d_{L2}$, o que resulta em:

$$Prob(x_1 \oplus x_2 = 0) = 1/2 + 2d_{L1}d_{L2}$$

Deste modo, o desvio associado à aproximação linear com duas variáveis é dado por $d_{L1,2} = 2d_{L1}d_{L2}$, resultado que pode ser generalizado para n variáveis pelo **Lema de Piling-Up**:

Lema de Piling-Up (MATSUI, 1993):

Sejam n variáveis binárias aleatórias e independentes $x_i (1 \leq i \leq n)$, que assumem os valores '0' e '1' com probabilidade p_i e $1 - p_i$, respectivamente. Tem-se que:

$$Prob(x_1 \oplus x_2 \oplus \cdots \oplus x_n = 0) = 1/2 + 2^{n-1} \prod_{i=1}^n d_{Li}$$

Ou, de maneira equivalente:

$$d_{L1,2,\dots,n} = 2^{n-1} \prod_{i=1}^n d_{Li}$$

Onde $d_{L1,2,\dots,n}$ representa o desvio de $x_1 \oplus x_2 \oplus \cdots \oplus x_n = 0$.

Por exemplo, considerando quatro variáveis binárias aleatórias independentes x_1, x_2, x_3 e x_4 , pode-se dizer que $Prob(x_1 \oplus x_2 = 0) = 1/2 + d_{L1,2}$ e $Prob(x_2 \oplus x_3 = 0) = 1/2 + d_{L2,3}$. É possível ainda escrever $Prob(x_1 \oplus x_3 = 0) = Prob([x_1 \oplus x_2] \oplus [x_2 \oplus x_3] = 0) = 1/2 + d_{L1,2}d_{L2,3}$ de acordo com o Lema de *Piling-up*, o que significa que $d_{L1,3} = d_{L1,2}d_{L2,3}$.

Na construção de uma aproximação linear para a cifra, os valores de x_i representam os bits na entrada ou saída das S-Boxes. As S-Boxes ativas são aquelas em cuja entrada há algum bit x_i e, portanto, devem ser aproximadas. O desvio de uma trilha linear é então diretamente proporcional ao desvios de suas S-Boxes ativas e inversamente proporcional a número das mesmas.

Apesar da grande utilidade do Lema de *Piling-Up* no cálculo da probabilidade associada a uma trilha linear, na prática é comum encontrar valores diferentes para os desvios teórico e experimental. A razão para esta discrepância reside em parte no fato das variáveis da aproximação linear não serem completamente independentes. Outro fator importante é a existência de **linear hulls** (NYBERG, 1995), que consistem em conjuntos de trilhas distintas que satisfazem uma mesma relação linear, sendo assim necessário somar as suas probabilidades para encontrar o valor real da probabilidade de tal aproximação. Estes dois fatores resultam, portanto, em um maior valor para o desvio real do que aquele calculado pelo Lema de *Piling-Up*.

4.3.2.2 Complexidade do Ataque

O número N_L de textos claros conhecidos necessários para o sucesso de um ataque é proporcional a $1/d_L^2$ (HEYS, 2002, Seção 3.6), podendo então ser calculado como:

$$N_L \approx c/d_L^2$$

Onde c é uma constante pequena. Esta equação diz simplesmente que o ataque exige um número de textos claros suficientes para que a aproximação construída seja

válida um número não-desprezível de vezes, permitindo então que a chave correta possa ser distinguida dentre as diversas chaves testadas. Logo, para uma cifra com tamanho de bloco n_b , é necessário que a trilha construída apresente um desvio significativamente maior do que $2^{-n_b/2}$ ou o número de textos claros necessários será maior do que o existente. É importante ressaltar, entretanto, que este é um resultado clássico. Existam atualmente análises mais precisas, que avaliam a chance de sucesso do ataque quando são removidas algumas das simplificações descritas, como o desvio linear nulo para chaves incorretas. Para tanto, estes trabalhos costumam basear-se fortemente em distribuições estatísticas para avaliar a probabilidade de sucesso do ataque, como em (SELctimescUK, 2008). De qualquer forma, o resultado clássico é normalmente aceito como uma boa aproximação para o cálculo do esforço envolvido em ataques lineares.

Técnicas de projeto de S-Boxes modernas costumam ter como requisito a baixa probabilidade de suas aproximações lineares. Além disto, busca-se aumentar o número de S-Boxes ativas para a cifra completa. Contudo, apesar destes fatores reduzirem as chances de sucesso de um ataque linear, é importante observar que a presença de *linear hulls* pode gerar resultados catastróficos para a segurança de uma cifra: mesmo que, individualmente, suas aproximações lineares apresentem um reduzido desvio, a combinação dos desvios das mesmas pode resultar em um valor acumulado suficientemente elevado para viabilizar ataques deste tipo.

4.3.3 Criptoanálise Diferencial

A Criptoanálise Diferencial é um ataque estatístico do tipo texto claro escolhido, sendo amplamente aplicável a algoritmos criptográficos iterativos. Ela foi apresentada por Eli Biham e Adi Shamir (BIHAM; SHAMIR, 1991).

A técnica baseia-se na escolha de pares de textos claros P^1 e P^2 apresentando uma diferença $\Delta P = P^1 \oplus P^2$ fixa. A encriptação destes textos claros resulta em textos

cifrados C^1 e C^2 , que irão apresentar uma diferença $\Delta C = C^1 \oplus C^2$ entre si. Assim, a Criptoanálise Diferencial explora a potencialmente alta probabilidade de que o processo de encriptação conserve uma relação entre ΔP e ΔC , que pode então ser analisada para recuperar informações sobre a chave utilizada pela cifra. Normalmente, a “diferença” refere-se à operação XOR (hipótese que será mantida ao longo da discussão). No entanto, é importante ressaltar que em algumas cifras ela pode ser definida de forma diferente para facilitar a análise, como ocorre em (MIRZA, 1998).

Ao par $(\Delta P, \Delta C)$ dá-se o nome *diferencial*. O ataque consiste na busca e utilização de um ΔP que leva a um valor de ΔC com uma alta probabilidade. Este par recebe o nome de *característica diferencial*. Analogamente ao que ocorre na criptoanálise linear (cf. seção 4.3.2) é necessário examinar as propriedades da S-Box utilizada para que seja possível distinguir uma diferencial interessante para o ataque: quanto maior o número de pares (X^1, X^2) que conservem uma relação $(\Delta X, \Delta Y)$ fixa, mais efetivo ele será, sendo 2^{n_s} o número máximo de pares para uma S-Box de n_s bits. Com $n_s = 4$, por exemplo, se aos 2^4 pares (X^1, X^2) que satisfazem $X^1 \oplus X^2 = 0001$ estão associados 2^3 pares (Y^1, Y^2) que satisfazem $Y^1 \oplus Y^2 = 0100$, a probabilidade do par $(0001, 0100)$ é de $p_D = 2^3/2^4$. Assim, supondo que a S-Box do exemplo de cifra mostrado na Figura 13 apresenta esta probabilidade, o ataque é perpetrado da seguinte maneira: diversos pares Z^1 e Z^2 apresentando a diferença $\Delta Z = 0001$ são encriptados com as chaves K^0 e K^1 ; efetua-se então a decriptação parcial dos pares W^1 e W^2 resultantes, usando todas as chaves K^1 possíveis; a chave K^1 correta é aquela que leva a uma diferença $\Delta Y = 0100$ com probabilidade da ordem de p_D . É importante notar que, no que diz respeito ao ataque, os pares (Z^1, Z^2) e (Z^2, Z^1) são exatamente iguais.

A extensão deste exemplo para uma cifra com diversos rounds e S-Boxes exige a construção de uma *trilha diferencial* de alta probabilidade, a qual apresenta a diferencial desejada e se estende do primeiro round da cifra até alguns poucos rounds anteriores ao último (em geral, não mais do que 2 ou 3). A decriptação parcial e aná-

lise estatística deve então ser realizada na saída desta trilha, considerando as S-Boxes ativas neste ponto. É importante notar que esta abordagem leva à supressão da influência das sub-chaves intermediárias, pois a operação de diferença utilizando o XOR faz com que elas se cancelem ao longo da trilha construída.

É recomendada a leitura de (HEYS, 2002, Seção 4) para maiores detalhes do ataque diferencial, além de um exemplo bastante ilustrativo de seu funcionamento.

4.3.3.1 Probabilidade de uma característica diferencial

Supondo que todas as S-Boxes (e, portanto, as diferenciais a elas associadas) são independentes entre si, calcula-se a probabilidade p_D de uma característica diferencial como:

$$p_D = \prod_{i=1}^{n_\alpha} \beta_i$$

Onde n_α indica o número de S-Boxes ativas que compõem a trilha diferencial (aquelas cujas entradas apresentam algum bit ‘1’ e, portanto, possuem uma probabilidade associada à sua saída) e β_i é a probabilidade de se encontrar a diferença desejada na i -ésima S-Box da trilha.

Na prática, é comum encontrar valores diferentes para o desvio teórico e o experimental. Um fator que influencia nesta discrepância é o fato de que a hipótese de independência entre as S-Boxes não é totalmente verdadeira, apesar de ser uma boa aproximação. Outro fator importante reside na existência de características diferenciais que podem ser atingidas por múltiplas trilhas diferenciais, i.e. existem trilhas de baixa probabilidade que também resultam na diferencial desejada (LAI; MASSEY; MURPHY, 1991), aumentando sua probabilidade.

4.3.3.2 Complexidade do ataque

Quanto menor o número de S-Boxes ativas e quanto maior a probabilidade das diferenciais nas mesmas, maior será a probabilidade associada à trilha diferencial construída e, portanto, menor a quantidade de textos claros necessários para construir o ataque. Apesar de ser difícil determinar com exatidão o número N_D de pares de textos claros necessários, uma regra geralmente empregada é:

$$N_D \approx c/p_D$$

Onde c é uma constante pequena. Esta regra diz simplesmente que são necessárias algumas ocorrências de pares seguindo a trilha diferencial construída, denominados *pares corretos*, para que seja possível distinguir a chave correta dentre as diversas chaves testadas. Como, para esta chave, a probabilidade de ocorrência de pares corretos deve ser próxima de $1/p_D$, a adoção deste valor é bastante razoável. Além disto, sendo n_b o tamanho de bloco da cifra, há ainda a exigência de que p_D seja significativamente maior do que 2^{1-n_b} para que o ataque seja possível: caso contrário, o número de pares de textos claros apresentando a diferença desejada será maior do que o máximo existente, 2^{n_b-1} (lembrando que, no ataque, os pares (P^1, P^2) e (P^2, P^1) são equivalentes). Novamente, este é um resultado clássico que, apesar de bastante empregado, mostra-se menos preciso que análises estatísticas mais rigorosas, como aquela apresentada em (SELctimescUK, 2008).

4.3.3.3 Outras modalidades de Ataque Diferencial

O *Ataque Diferencial Truncado* foi introduzido pelo criptógrafo Lars R. Knudsen, em (KNUDSEN, 1995). Knudsen mostra exemplos de cifras de Feistel que, apesar de consideradas seguras contra ataques diferenciais, são vulneráveis a ataques diferenciais truncados, evidenciando a importância de se considerar esta técnica na análise de segurança de uma cifra. Ainda no mesmo artigo, é também introduzido o conceito de

Diferenciais de Alta Ordem.

Enquanto Ataque Diferencial clássico utiliza diferenciais completas (cf. seção 4.3.3), Ataque Diferencial Truncado baseia-se no uso de diferenciais envolvendo apenas uma parte dos bits dos textos claros e cifrados. Mais precisamente, representando a diferencial completa por $(\Delta P, \Delta C)$, a diferencial truncada seria $(\Delta P', \Delta C')$, onde $\Delta P'$ é um subconjunto dos bits de ΔP e $\Delta C'$ é um subconjunto dos bits de ΔC . Portanto, o Ataque Diferencial Truncado distingue-se da versão completa em dois pontos principais: na construção da característica diferencial, que envolve menos bits; nos bits que compõem a chave parcial testada, que não serão necessariamente os bits de saída da última S-Box ativa. Assim, é possível reduzir o número de S-Boxes que compõem a trilha diferencial, o que resulta em trilhas cuja probabilidade associada é potencialmente maior do que a obtida com diferenciais completas.

Diferenciais de Alta Ordem, por outro lado, ao invés de considerar apenas a diferença entre pares de textos claros/cifrados, considera diferenças entre diferenças e assim por diante. Para algumas cifras, esta abordagem mostra-se mais vantajosa que o Ataque Diferencial clássico.

Outra modalidade desta mesma família envolve o conceito de ***Diferenciais Impossíveis*** (KNUDSEN, 1998), que exploram trilhas nas quais é impossível encontrar alguns valores para o estado da cifra.

4.3.4 Ataque de Interpolação

O ataque de interpolação é um ataque do tipo texto claro conhecido, descrito pela primeira vez em (JAKOBSEN; KNUDSEN, 1997). Ele explora S-Boxes que possam ser representadas por funções algébricas razoavelmente simples, i.e. S-Boxes com uma baixa não-linearidade. Em geral, procura-se substituir estes elementos por produtos de funções quadráticas (polinômios sobre um corpo finito) de baixa ordem. Fazendo esta aproximação para a cifra completa, ela passa a ser escrita na forma de operações

algébricas simples, permitindo representar os textos cifrados como uma função de suas mensagens claras correspondentes, de maneira independente da chave criptográfica utilizada. Esta função de interpolação é normalmente obtida por meio da **Fórmula de Interpolação de Lagrange**, que pode ser escrita como:

$$p(x) = \frac{f(x)}{(x - x_1)f'(x_1)}y_1 + \frac{f(x)}{(x - x_2)f'(x_2)}y_2 + \cdots + \frac{f(x)}{(x - x_n)f'(x_n)}y_n$$

onde

$$f(x) = (x - x_1)(x - x_2) \cdots (x - x_n)$$

Considerando uma cifra de bloco iterativa de tamanho de bloco n_b , cada texto cifrado pode ser escrito como um polinômio de interpolação multiplicado por seu texto claro correspondente. Sendo n_c o número de coeficientes deste polinômio, prova-se (JAKOBSEN; KNUDSEN, 1997, Seção 3.1, Teorema 2) que, se $n_c \leq 2^{n_b}$, é possível construir uma interpolação a partir de n_c textos claros conhecidos, encriptados com uma mesma chave secreta K , resultando em um algoritmo equivalente à encriptação/decriptação com K . O ataque envolve uma complexidade temporal da ordem de n_c .

Uma variante do ataque consiste na utilização de textos claros escolhidos pelo atacante, que pode fixar alguns dos bits destes textos visando à construção de polinômios com poucos coeficientes. Apesar do aumento na velocidade do ataque, tal abordagem apresenta uma desvantagem: o polinômio construído desta maneira só substitui adequadamente a cifra quando os textos claros apresentam os mesmos bits fixados pelo atacante. Além disto, construindo-se um polinômio que cobre a cifra completa exceto por 1 round, este ataque pode ser estendido para recuperar a chave usada pela cifra (JAKOBSEN; KNUDSEN, 1997, Seção 3.2, Teorema 3).

É importante ressaltar, entretanto, que a viabilidade desta modalidade está atrelada à baixa complexidade da S-Box utilizada pela cifra em questão. Portanto, a não-

linearidade no projeto de S-Boxes atuais assume mais uma vez um papel fundamental.

4.3.5 Ataque de Saturação

O *Ataque de Saturação*, também conhecido como *Ataque Integral* ou *Ataque Quadrado*, é um ataque do tipo texto claro escolhido que apareceu pela primeira vez no artigo descrevendo a cifra de bloco SQUARE (DAEMEN; KNUDSEN; RIJMEN, 1997), donde o nome “Ataque Quadrado” (“*Square Attack*”). Mais tarde, o ataque foi generalizado para outras cifras, como Rijndael (DAEMEN; RIJMEN, 2002, Seção 10.2) e Twofish (LUCKS, 2000). Ele é particularmente eficiente contra cifras baseadas em redes de substituição-permutação, figurando atualmente como um dos mais efetivos contra algoritmos que seguem a Estratégia de Trilha Larga (cf. Capítulo 5) com um reduzido número de rounds.

Diz-se que a posição de um bloco está *saturada* quando se tem à disposição uma quantidade de blocos suficiente para que esta posição assuma todos os valores possíveis. O ataque de saturação consiste basicamente na utilização de conjuntos de textos claros com uma ou mais posições saturadas que, após um reduzido número de rounds, se propagarão para os estados seguintes de forma a distingui-los facilmente de um estado aleatório. Um conjunto de textos claros com esta propriedade recebe o nome de *integral*.

O número exato de rounds passíveis de serem atacados depende bastante da capacidade de difusão da cifra: como o ataque explora a forma como a saturação se propaga, quanto maior o número de posições que mantêm uma relação de dependência entre dois rounds consecutivos, mais difícil de se prever o resultado da propagação e, portanto, maior o número de integrais necessárias e maior a quantidade de chaves que devem ser testadas no ataque. Em cifras como o AES, por exemplo, que atinge a difusão total em 2 rounds (i.e. após 2 rounds, todas as posições do bloco cifrado dependem de todas as posições do bloco de texto claro), o ataque se baseia em uma caracterís-

tica distintiva que se propaga por até 3 rounds, sendo o ataque mais básico aplicado a 4 rounds da cifra. O mesmo pode ainda ser estendido a 6 rounds adicionando-se um round no início e outro no final do ataque básico, aumentando consideravelmente, entretanto, o seu custo total.

Existe ainda uma melhoria provida pela técnica de programação dinâmica denominada ***Soma Parcial*** (FERGUSON et al., 2000). Usando este artifício, pode-se reduzir o esforço computacional ao custo de uma maior utilização de memória para armazenar resultados parciais do ataque.

O Apêndice B descreve o Ataque de Saturação aplicado ao CURUPIRA, o qual é introduzido no Capítulo 7.

4.3.6 Ataque Gilbert-Minier

O ataque Gilbert-Minier (GILBERT; MINIER, 2000) é um ataque do tipo texto claro escolhido e explora a existência de colisões entre algumas funções parciais da cifra. Esta modalidade de ataque foi inicialmente desenvolvida para o AES, sendo aplicável a até 7 rounds do mesmo. Ele pode ainda ser estendido para outras cifras da mesma família, utilizando conceitos semelhantes.

Similarmente ao Ataque de Saturação, o Ataque Gilbert-Minier se utiliza de blocos idênticos exceto por uma posição. Ele explora a dependência dos estados internos da cifra em relação aos bits do bloco inicial e das chaves de round utilizadas. A existência de posições que dependam de um pequeno número destes bits permite a construção de ***colisões*** internas, ou seja, é possível identificar estados que apresentam valores idênticos para alguns grupos de bits, mesmo quando seus textos claros correspondentes são distintos. Devido à previsibilidade deste comportamento, é possível distinguir estes estados de um conjunto completamente aleatório de dados. Assim, a decritação parcial dos textos cifrados usando um conjunto reduzido de chaves leva à identificação da chave correta, pois a mesma deve causar a colisão no estado interno esperado.

Assim como ocorre com o Ataque de Saturação, quanto maior a velocidade de difusão de uma cifra, mais ela é resistente ao Ataque Gilbert-Minier. Isto ocorre porque, entre dois rounds quaisquer, uma elevada difusão significa que cada posição de um estado depende de um grande número de posições do estado anterior, bem como das chaves de round utilizadas. Esta característica reduz a previsibilidade da ocorrência de colisões internas.

4.3.7 Ataque de Extensão Geral

Esta não é exatamente uma modalidade de ataque criptoanalítico, mas sim uma constatação de que qualquer ataque a R rounds de uma cifra pode ser estendido a $R + 1$ ou mais rounds, simplesmente testando todas as chaves K^{R+1} possíveis e efetuando um ataque a R rounds da mesma (LUCKS, 2000).

Obviamente, este método aumenta a complexidade do ataque, exigindo maior quantidade de processamento, memória, textos claros/cifrados, etc. Todavia, desde que a complexidade resultante ainda seja inferior à de um ataque de busca exaustiva, este procedimento pode se mostrar bastante interessante, viabilizando ataques a um maior número de rounds.

4.3.8 Ataques Algébricos

Ataques Algébricos compreendem uma classe de ataques cujo alvo principal são cifras com uma estrutura altamente matemática. Uma modalidade desta família conhecida como *Ataque XSL* (COURTOIS; PIEPRZYK, 2002) causou certo furor quando foi apresentada, posto que seus autores declararam a capacidade de quebrar o próprio AES por meio da técnica. Contudo, até o momento de redação deste documento, nenhum ataque foi demonstrado, nem mesmo a versões reduzidas do AES.

O ataque é do tipo texto claro conhecido. Resumidamente, ele consiste na análise da cifra em questão, seguida da construção de um sistema de equações quadráticas si-

multâneas que, resolvidas, permitem a recuperação da chave utilizada pela cifra. Em geral estes sistemas costumam ser bastante grandes, apresentando, por exemplo, 8000 equações com 1600 variáveis para o AES com chaves de 128 bits. A ataque XSL necessita de um algoritmo especializado para a resolução deste sistema de equações para que seja eficiente. O cálculo de sua complexidade depende, portanto, da capacidade de se construir tal algoritmo, dando ao ataque uma natureza bastante heurística. Apesar de terem sido desenvolvidas teorias que apóiam sua viabilidade (YANG; CHEN, 2004), a questão ainda permanece em aberto dado a falta de provas experimentais do sucesso da técnica.

4.3.9 Outros Ataques

O *Ataque Bumerangue* (WAGNER, 1999), um ataque do tipo texto claro escolhido de forma adaptativa, expande a capacidade do ataque diferencial. Ele consiste na divisão da trilha diferencial em duas sub-trilhas com metade do tamanho original, a primeira começando no primeiro round da cifra e a outra, no último. Este ataque é particularmente interessante no caso de cifras cuja resistência a ataques é diferente nos sentidos de encriptação e deciptação, já que uma das duas metades pode ser mais facilmente quebrada.

O *Slide Attack* (BIRYUKOV; WAGNER, 1999) é uma modalidade de criptoanálise que explora fraquezas existentes no processo de escalonamento de chaves da cifra alvo. Uma fraqueza comum é a existência de ciclos internos no escalonamento, ou seja, uma repetição periódica na seqüência de sub-chaves de round. Neste caso, esta técnica é capaz de dividir a cifra completa em porções reduzidas e atacá-las separadamente, fato que derrubou o preceito de que mesmo cifras pouco seguras poderiam se tornar bastante resistentes simplesmente pela adoção de um número elevado de rounds.

O ataque χ^2 (leia “chi-quadrado”) foi proposto pela primeira vez por Serge Vaudenay (VAUDENAY, 1996), tendo como objeto a criptoanálise estatística do DES. Como

o próprio nome indica, esta técnica baseia-se na aplicação do teste χ^2 (GREENWOOD; NIKULIN, 1996). Desta forma, a cifra atacada pode ser tratada como uma caixa preta sobre a qual são buscadas relações que produzam um desvio significativo do que se esperaria em uma permutação uniformemente distribuída. Este desvio, semelhante a uma característica linear ou diferencial, pode ser usado para recuperar a chave utilizada pela cifra. Para maiores detalhes sobre ataques estatísticos, recomenda-se a leitura da tese de doutorado de Pascal Junod (JUNOD, 2005).

Ataques de Temporização não exploram fraquezas na estrutura da cifra em si, mas sim em uma implementação com elementos dependentes da chave utilizada e mensuráveis. Por exemplo, uma implementação em software na qual algumas linhas de código são executadas condicionalmente, como uma vez para cada bit ‘1’ da chave, apresentará um tempo de execução dependente da chave utilizada; esta informação pode ser utilizada por um atacante na determinação das chaves mais provavelmente utilizadas pelo algoritmo.

Apesar de não ser exatamente um ataque, a existência de **chaves fracas** pode reduzir de maneira significativa a segurança de uma cifra. Recebem esta denominação as chaves criptográficas que, usadas em conjunto com uma determinada cifra, fazem com que a mesma apresente um comportamento inadequado. Um exemplo bastante conhecido é o caso das 4 chaves fracas do DES (todos os bits ‘0’, todos os bits ‘1’, a primeira metade dos bits ‘1’ e a segunda ‘0’ e vice-versa), que fazem com que cada dois rounds da cifra se anulem mutuamente. Normalmente, caso tais chaves existam para uma determinada cifra, elas ocupam um espaço pequeno no universo de chaves possíveis. Assim, apesar de ser difícil provar formalmente a sua inexistência para a maioria das cifras, o projeto de cifras de bloco atuais costuma incluir técnicas para reduzir sua probabilidade de ocorrência.

4.4 Resumo

Este capítulo discorreu sobre a forma como operações básicas como substituição e permutação são capazes de prover, respectivamente, confusão e difusão à cifra, dois conceitos de fundamental importância na construção de algoritmos criptográficos seguros. Foram então apresentadas duas famílias de cifras de bloco que se utilizam intensamente destas operações básicas, denominadas Redes de Substituição-Permutação (SPNs) e Cifras de Feistel. Estas estruturas são bastante conhecidas, existindo atualmente diversas análises sobre as mesmas e, principalmente, sobre a segurança das cifras que adotam estas formas.

Ainda, tendo em vista a sua importância no projeto de cifras atuais, algumas das mais proeminentes técnicas de criptoanálise conhecidas foram mostradas neste capítulo. Apesar de sua grande diversidade, elas evidenciam a importância das operações de substituição não-linear e permutação dos bits dentro de cada bloco. O entendimento destas técnicas, em especial dos Ataques Linear e Diferencial, serão fundamentais no Capítulo 5, que discute os detalhes envolvidos na Estratégia de Trilha Larga, desenvolvida especialmente para evitar estas famílias de ataques.

5 ESTRATÉGIA DE TRILHA LARGA

O capítulo anterior lançou as bases sobre as operações de Substituição e Permutação que formam as Redes de Substituição-Permutação, discutindo ainda a Criptoanálise Moderna. Este Capítulo discorre sobre a metodologia conhecida como Estratégia de Trilha Larga, que utiliza-se amplamente destes conhecimentos para prover segurança contra diversas formas de criptoanálise, em especial ataques lineares e diferenciais. Além dos conceitos matemáticos envolvidos, são também discutidas formas de otimizar as operações que compõem as cifras construídas segundo esta metodologia, bem como garantir um nível de segurança adequado (e mensurável) para as mesmas.

5.1 Introdução

O surgimento da Criptoanálise Diferencial e da Criptoanálise Linear, baseadas na construção de trilhas ao longo de vários rounds da cifra, levaram ao desenvolvimento de várias teorias sobre cifras de bloco. Para deixar esta idéia um pouco mais clara, a Figura 14 mostra uma trilha diferencial (linhas mais escuras) construída para uma SPN típica: as S-Boxes S_{11} a S_{44} operam sobre grupos de 4 bits e são intercaladas com camadas de difusão que espalham os bits de sua saída entre as S-Boxes seguintes.

A tabela diferencial, na porção esquerda da figura, é construída considerando todos os pares de entradas possíveis da S-Box: para cada par de entradas apresentando uma diferença ΔX , obtém-se a correspondente diferença ΔY na saída, incrementando o valor do par na tabela. Assim, é possível observar que, dos 16 pares possíveis apresentando

uma diferença $\Delta X = B$ na entrada da S-Box, 8 resultam em uma diferença na saída igual a $\Delta Y = 2$. Portanto, o par $(B, 2)$ tem uma probabilidade associada de $p_D = 50\%$. O mesmo procedimento é repetido por toda a cifra, dando origem a uma trilha que cobre as S-Boxes S_{12} , S_{23} , S_{32} e S_{33} (denominadas S-Boxes ativas). O ataque baseia-se então no uso de textos claros com esta diferença na entrada, tendo uma efetividade proporcional à probabilidade da trilha completa, conforme descrito na seção 4.3.3. O princípio envolvido no ataque linear é semelhante, mas este utiliza-se de relações entre alguns bits específicos na entrada e saída da S-Box (cf. seção 4.3.2).

Figura 14: Exemplo de Trilha Diferencial (linhas mais escuras) construída a partir de uma tabela diferencial, à esquerda. Esta tabela mostra o número de pares com diferença ΔX que resulta em pares com diferença ΔY na saída de cada S-Box (valores em hexadecimal). Observa-se que o valor na entrada da trilha (S-Box S_{12}) corresponde à máxima probabilidade na tabela ($p_D = 8/16 = 50\%$). Adaptado de (HEYS, 2002).

Atualmente, um importante requisito a ser satisfeito pelo projeto de cifras seguras é resistir a estas e outras modalidades de ataque, consideradas o estado da arte da criptoanálise. Preferencialmente, esta resistência deve ser demonstrável matematicamente de forma rigorosa. Um segundo requisito, este de ordem prática, refere-se ao bom

desempenho do algoritmo em termos de memória, processamento e outros recursos do sistema.

A Estratégia de Trilha Larga (ETL) (DAEMEN, 1995) é uma estratégia de projeto de cifras de bloco iterativas que combina de forma elegante eficiência e resistência a ataques diferenciais e lineares. Além disto, uma propriedade bastante interessante desta metodologia refere-se à possibilidade de efetuar análises quantitativas de sua robustez.

Cifras construídas segundo a ETL apresentam uma estrutura típica de SPN (cf. seção 4.1). Deste modo, elas são compostas por dois tipos diferentes de transformações inversíveis, cada qual com funcionalidade e requisitos próprios:

- γ : uma camada de substituição não-linear, na qual cada bit na saída depende de um número limitado de bits na entrada e grupos de bits vizinhos na saída dependem de grupos de bits próximos entre si na entrada.
- λ : uma camada linear de elevado poder de difusão.

Basicamente, as funções de round ρ são construídas a partir da conjunção destas duas primitivas, i.e. $\rho = \lambda \circ \gamma$. Para cada aplicação de ρ , é intercalada uma operação de adição de sub-chave, consistindo simplesmente no XOR entre os dados de saída do round e sua sub-chave correspondente, conforme ilustrado na Figura 15 (repare que esta figura é equivalente à SPN mostrada na porção esquerda da Figura 11, na seção 4.1). Este tipo de estrutura costuma apresentar uma boa resistência à criptoanálise, em especial a ataques lineares e diferenciais, conforme explicitado mais adiante.

A transformação γ é normalmente realizada por uma mesma S-Box, que atua separadamente nos grupos de n_s bits que compõem um estado. Posto que o uso de S-Boxes distintas para grupos distintos não leva necessariamente a um maior nível de segurança contra ataques (POSCHMANN et al., 2006), esta abordagem é preferível pois resulta em

Figura 15: SPN no contexto da Estratégia de Trilha Larga, evidenciando as operações γ e λ

uma implementação mais leve para a camada de substituição, tanto em software quanto em hardware.

A transformação λ , por sua vez, combina linearmente grupos de bits, de forma que cada grupo na saída do round seja uma função linear de alguns dos grupos da sua entrada. Uma forma simples e bastante adotada de implementar esta operação é por meio de uma multiplicação por matriz: os estados são organizados em uma estrutura matricial e então multiplicados à esquerda (resp. direita) pela matriz definida pela transformação λ , de forma que cada elemento de uma coluna (resp. linha) na matriz resultante dependam de todos os elementos de sua coluna (resp. linha) na matriz original. As propriedades desejáveis para esta transformação também serão apresentadas mais adiante.

5.2 Segurança contra Ataques Lineares

Além dos conceitos de probabilidade de uma aproximação linear e desvio de uma aproximação, uma forma útil de medir a qualidade de uma equação linear associada a uma S-Box é por meio da **correlação** entre esta aproximação e a S-Box original. Se f

e g são duas funções booleanas, calcula-se a sua correlação como $C(f, g) = 2 \cdot \text{Prob}(x \mid f(x) = g(x)) - 1$, de forma que $C(f, g) = C(g, f)$. Outro conceito importante é a **paridade** ℓ_u de um vetor booleano, uma função booleana que consiste no XOR de um número arbitrário de seus bits, selecionados de acordo com um **vetor de seleção** u . Para vetores de n bits, existem então 2^n paridades possíveis.

$$\ell_u(x) = \bigoplus_{i=0}^{n-1} u_i \cdot x_i.$$

Para que um ataque linear seja possível contra uma cifra de tamanho de bloco n_b , é necessário construir uma trilha linear apresentando um desvio significativamente maior do que $2^{-n_b/2}$, ou seja, o número de textos claros necessários no ataque deve ser bem menor do que o máximo existente, 2^{n_b} . Usando o conceito de correlação, pode-se ainda dizer que o sucesso do ataque depende da existência de uma aproximação para a cifra cuja correlação seja significativamente superior a $2^{-n_b/2}$. Para evitar esta modalidade de criptoanálise, portanto, escolhe-se um número de rounds tal que não existam trilhas cuja correlação associada seja maior do que $2^{-n_b/2}$.

É importante notar que a ETL não garante a inexistência de correlações com alta probabilidade para a cifra completa, mas apenas para trilhas individuais. De fato, de acordo com a igualdade de Parseval, para uma paridade de saída qualquer, a soma dos quadrados de sua correlação com todas as paridades possíveis na entrada deve ser igual a 1. Supondo que cada paridade de saída esteja uniformemente distribuída entre todas as 2^{n_b} paridades de entrada possíveis, a correlação para cada uma destas paridades de entrada terá uma amplitude de $2^{-n_b/2}$. Na prática, é muito pouco provável que esta distribuição uniforme seja atingida e, portanto, existirão correlações cuja magnitude seja maior do que $2^{-n_b/2}$. Este resultado também se aplica ao mapeamento efetuado em uma cifra usando uma determinada chave de encriptação. Assim, a existência de valores elevados de correlação em alguma trilha é um resultado matemático que não pode ser contornado por projeto.

Por outro lado, é importante ressaltar que, na ausência de correlações locais com valores elevados, uma alta correlação só pode ser concebida pela existência de *linear hulls*, conforme discutido na seção 4.3.2.1. Mais precisamente, uma correlação acima de $N \cdot 2^{-n_b/2}$ ($N > 1$) deve resultar da soma das correlações de, pelo menos, N trilhas lineares distintas. Como este conjunto de trilhas é altamente dependente da chave utilizada, encontrar este conjunto é inviável na prática. Tal afirmação não consiste em uma prova definitiva de segurança da ETL, mas demonstra sua grande capacidade de reduzir a viabilidade desta modalidade de ataque.

5.3 Segurança contra Ataques Diferenciais

Com a ETL, a segurança contra técnicas de criptoanálise diferencial é obtida ao garantir uma baixa probabilidade associada a trilhas diferenciais. Para que esta modalidade de ataque seja possível contra uma cifra de tamanho de bloco n_b , é necessário encontrar uma trilha com uma probabilidade significativamente maior do que 2^{1-n_b} , ou seja, o número de pares de textos claros necessários deve ser bem menor do que o máximo existente para uma diferença qualquer, $2^{n_b}/2$. Logo, para evitar o ataque, escolhe-se um número de rounds tal que não existam trilhas cuja probabilidade seja maior do que 2^{1-n_b} .

É importante ressaltar que a ETL não garante que todas as características diferenciais tenham probabilidade abaixo de 2^{1-n_b} , mas apenas que isso seja válido para trilhas individuais. Conforme discutido na seção 4.3.3.2, múltiplas trilhas de baixa probabilidade podem resultar em uma mesma característica diferencial, aumentando sua probabilidade. De fato, em razão das propriedades de qualquer tabela de diferenciais, uma S-Box apresenta necessariamente diferenciais com probabilidade igual (ou maior do que) 2^{1-n_b} , um resultado matemático que não pode ser contornado.

Dizer que uma diferencial possui probabilidade p_D implica na existência de $p_D \cdot$

2^{1-n_b} pares de textos claros que a satisfazem, enquanto a trilha seguida por cada um destes pares depende diretamente da chave utilizada. Assumindo que a distribuição destes pares entre as trilhas possíveis respeite uma Distribuição de Poisson, o número esperado de pares que seguem uma trilha diferencial de probabilidade associada 2^{-p_t} é $2^{n_b-1-p_t}$. Considerando que a probabilidade desta trilha seja baixa, de forma que $2^{-p_t} < 2^{1-n_b}$, e que ela seja seguida por pelo menos um par, a probabilidade de que ela seja seguida por mais de um par é aproximadamente $2^{n_b-1-p_t}$. Portanto, se não houver trilhas diferenciais com uma probabilidade associada maior do que 2^{1-n_b} , os $p_D \cdot 2^{1-n_b}$ pares que formam a característica diferencial em questão seguem praticamente $p_D \cdot 2^{1-n_b}$ trilhas distintas.

Portanto, se a cifra apresentar apenas trilhas diferenciais com uma baixa probabilidade associada, a existência de diferenciais com uma alta probabilidade é possível apenas quando múltiplas trilhas se combinam para formá-la. Para cada diferente chave utilizada, entretanto, cada uma destas trilhas individuais pode ou não ser seguida por um par. Esta imprevisibilidade faz com que a construção de uma trilha diferencial de alta probabilidade seja praticamente irrealizável, mesmo quando se conhece a chave (e ainda mais quando não se tem acesso à mesma). Apesar desta propriedade também não consistir em uma prova definitiva da segurança da ETL contra ataques diferenciais, ela garante uma redução considerável da viabilidade dos mesmos.

5.4 Peso de uma trilha

Em um ataque linear, a correlação de uma trilha pode ser calculada como o produto das correlações associadas às diferentes aproximações das S-Boxes ativas. Define-se o *peso* W de uma correlação C como o negativo do logaritmo de sua amplitude: $W(C) = -\log_2(C)$. Por conseguinte, a peso de uma trilha linear é dado pela soma dos pesos das S-Boxes que a compõem. Com esta definição, é possível calcular um limite inferior para o peso de qualquer trilha linear: ele é sempre maior ou igual ao número

de S-Boxes ativas multiplicado pelo menor peso (de correlação) destas S-Boxes.

Analogamente, o peso de uma diferencial com probabilidade p_D é definido como o negativo do logaritmo desta probabilidade: $W(p_D) = -\log_2(p_D)$. O peso de uma trilha diferencial é então calculado pela soma dos pesos das diferenciais de cada S-Box ativa e, portanto, seu limite inferior é dado pelo número de S-Boxes ativas multiplicado pelo menor peso (diferencial) destas S-Boxes.

Por definição, quanto maior o peso de uma trilha, menor é a probabilidade associada à mesma (e, portanto, menores as possibilidades de se construir um ataque a partir desta trilha). É por esta razão que, para aumentar a segurança de uma cifra contra ataques lineares e diferenciais, a ETL busca maximizar o limite inferior do peso de suas trilhas.

5.4.1 O conceito de “Trilhas Largas”

O raciocínio anterior sugere duas formas possíveis para eliminar trilhas com um baixo peso:

- Projetar S-Boxes cujos pesos diferencial e de correlação tenham um limite inferior elevado (reduzindo, assim, os valores em suas tabelas linear e diferencial);
- Projetar funções de round que maximizam o número de S-Boxes ativas em qualquer trilha.

A amplitude máxima de uma correlação para uma S-Box inversível de n_s bits é ao menos de $2^{n_s/2}$. Isto leva a um limite superior para o valor mínimo do peso de uma correlação: $\min(W(C)) \leq n_s/2$. Por outro lado, a máxima probabilidade de uma característica diferencial é ao menos 2^{n_s-2} , resultando em um limite superior para o valor mínimo assumido pelo peso diferencial: $\min(W(p_D)) \leq n_s - 2$. O uso de S-Boxes grandes (i.e. com n_s grande) seria então uma solução possível para aumentar o peso

das trilhas. No entanto, esta abordagem é bastante custosa em termos computacionais e não é a solução adotada pela ETL.

A ETL foca-se no projeto de funções de round que maximizam o número de S-Boxes ativas em qualquer trilha (diferencial ou linear). Isto faz com que toda e qualquer trilha que possa ser construída apresente um perfil “alargado”, pois cobre muitas S-Boxes. Assim, contrastando com a trilha mostrada na Figura 14 mostrada anteriormente, uma trilha “larga” seria mais semelhante àquela mostrada na Figura 16, que cobre diversas S-Boxes em todo o seu caminho. Para que tal propriedade seja satisfeita, as cifras que seguem esta metodologia investem uma boa parte dos recursos computacionais na camada de transformação linear, com o intuito de maximizar a capacidade de difusão da cifra completa.

Figura 16: Trilha com perfil “alargado”, cobrindo um número grande de S-Boxes. A ETL garante que todas as trilhas individuais apresentem tal característica. Adaptado de (HEYS, 2002)

O próprio conceito de difusão assume, neste contexto, uma forma bem particular. Inevitavelmente, o mapeamento provido por uma S-Box causa de certa forma uma interação entre seus bits de entrada e saída, o que pode ser considerado uma forma de difusão. Entretanto, não é neste tipo de difusão restrita às S-Boxes que a ETL se interessa, já que a mesma não é capaz de aumentar o peso das trilhas lineares ou diferenciais. Para cifras construídas seguindo a ETL, a difusão é proporcionada somente pela operação linear λ , sendo medida por meio da introdução de um novo conceito: o *fator de ramificação*.

5.5 Fator de ramificação

Considere uma função de partição α que divide os blocos de um estado em n_α grupos de bits. O peso de grupo $w_\alpha(s)$ deste estado com relação a esta partição é definido como o número de grupos que apresentam pelo menos um bit diferente de ‘0’ e estes grupos são denominados *grupos ativos*. Esta definição se aplica tanto a estados de uma trilha linear quanto àqueles de uma trilha diferencial.

O *fator de ramificação* $\mathcal{B}$ de uma transformação ϕ em relação a uma função de partição α se define como:

$$\mathcal{B}(\phi) = \min_{x \neq 0} (w_\alpha(x) + w_\alpha(\phi(x))).$$

Ou seja, o fator de ramificação é determinado pelo valor mínimo da soma dos pesos dos grupos antes e após a transformação ϕ . Em consequência, o fator de ramificação possui um limite superior que é atingido quando existe apenas 1 grupo ativo antes da transformação ϕ e exatamente n_α grupos ativos após a aplicação de ϕ , i.e. $\max(\mathcal{B}(\phi)) = 1 + n_\alpha$.

No caso de α corresponder à função de partição usada pela cifra (i.e. caso os grupos criados sejam os mesmos grupos sobre os quais operam as S-Boxes, sendo

então $n_\alpha = n_s$), o número total de S-Boxes ativas é idêntico ao número de grupos ativos em um estado. Como este caso facilita a discussão, permitindo relacionar o peso de uma trilha e o fator de ramificação, ele será adotado ao longo desta discussão a menos que o contrário seja explicitado.

5.5.1 Propriedades

Algumas propriedades importantes para a sequência da discussão são dadas a seguir:

- Por definição, os fatores de ramificação de uma transformação e de sua inversa são idênticos;
- O peso de uma trilha (diferencial ou linear) não é afetado por operações de adição de chave, independentemente da partição α . Este resultado é condizente com as seções 4.3.3 e 4.3.2, que mostram como o efeito da chave é anulado pela operação de subtração (ataque diferencial) ou pela seleção de bits (ataque linear);
- Transformações que operam sobre os grupos definidos por uma partição α e não são capazes de tornar um grupo inativo em um grupo ativo ou vice-versa (uma função de permutação, por exemplo), não alteram o número total de grupos ativos em um bloco e, portanto, não afetam o peso da trilha.

5.5.2 Difusão associada a 2 rounds de uma cifra

As propriedades do fator de ramificação levam a resultados interessantes para uma função de round da forma $\rho = \gamma \circ \lambda$. Como a transformação γ não afeta o número de S-Boxes ativas em uma trilha (diferencial ou linear), o fator de ramificação de ρ não é afetado por esta transformação, mas apenas por λ . Por conseguinte, para trilhas construídas com funções de round desta forma, tem-se:

Trilhas de 1 round: Para trilhas (diferenciais ou lineares) de um único round, o limite inferior para o número de S-Boxes ativas é 1. Isto ocorre porque o menor peso de grupo possível na entrada das S-boxes é 1, independentemente da transformação λ utilizada.

Trilhas de 2 rounds: Para trilhas (diferenciais ou lineares) de 2 rounds, o efeito de difusão de λ assume um papel importante. Conforme o *Teorema da Propagação em 2 Rounds*(cf. (DAEMEN; RIJMEN, 2001, seção 4.3, Teorema 1)), prova-se que o limite inferior do número de S-Boxes ativas para uma trilha de dois rounds é dado pelo fator de ramificação de λ .

5.6 Combinação de segurança e eficiência

Graças ao Teorema da Propagação em 2 Rounds, é possível concluir que trilhas apresentando $2R$ rounds podem ser decompostas em R trilhas de 2 rounds, cujo peso pode ser maximizado usando uma camada de difusão λ com alto fator de ramificação. Todavia, transformações com esta propriedade tendem a ter um alto custo de implementação associado. Uma solução mais eficiente consiste na utilização de duas funções de round diferentes, definidas como $\rho^a = \theta \circ \gamma$ e $\rho^b = \Theta \circ \gamma$, onde a transformação γ é a mesma descrita anteriormente, atuando em grupos de n_s bits. Assim, θ e Θ podem atuar em conjuntos menores de bits (sendo, portanto, operações menos custosas computacionalmente) e, em conjunto, garantir um elevado fator de ramificação, conforme explicado a seguir.

5.6.1 A transformação de difusão θ

A transformação θ atua sobre estados de n_b bits, organizados em uma matriz na qual cada grupo de n_s bits ocupa uma posição distinta. A operação θ opera sobre cada coluna separadamente, combinando linearmente seus elementos. Como esta transfor-

mação precisa garantir apenas a difusão no interior de cada coluna de forma isolada, seu custo de implementação é reduzido.

Analogamente à definição de grupos ativos, pode-se então definir o conceito de *colunas ativas*. Assim, considerando uma função de partição Ξ que divide o estado em n_Ξ colunas, cada qual com n_ξ elementos, obtém-se o seguinte limite superior para o fator de ramificação da transformação θ :

$$\mathcal{B}(\theta) \leq \min_{\xi} (n_\xi + 1).$$

Ou seja, o tamanho da menor coluna impõe um limite superior para o fator de ramificação. Além disto, estendendo-se o Teorema da Propagação de 2 Rounds, o peso de uma trilha de 2 rounds com o primeiro round da forma $\rho^a = \theta \circ \gamma$ tem $\mathcal{N} \cdot \mathcal{B}(\theta)$ como limite inferior, onde $\mathcal{N}$ é o número de colunas ativas na entrada do segundo round.

5.6.2 A transformação linear Θ

A transformação Θ tem como objetivo embaralhar grupos de colunas diferentes, provendo assim difusão entre elas. Como requisito de projeto, Θ deve apresentar um elevado fator de ramificação em relação à função de partição Ξ . Este *fator de ramificação de coluna* é denotado $\mathcal{B}(\Theta, \Xi)$.

5.6.3 Construção eficiente de Θ

Ao contrário de θ , Θ não opera em colunas diferentes de forma independente, o que pode levar a custos de implementação elevados. Contudo, existe uma construção bastante eficiente para tal transformação, dada por $\Theta = \pi \circ \theta \circ \pi$, onde π é uma transformação que efetua a permutação de grupos pertencentes a colunas distintas. Conforme discutido na seção 5.5.1, π não afeta o número de grupos ativos em uma trilha.

Como π realiza a difusão entre colunas, é intuitivo considerar que uma boa difusão requer que os grupos distintos em uma coluna sejam distribuídos para o maior número possível de colunas diferentes. Define-se ***difusão colunar ótima*** quando todos os grupos distintos em uma coluna são distribuídos entre todas as colunas. Claramente, para que isto seja possível, o número de colunas n_{Ξ} deve ser suficiente para acomodar todos os n_{ξ} grupos, ou seja, é necessário que $n_{\Xi} \geq \max(n_{\xi})$. Neste caso, demonstra-se que um limite inferior de $\mathcal{B}(\Theta, \Xi)$ é dado por $\mathcal{B}(\theta)$ (DAEMEN; RIJMEN, 2001, seção 5.4, Lema 2). Este resultado fundamental pode ser melhor compreendido quando se considera que todos os grupos ativos em uma coluna após a aplicação de θ serão distribuídos entre as diferentes colunas após a aplicação de Θ .

5.6.4 Construção eficiente da função de round

Considerando que Θ assume a forma apresentada na seção 5.6.3, pode-se fazer a seguinte construção para 4 rounds da cifra:

$$\begin{aligned} R^4 &= \rho^a \circ \rho^b \circ \rho^a \circ \rho^b \\ &= (\theta \circ \gamma) \circ (\pi \circ \theta \circ \pi \circ \gamma) \circ (\theta \circ \gamma) \circ (\pi \circ \theta \circ \pi \circ \gamma) \end{aligned}$$

Como λ atua em grupos de forma independente e π apenas reorganiza estes grupos, as transformações são intercambiáveis, ou seja, a sua ordem de aplicação não altera o resultado. Desta forma, pode-se escrever:

$$\begin{aligned} R^4 &= (\theta \circ \pi \circ \gamma) \circ (\theta \circ \pi \circ \gamma) \circ (\theta \circ \pi \circ \gamma) \circ (\theta \circ \pi \circ \gamma) \\ &= \rho^c \circ \rho^c \circ \rho^c \circ \rho^c \end{aligned}$$

A combinação do fator de ramificação de θ e o fator de ramificação de coluna de Θ permite determinar um limite inferior para o peso de uma trilha construída sobre 4 rounds da cifra. Quando estes 4 rounds apresentam a forma $\rho^a \circ \rho^b \circ \rho^a \circ \rho^b$, tal limite é dado por $\mathcal{B}(\theta) \cdot \mathcal{B}(\Theta, \Xi)$ (cf. (DAEMEN; RIJMEN, 2001, seção 5.3, Teorema 2)). Estendendo este resultado para a construção com $\rho^c = \theta \circ \pi \circ \gamma$, obtém-se um resultado

de fundamental importância para a Estratégia de Trilha Larga:

Teorema da Propagação em 4 Rounds: para uma cifra de bloco iterativa apresentando uma função de round da forma $\rho = \theta \circ \pi \circ \gamma$, o limite inferior para o número de S-Boxes ativas para qualquer trilha de 4 rounds é dado por $(\mathcal{B}(\theta))^2$.

Conclui-se, portanto, que uma camada de difusão com elevado fator de ramificação, aliada à estrutura proposta pela ETL e aplicada em um número suficiente de rounds constitui uma abordagem altamente eficaz contra ataques diferenciais e lineares. Uma forma bastante interessante de construir tal camada de difusão é apresentada a seguir.

5.7 Códigos MDS

Esta seção apresenta algumas definições importantes com relação à teoria dos códigos lineares. Para uma discussão mais profunda sobre o assunto, recomenda-se a leitura de (MACWILLIAMS; SLOANE, 1977).

A **Distância de Hamming** entre dois vetores u e v do espaço de vetores n -dimensionais $\text{GF}(2^p)^n$ é o número de coordenadas para as quais u e v diferem. O **Peso de Hamming** $w_h(a)$ de um elemento $a \in \text{GF}(2^p)^n$ é a Distância de Hamming entre a e o vetor nulo de $\text{GF}(2^p)^n$, i.e. o número de componentes não-nulos de a . Desta forma, o Peso de Hamming é um caso particular do peso de grupo, quando a função de partição α divide o bloco em grupos de apenas 1 bit.

Um **código linear** $[n, k, d]$ em $\text{GF}(2^p)$ é um sub-espço k -dimensional do espaço de vetores $\text{GF}(2^p)^n$ que respeitam o limite superior $d \leq n - k + 1$, ou seja, a Distância de Hamming entre dois vetores distintos quaisquer deste sub-espço é de pelo menos d , sendo d o maior número para o qual esta propriedade é válida. Para $d = k/2$, por exemplo, garante-se que dois vetores quaisquer deste código apresentam no mínimo a metade de seus componentes diferentes. Desta forma, quanto maior o valor de d ,

maior será a diferença mínima entre dois vetores quaisquer. Um código linear para o qual $d = n - k + 1$ é denominado **Código MDS** (*Maximal Distance Separable* ou “Separável pela Máxima Distância”).

Uma **matriz geratriz** G de um código linear $[n, k, d]$ C é uma matriz $k \times n$ cujas linhas formam uma base de C . Uma matriz geratriz está na forma escalonada ou padrão se a mesma for da forma $G = [I_{k \times k} A_{k \times (n-k)}]$, onde $I_{k \times k}$ é a matriz identidade de ordem k . Em outras palavras, a matriz geratriz na forma escalonada é aquela que se obtém pela concatenação da matriz identidade $I_{k \times k}$ com uma matriz $A_{k \times (n-k)}$. Será utilizada a notação $G = [I A]$, omitindo os índices das matrizes sempre que suas dimensões forem irrelevantes para a discussão ou claras a partir do contexto. A Figura 17 ilustra a forma como são construídos estes códigos.

Figura 17: Esquema ilustrativo de códigos MDS, evidenciando a matriz geratriz

Um código linear $[n, k, d]$ C com matriz geratriz $G = [I_{k \times k} A_{k \times (n-k)}]$ é MDS se, e somente se, toda sub-matriz quadrada formada por linhas e colunas de A é não-singular, ou seja, possuem determinante não-nulo (MACWILLIAMS; SLOANE, 1977, capítulo 11, § 4, teorema 8). A matriz A é então denominada uma matriz MDS.

Seja θ a função definida como a multiplicação por uma matriz MDS A de dimensões $n_\xi \times n_\xi$. Seja ainda um código linear cujos vetores $2n_\xi$ -dimensionais são formados pela concatenação das colunas de tamanho n_ξ na entrada e na saída de θ . Este código é então um Código MDS $[2n_\xi, n_\xi, d]$ e, portanto, respeita a relação $d = n_\xi + 1$. Desta

forma, pode-se garantir que a alteração de uma única posição do vetor de entrada de θ leva à alteração de n_ξ posições na sua saída, ou seja, *todas* as posições na saída são alteradas. Esta propriedade das matrizes MDS é extremamente interessante no contexto da ETL: o uso desta classe de matrizes na camada de difusão θ garante que o fator de ramificação desta operação atinja seu limite superior: $\mathcal{B} = d$. Ainda, de acordo com o Teorema da Propagação em 4 Rounds enunciado na seção 5.6.4, garante-se um elevado número de S-Boxes ativas (d^2) para 4 rounds de uma cifra construída segundo a ETL.

5.8 Resumo

Este capítulo apresentou os conceitos envolvidos na metodologia de projeto de cifras de bloco conhecida como Estratégia de Trilha Larga (ETL). Desenvolvida especialmente contra ataques diferenciais e lineares, a ETL reduz a probabilidade de qualquer trilha individual (seja ela linear ou diferencial) que possa ser construída, inviabilizando a aplicação destas técnicas. Para isto, ela organiza os bits de cada bloco em matrizes e aplica iterativamente três diferentes operações: substituição altamente não-linear (γ), difusão inter-colunar (π) e difusão intra-colunar (θ). Esta última operação costuma ser implementada por meio de multiplicação por matrizes MDS, com o intuito de maximizar seu poder de difusão. É desta forma que esta metodologia permite a construção de algoritmos com segurança mensurável, sendo a base para diversas cifras consideradas seguras atualmente, como o próprio AES.

Além de um elevado nível de segurança, as operações envolvidas na ETL costumam ser bastante flexíveis, permitindo construções eficientes nas mais diversas plataformas. O uso de cifras com esta flexibilidade é especialmente interessante no contexto de redes de sensores, já que as mesmas podem agrupar, além de dispositivos com recursos restritos, servidores com elevada disponibilidade de memória e capacidade de processamento. Assim, garante-se que cada implementação possa aproveitar da melhor forma possível os recursos da plataforma à qual se destina.

Portanto, para compreender melhor a forma como uma cifra desenvolvida conforme a ETL pode ser inserida no contexto de redes de sensores, é importante estudar a forma como o TinySec, protocolo da camada de enlace desenvolvido para prover segurança a estas redes, foi desenvolvido. Este é o assunto abordado no Capítulo 6.

6 O PROTOCOLO TINYSEC PARA REDES DE SENSORES

O capítulo anterior discutiu os detalhes da Estratégia de Trilha Larga, metodologia que permite a construção de cifras de bloco consideradas bastante seguras mesmo frente às mais avançadas técnicas de criptoanálise conhecidas atualmente. Entretanto, considerando os objetivos deste trabalho, a discussão apresentada seria de interesse puramente teórico caso não houvesse uma boa oportunidade de aplicação da mesma no contexto de redes de sensores. É por esta razão que este capítulo descreve o protocolo da camada de enlace batizado de TinySec, atualmente bastante popular e amplamente utilizado quando é necessário prover segurança a esta classe de redes.

O TinySec foi desenvolvido especialmente para cenários apresentando restrições comuns a redes móveis e de sensores, tendo o cuidado de evitar vulnerabilidades encontradas em outros protocolos como o IEEE 802.11b (BOLAND; MOUSAVI, 2004) e o GSM (SIDDIQUE; AMIR, 2006), também voltados a redes sem fio. Aliado a um algoritmo criptográfico adequado, portanto, o Tinysec é capaz de prover controle de acesso, integridade e confidencialidade, em troca de um pequeno aumento de processamento, utilização de banda e consumo de energia. Conforme será explicitado durante este capítulo, entretanto, o algoritmo padrão adotado pelo protocolo apresenta uma baixa margem de segurança, sendo portanto recomendável a utilização de uma cifra mais adequada para garantir sua robustez.

É importante ressaltar que este capítulo cobre apenas as características principais do protocolo TinySec, bem como suas diretrizes de projeto. Para maiores detalhes sobre

o mesmo, recomenda-se a leitura de (KARLOF; SASTRY; WAGNER, 2004).

6.1 Projeto do Protocolo

O TinySec foi desenvolvido para utilização conjunta com a plataforma TinyOS (LEVIS et al., 2004), localizando-se na camada de enlace de forma transparente para aplicações que se utilizem do protocolo. Desta forma, provê-se grande facilidade de uso e integração com aplicativos.

As características do TinySec garantem que as necessidades de segurança mais básicas relacionadas a redes de sensores sejam cobertas. Desta forma, o foco é dado a requisitos de autenticidade, integridade e confidencialidade das mensagens, não sendo cobertos ataques como “exaustão de recursos” (em que um adversário envia diversas mensagens a um mesmo nó, de forma a exaurir sua fonte de energia), “sequestro de nós” (pelo qual um oponente recupera o material criptográfico de um nó físico e utiliza esta informação para se passar por um nó legítimo na rede) ou “repetição de pacotes” (que consistem no reenvio de mensagens antigas trocadas entre nós, as quais serão aceitas por se tratar de mensagens legítimas). A proteção contra tais ataques é relegada à camada de aplicação. Além de tirar proveito de serviços presentes nas camadas superiores, o protocolo usa também algumas das restrições presentes em redes de sensores a seu favor, como a limitação de mensagens que um adversário poderia recuperar ou injetar na rede em razão da reduzida largura de banda disponível.

Esta simplicidade apresentada pelo TinySec tem como objetivo não apenas garantir uma maior facilidade de implementação, mas principalmente fornecer um serviço de reduzido impacto no desempenho da rede. Tal característica de projeto é altamente desejável considerando-se as restrições dos dispositivos alvo, sendo este um dos principais motivos para a ampla aceitação do protocolo. De acordo com seus autores, entretanto, ele é recomendado para a maioria das aplicações nas quais a segurança seja

um fator importante, porém não crítico. Quando segurança se torna um fator nevrálgico do sistema, protocolos mais robustos e, portanto, pesados seriam mais adequados.

6.2 Modos de operação do protocolo e Estrutura de pacotes

De acordo com as necessidades de cada rede em que é aplicado, o protocolo TinySec pode ser usado em dois modos de operação: o modo *TinySec-Auth* provê integridade e autenticidade às mensagens pela utilização de *MACs* (*Message Authentication Codes*, ou “Códigos de Autenticação de Mensagens”); já o modo *TinySec-AE* se baseia tanto na utilização de MACs quanto de um algoritmo criptográfico para prover confidencialidade às mensagens. Em ambos os casos, o MAC é calculado sobre todo o pacote (dados + cabeçalho), com a diferença que no modo TinySec-AE os dados são previamente encriptados. Qualquer mecanismo de distribuição de chaves pode ser adotado, sendo então possível tanto a utilização de uma única chave para toda a rede quanto apenas entre pares de nós comunicantes ou mesmo entre grupos de nós.

Assim, os pacotes do protocolo TinySec podem assumir uma das formas mostradas na Figura 18.

Figura 18: Formato dos pacotes TinySec e TinyOS

É instrutivo analisar as diferenças existentes entre os pacotes TinySec-AE e TinySec-Auth:

- Ambos possuem campos em comum: *Dest* indica o destinatário do pacote; *AM* (*Active Message type*, ou “Tipo de Mensagem Ativa”) serve para indicar o protocolo da camada superior que deve tratar o pacote; *Len* refere-se ao tamanho do mesmo; *Data* e *MAC* referem-se aos dados e ao campo autenticação do pacote, respectivamente.
- A diferença de tamanho entre pacotes dos dois tipos resume-se a dois campos extras (4 bytes) no caso do TinySec-AE: *Src*, indicando a origem do pacote e *Ctr*, um contador usado para compor o vetor de inicialização *IV*, conforme será discutido na seção 6.3.1.
- O campo *Data* no pacote TinySec-AE encontra-se encriptado, enquanto o campo correspondente no pacote TinySec-Auth é enviado às claras.
- O campo *IV* existe apenas no modo TinySec-AE, já que o mesmo é usado pelo algoritmo criptográfico.
- Devido às diferenças de tamanho dos pacotes e operações envolvidas na geração dos mesmos, fica claro que a adoção do modo TinySec-AE implica em maior consumo de banda, energia e processamento do que a utilização do modo TinySec-Auth.

6.3 Criptografia

Duas decisões são de vital importância no projeto de um esquema de encriptação semanticamente seguro: o formato adotado pelo *IV* e a escolha do algoritmo criptográfico.

6.3.1 Formato do campo IV

O formato do *IV* usado pelo protocolo TinySec é singular: apesar de ser composto de 8 bytes, 4 deles correspondem na verdade a campos existentes no cabeçalho mesmo quando o modo TinySec-Auth é utilizado, a saber: endereço de destino (2 bytes), *AM* (1 byte) e tamanho do pacote (1 byte). Os outros 4 bytes garantem que cada nó seja capaz de enviar até 2^{16} mensagens antes de reutilizar um valor de *IV*, posto que 2 destes bytes correspondem ao seu endereço e os outros 2 correspondem a um contador iniciando em '0'.

A decisão quanto ao tamanho deste campo é o resultado de um compromisso entre desempenho e segurança: quanto maior o número de bits dedicados ao campo *IV*, maior o número de valores ele pode assumir e, portanto, maior a segurança do protocolo. Todavia, campos maiores implicam em pacotes maiores trocados entre nós, aumentando então a latência, consumo de banda e de energia durante a comunicação. Assim, para a formação do *IV*, apenas 4 bytes são adicionados ao pacote cifrado, enquanto os outros 4 bytes são “reaproveitados” com o intuito de aumentar sua variabilidade. De qualquer forma, em razão do reduzido tamanho deste campo, é consideravelmente grande a possibilidade de repetição de *IVs* ao longo da vida útil de um nó.

6.3.2 Algoritmo criptográfico

É importante ressaltar que o protocolo TinySec foi desenvolvido de maneira a aceitar diferentes algoritmos criptográficos. Contudo, restrições intrínsecas às redes de sensores, bem como características do próprio protocolo, fazem com que algumas cifras sejam mais recomendadas do que outras.

Conforme discutido na seção 3.1.3, apesar de cifras de fluxo serem potencialmente mais rápido do que cifras de bloco, elas se mostram altamente vulneráveis a ataques

no caso de reutilização de *IVs*. Portanto, em um contexto no qual existe uma grande possibilidade de sua repetição, como é o caso do TinySec, a utilização deste grupo de cifras não é recomendável.

Portanto, o uso de uma cifra de bloco em um modo de operação adequado apresenta-se como a melhor opção. Dentre os modos de operação possíveis (cf. seção 3.2.1), os idealizadores do TinySec optaram pelo CBC com pré-criptação do *IV*, devido ao seu bom comportamento com relação à repetição deste campo. É também usada a técnica de *Ciphertext Stealing* descrita na seção 3.2.2, garantindo assim que o tamanho dos blocos cifrados se mantenha igual ao de seus blocos claros correspondentes.

O AES e o TDEA (NIST, 2004) foram cogitados como cifras padrão, mas foram rejeitados: o primeiro, por adotar um tamanho de bloco (128 bits) muito grande e o segundo pelo baixo desempenho apresentado durante os testes com o protocolo. Assim, a escolha dos autores do TinySec recaiu sobre o Skipjack, ficando o RC5 (RIVEST, 1995) como uma alternativa.

6.4 Críticas

Conforme apresentado neste capítulo, o protocolo TinySec se mostra muito atraente para prover serviços básicos de segurança na comunicação entre nós de uma rede de sensores. Desta forma, é possível obter confidencialidade (TinySec-AE), integridade e autenticidade (TinySec-AE e TinySec-Auth). Balanceando desempenho e segurança, o protocolo apresenta boa aceitação em ambientes apresentando restrições de processamento, memória, energia e capacidade de banda.

Com relação à escolha do algoritmo criptográfico, entretanto, cabe uma crítica ao protocolo: o Skipjack é uma cifra de bloco que não apenas usa chaves criptográficas consideravelmente pequenas (80 bits), mas também apresenta uma margem de

segurança reduzida, posto que 31 de seus 32 passos podem ser criptoanalizados com sucesso (BIHAM; BIRYUKOV; SHAMIR, 1999). Já o RC5, apesar de apresentar um desempenho superior ao do Skipjack caso as sub-chaves de round sejam pré-computadas, tem sua utilização dificultada pela presença de patentes. Por outro lado, tanto o AES quanto o TDEA, apesar de mais seguros, não são adequados por questões de desempenho. Já uma cifra mais leve como o BKSQ (DAEMEN; RIJMEN, 1998) poderia ser adotada com algumas vantagens, já que o mesmo foi desenvolvido tendo *smart cards* em mente. No entanto, esta última não apresenta um comportamento involutivo, fazendo com que a encriptação e a decrptação devam ser implementadas separadamente e, deste modo, elevando a quantidade de memória necessária para sua operação. Esta desvantagem poderia ser compensada adotando-se um modo de operação que utiliza o processo de encriptação em uma única direção, como o modo CTR. No entanto, este é um modo de operação em fluxo, levando a desvantagens intrínsecas a esta categoria de cifras, como uma profunda dependência da não reutilização de vetores de inicialização (IVs). Como este não é o cenário no qual atua o TinySec, o BKSQ também foi rejeitado. Já o AES, apesar de permitir uma implementação eficiente e prover um bom nível de segurança, ainda carrega o problema do tamanho de bloco razoavelmente grande quando se considera o tamanho das mensagens trafegadas. Mesmo que o impacto sobre a transmissão seja evitado graças ao uso de *Ciphertext Stealing*, o impacto sobre o processamento total persiste quando o tamanho dos pacotes não é igual ou pouco inferior a um múltiplo de 16 bytes. Além disto, a decrptação com esta cifra exige o pré-cálculo de chaves, de forma que esta não seja uma opção ótima neste tipo de plataforma.

Portanto, apesar de seu projeto bem planejado, a segurança provida pelo TinySec fica ameaçada caso o mesmo não seja usado em conjunto com uma cifra ao mesmo tempo mais resistente a ataques e apresentado um desempenho semelhante ao do Skipjack na plataforma alvo. É com esta preocupação em mente que foi desenvolvida uma

solução leve, segura e de alto desempenho: o CURUPIRA, descrito em detalhes no Capítulo 7.

7 O CURUPIRA

Conforme discutido no capítulo anterior, um problema maior com o protocolo TinySec é sua escolha de algoritmo criptográfico padrão, o Skipjack, uma cifra de bloco apresentando uma baixa margem de segurança. Desta forma, existe uma interessante oportunidade de melhoria do protocolo pela utilização de uma cifra mais robusta. No entanto, como a escolha do Skipjack foi primordialmente motivada em razão de seu desempenho em plataformas com recursos limitados e ausência de patentes, qualquer cifra que venha a substituí-lo deve também contemplar estas características. É a estes requisitos que o algoritmo apresentado neste capítulo, o CURUPIRA, se propõe a satisfazer.

7.1 Características Gerais

O CURUPIRA é uma cifra de bloco especialmente projetada para plataformas apresentando restrições como reduzida disponibilidade de memória, energia e capacidade de processamento, como é o caso de redes móveis e redes de sensores, ou então sistemas fortemente dependentes de *tokens* e *smart cards*. Ele opera com blocos de 96 bits e aceita chaves de 96, 144 ou 192 bits, com um número variado de rounds. A cifra segue a Estratégia de Trilha Larga descrita no Capítulo 5, assim como o próprio AES. O CURUPIRA permite ainda grande flexibilidade na sua implementação, sendo possível utilizar adequadamente os recursos disponíveis na plataforma alvo e buscar um bom equilíbrio entre velocidade de processamento e memória necessária. Além disto, tendo

em vista as restrições intrínsecas ao ambiente para o qual a cifra foi desenvolvida, ela apresenta tanto uma estrutura involutiva quanto duas possibilidades de escalonamento de chaves, ambas facilmente inversíveis (mas apenas uma delas demonstrando comportamento cíclico).

Diversas cifras atuais apresentam estrutura involutiva. Apesar de todas as Cifras de Feistel clássicas e outras cifras mais gerais como Skipjack e MISTY1 (MATSUI, 1997) possuírem tal característica, este não é o caso do AES. Conforme discutido na seção 4.1, a importância de uma estrutura involutiva reside na equivalência de segurança apresentada tanto pelo algoritmo de encriptação quanto pelo de deciptação. Além disto, esta propriedade é bastante interessante em termos de compactação de código, um requisito de grande importância no caso do TinySec em razão de sua escolha pelo modo de operação CBC, que exige a implementação de ambos os algoritmos de encriptação e deciptação (cf. seção 3.2.1.2).

A maneira usual de fazer com que uma cifra projetada segundo a Estratégia de Trilha Larga apresente uma estrutura involutiva consiste no uso de operações de transposição de matrizes, exigindo que os blocos sejam arranjados em matrizes quadradas, ou então pela adoção de multiplicações por uma grande matriz unitária (i.e. uma matriz M satisfazendo $M \cdot M^{-1} = I$), o que se reflete em um maior tempo de processamento. A solução adotada no CURUPIRA é bem mais leve, (veja seção 7.3.1.2), evitando estas desvantagens. De certa forma, a cifra apresentada é semelhante ao BKSQ (DAEMEN; RIJMEN, 1998), outra cifra que segue a Estratégia de Trilha Larga e foi projetada para utilização em *smart cards*, mas que não apresenta uma estrutura involutiva.

O escalonamento de chaves de uma cifra de bloco é dito cíclico ou periódico se (1) ele aplica iterativamente uma determinada função sobre a chave inicial para gerar as sub-chaves de cada round e (2) a função em questão torna-se a identidade após um certo número de iterações. É importante ressaltar que, neste caso, as sub-chaves não precisam ser computadas de maneira incremental: elas podem ser geradas na ordem

inversa (uma necessidade comum durante o processo de decriptação), em uma ordem arbitrária ou até mesmo em paralelo. Esta última característica é particularmente interessante caso a cifra seja implementada em um hardware dedicado como, por exemplo, para ser usada em um servidor. Além disto, cada chave pode ser gerada exatamente no momento de sua utilização, ao contrário do que ocorre com algumas cifras nas quais todas as sub-chaves precisam ser pré-computadas e armazenadas em uma grande tabela antes que comecem a ser utilizadas. Este documento inclui duas propostas para o algoritmo de escalonamento, sendo que uma delas satisfaz esta propriedade e dá origem à cifra denominada CURUPIRA-1. Já a segunda versão deste algoritmo, utilizada pela cifra denominada CURUPIRA-2, não apresenta tal comportamento, mas tem como vantagem a sua velocidade superior. Apesar de distintas, estas duas versões serão indistintamente referenciadas como CURUPIRA ao longo deste documento, exceto quando sua diferenciação for necessária para a discussão.

Como costuma ser *de rigueur* em qualquer proposta de um novo algoritmo criptográfico, a discussão a seguir dá foco especial à sua análise de segurança, sem perder de vista os aspectos de implementação.

7.2 Preliminares Matemáticas e Notação

A seguir são apresentados alguns conceitos matemáticos importantes para a compreensão das operações envolvidas no CURUPIRA.

7.2.1 Corpos Finitos

Os elementos de um corpo finito (LIDL; NIEDERREITER, 1997) podem ser representados de diversas formas diferentes. Para qualquer potência de um número primo existe um único corpo finito, o que garante a que todas as representações de $GF(2^8)$ sejam isomórficas. Apesar desta equivalência, a representação utilizada tem influência

na complexidade de sua implementação. Neste trabalho foi escolhida a representação polinomial clássica: um byte u , formado pelos bits $[u_7u_6u_5u_4u_3u_2u_1u_0]$, é interpretado como um polinômio cujos coeficientes pertencem ao conjunto $0, 1$, ou seja, $u = u_7x^7 + u_6x^6 + u_5x^5 + u_4x^4 + u_3x^3 + u_2x^2 + u_1x + u_0$.

Exemplo: o byte representado pelo valor hexadecimal 57 (valor em binário: 01010111) corresponde, em $GF(2^8)$, ao polinômio $x^6 + x^4 + x^2 + x + 1$.

7.2.1.1 Adição

A soma de dois polinômios quaisquer é dada pela soma módulo 2 de seus coeficientes.

Exemplo: $57 + 83 = D4$ ou, em notação polinomial:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

Esta operação é mais facilmente observável quando se utiliza a notação binária: $01010111 + 10000011 = 11010100$. Assim, é fácil perceber que a adição em $GF(2^8)$ nada mais é do que a operação de XOR (representada como $\oplus$) byte a byte. Todas as condições necessárias são então satisfeitas para que seja definido um grupo Abelianiano para a operação de adição: propriedades comutativa e associativa, elemento neutro (00), elemento inverso (todo elemento é seu próprio inverso aditivo) e fechamento (a soma de dois elementos do grupo ainda pertence ao grupo). Como todo elemento é seu próprio inverso aditivo, as operações de adição e subtração se confundem, sendo ambas iguais a um simples XOR.

7.2.1.2 Multiplicação

Considerando a representação polinomial, a multiplicação em $GF(2^8)$ corresponde à multiplicação dos polinômios módulo um polinômio binário irredutível de grau 8. Um polinômio irredutível é aquele que não possui divisores, exceto por 1 e por si

mesmo (algo semelhante a um número primo quando se considera o conjunto dos números inteiros). No CURUPIRA, o polinômio escolhido é dado por $p(x) = x^8 + x^6 + x^3 + x^2 + 1$ (14D em notação hexadecimal). Este polinômio $p(x)$ foi escolhido por ser a primitiva pentanomial de grau 8 em $\text{GF}(2^8)$ para a qual a raiz cúbica da unidade $c(x)$ assume a forma mais simples possível, a saber: $c(x) = x^{85} \bmod p(x) = x^4 + x^3 + x^2$ (ou 1C em hexadecimal). Como multiplicações pelo polinômio gerador $g(x) = x$ de $\text{GF}(2^8)$ e também por $c(x)$ são freqüentes no algoritmo proposto, é fundamental que $c(x)$ apresente a forma mais simples possível por razões de eficiência.

O exemplo a seguir ilustra o modo como são realizadas as multiplicações no CURUPIRA:

Exemplo: $57 \cdot 83 = 43$ ou, em notação polinomial:

$$\begin{aligned}
 & (x^6 + x^4 + x^2 + x + 1) \cdot (x^7 + x + 1) \\
 &= x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1 \\
 &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \\
 &= (x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \bmod (x^8 + x^6 + x^3 + x^2 + 1) \\
 &= x^6 + x + 1
 \end{aligned}$$

Obviamente, o resultado de qualquer multiplicação será um polinômio de grau menor do que 8, incorrendo na propriedade do fechamento. Além disto, a multiplicação assim definida é associativa e apresenta o elemento neutro ($\mathbf{01}$), apesar de não existir uma operação simples no nível de byte como ocorre com a adição. Para qualquer polinômio binário $b(x)$ de grau abaixo de 8, o Algoritmo de Euclides estendido pode ser usado para calcular os polinômios $g(x)$ e $h(x)$ de forma que $b(x) \cdot g(x) + m(x) \cdot h(x) = 1$, o que é equivalente a dizer que $(b(x) \cdot g(x)) \bmod m(x) = 1$ ou que $b^{-1}(x) = g(x) \bmod m(x)$, i.e. existe o elemento inverso. Finalmente, pode-se escrever $g(x) \cdot (b(x) + h(x)) = g(x) \cdot b(x) + g(x) \cdot h(x)$ (associatividade). Como estas 5 propriedades são verificadas, o conjunto de todos os 256 valores possíveis dos bytes,

considerando o XOR como operação de adição e a multiplicação definida acima, tem a estrutura de um Corpo Finito $\text{GF}(2^8)$. O uso de Corpos Finitos em Criptografia é interessante, entre outras coisas, por razões de eficiência e pela garantia da inversibilidade das operações.

7.2.2 Matrizes em $\text{GF}(2^m)$

O conjunto de todas as matrizes $3 \times n$ em $\text{GF}(2^m)$ é denotado $\mathbb{M}_n$, enquanto O e I representam as matrizes nula e identidade, respectivamente.

Existem 3 matrizes que são de fundamental importância no CURUPIRA:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Por inspeção direta, mostra-se que A e B são nilpotentes ($A^2 = B^2 = O$), enquanto a matriz C é idempotente ($C^2 = C$) em $\text{GF}(2^m)$. Pode-se ainda observar que $AB = BA = O$.

Assim, as matrizes da forma $D = I + \alpha A + \beta B$ são involuções, ou seja, $D^2 = I$, $\forall \alpha, \beta \in \text{GF}(2^m)$. Por enumeração, é fácil de perceber que os determinantes de todas as sub-matrizes quadradas formadas pelas linhas e colunas de D apresentam uma das formas $\{\alpha, \alpha + 1, \beta, \beta + 1, \alpha + \beta, \alpha + \beta + 1\}$, implicando que D é MDS se, e somente se, $\{\alpha \neq 0, 1; \beta \neq 0, 1; \beta \neq \alpha, \alpha + 1\}$. Os menores coeficientes possíveis que satisfazem estas restrições são $\alpha = 2, \beta = 4$. Portanto, a forma mais simples que pode ser assumida pela matriz D para que ela seja MDS é $D = I + 2A + 4B$.

Além disto, um cálculo simples mostra que a matriz da forma $E = I + cC$ é inversível se, e somente se, $c \neq 1$, resultando em $E^{-1} = I + \left(\frac{c}{c+1}\right)C$. Os determinantes de todas as sub-matrizes quadradas formadas pelas linhas e colunas de E assumem uma das formas $\{1, c, c + 1\}$, de modo que E é MDS se, e somente se, $\{c \neq 0, 1\}$. Definindo

c como a raiz cúbica da unidade, tem-se que $c^3 = 1$ e também que $c^2 + c + 1 = 0$ (tendo em vista que $c^3 + 1 = (c + 1)(c^2 + c + 1)$ e $c \neq 1$). Assim, pode-se escrever:

$$c^3 = 1 \Rightarrow c^2 = 1/c$$

$$c^3 + 1 = 0 \Rightarrow (c + 1)(c^2 + c + 1) = 0 \Rightarrow c^2 + c + 1 = 0 \text{ (já que, por projeto, } c \neq 1)$$

$$c^2 = c + 1 \Rightarrow 1/(c + 1) = 1/c^2 = c$$

$$\underline{\underline{c/(c + 1) = c^2 = c + 1}}$$

O que leva a uma forma particularmente simples para E^{-1} , a saber $E^{-1} = I + (c + 1)C$.

As matrizes D e E assim definidas assumem um papel fundamental CURUPIRA, conforme será explicitado nas seções 7.3.1.3 e 7.3.2.3. O Apêndice A dá alguns detalhes adicionais sobre o modo como estas matrizes foram obtidas e os requisitos de projeto que elas satisfazem.

7.2.3 Propriedades Criptográficas

Algumas das definições a seguir já foram de certa forma exploradas ao longo do documento, sendo aqui formalizadas por razões de clareza.

Um produto de m variáveis Booleanas distintas é denominado um **produto de ordem- m** destas variáveis. Toda função Booleana $f : \text{GF}(2)^n \rightarrow \text{GF}(2)$ pode ser escrita como uma soma (em $\text{GF}(2)$) de diferentes produtos de ordem- m de seus argumentos, com $0 \leq m \leq n$; esta forma de representação é denominada **forma algébrica normal** de f . A **ordem não-linear** de f , denotada $\nu(f)$, corresponde à máxima ordem dos termos aparecendo em sua forma algébrica normal.

Uma função Booleana **linear** é uma função Booleana de ordem não-linear 1, i.e. sua forma algébrica normal apresenta apenas argumentos isolados. Dado um vetor de seleção $u \in \text{GF}(2)^n$, denota-se por $\ell_u(x) : \text{GF}(2)^n \rightarrow \text{GF}(2)$ a **paridade**, i.e. a função Booleana linear consistindo na soma (XOR) de n bits de x selecionados de acordo com

o vetor u :

$$\ell_u(x) = \bigoplus_{i=0}^{n-1} u_i \cdot x_i.$$

Um mapeamento $S : \text{GF}(2^n) \rightarrow \text{GF}(2^n), x \mapsto S[x]$, é denominado ***substitution box***, ou simplesmente S-box. Uma S-box pode também ser vista como um mapeamento $S : \text{GF}(2)^n \rightarrow \text{GF}(2)^n$ e, portanto, descrita em função de suas funções Booleanas constituintes $s_i : \text{GF}(2)^n \rightarrow \text{GF}(2), 0 \leq i \leq n-1$, ou seja $S[x] = (s_0(x), \dots, s_{n-1}(x))$.

A ***ordem não-linear*** de uma S-box S , denotada ν_S , é a ordem não-linear mínima sobre todas as combinações lineares dos componentes de S . Assim, ela mede a dificuldade de se representar a função implementada pela S-box como uma função linear de seus argumentos, sendo dada por:

$$\nu_S = \min_{\alpha \in \text{GF}(2)^n} \{\nu(\ell_\alpha \circ S)\}.$$

A ***tabela diferencial*** de uma S-box S , construída a partir da relação entre as diferenças na entrada e na saída da S-box, é definida como:

$$e_S(a, b) = \#\{c \in \text{GF}(2^n) \mid S[c \oplus a] \oplus S[c] = b\}.$$

O parâmetro δ de uma S-box S , que evidencia a máxima probabilidade de uma diferencial na tabela e_S , é definido como:

$$\delta_S = 2^{-n} \cdot \max_{a \neq 0, b} e_S(a, b).$$

O produto $\delta_S \cdot 2^n$ é denominado ***uniformidade diferencial*** de S , e mede o número de pares que levam à diferencial de máxima probabilidade.

A ***correlação*** $C(f, g)$ entre as funções Booleanas f e g pode ser calculada como:

$$C(f, g) = 2 \cdot \text{Prob}(x \mid f(x) = g(x)) - 1.$$

O parâmetro λ de uma S-box S é definido como o valor máximo da correlação

entre combinações lineares dos bits de entrada e combinações lineares de bits de saída de S :

$$\lambda_S = \max_{(i,j) \neq (0,0)} c(\ell_i, \ell_j \circ S).$$

7.2.4 Outras notações utilizadas

Seja uma sequência de funções $f_m, f_{m+1}, \dots, f_{n-1}, f_n$, $m \leq n$, será utilizada a notação:

$$\bigcirc_{r=m}^n f_r \equiv f_n \circ f_{n-1} \circ \dots \circ f_{m+1} \circ f_m$$

e

$$\bigcirc_m^{r=n} f_r \equiv f_m \circ f_{m+1} \circ \dots \circ f_{n-1} \circ f_n.$$

7.3 Estrutura do CURUPIRA

O CURUPIRA é uma cifra de bloco iterativa que opera em *estados* de 96 bits, organizados em matrizes $\mathbb{M}_4$. Ele usa *chaves criptográficas* de $48t$ bits ($2 \leq t \leq 4$), organizadas em matrizes $\mathbb{M}_{2t}$. A seguir, cada uma das operações e constantes que constituem o CURUPIRA serão definidas individualmente; em seguida, a cifra completa será explicitada em termos destes componentes.

7.3.1 Elementos da Encriptação e Decriptação

Primeiramente, serão descritas as operações que compõem os processos de encriptação e decriptação, cujas estruturas satisfazem os requisitos da Estratégia de Trilha Larga.

7.3.1.1 A camada não-linear γ

A função $\gamma : \mathbb{M}_n \rightarrow \mathbb{M}_n$ consiste na aplicação de uma S-box não-linear $S : \text{GF}(2^8) \rightarrow \text{GF}(2^8)$ a todos os bytes de seu argumento de modo individual:

Tabela 1: CURUPIRA: as mini-boxes P e Q

u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$P[u]$	3	F	E	0	5	4	B	C	D	A	9	6	7	8	2	1
$Q[u]$	9	E	5	6	A	2	3	C	F	0	4	D	7	B	1	8

$$\gamma(a) = b \Leftrightarrow b_{i,j} = S[a_{i,j}], \quad 0 \leq i < 3, \quad 0 \leq j < n.$$

A S-box adotada pelo CURUPIRA S-box é a mesma utilizada pelas cifras ANUBIS (BARRETO; RIJMEN, 2000a) e KHAZAD (BARRETO; RIJMEN, 2000b), que apresenta $\delta_S = 2^{-5}$, $\lambda_S = 2^{-2}$, e $\nu_S = 7$ e, assim, previne ataques diferenciais, lineares e de interpolação. Ela foi também concebida de forma que $S[S[x]] = x$, $\forall x \in \text{GF}(2^8)$. Assim, γ é uma involução.

Esta S-box pode ser computada sob demanda usando o algoritmo 1, que o faz a partir de duas *mini-boxes* geradoras (veja tabela 1). O uso destas *mini-boxes* é interessante porque ambas podem ser armazenadas em apenas 16 bytes. Por outro lado, caso haja espaço disponível na plataforma alvo, a S-Box completa pode ser pré-computada durante a compilação ou durante a inicialização do sistema, sendo então armazenada em 256 bytes. Desta forma, a implementação da camada de substituição é, por si só, bastante flexível em termos de velocidade de processamento e uso de memória.

Algorithm 1 Cálculo de $S[u]$ a partir das mini-boxes P e Q

ENTRADA: $u(x) \in \text{GF}(2^8)$, representado como um byte u .

SAÍDA: $S[u]$.

- 1: $u_h \leftarrow P[(u \gg 4) \& F]$, $u_l \leftarrow Q[u \& F]$
 - 2: $u'_h \leftarrow Q[(u_h \& C) \oplus ((u_l \gg 2) \& 3)]$, $u'_l \leftarrow P[((u_h \ll 2) \& C) \oplus (u_l \& 3)]$
 - 3: $u_h \leftarrow P[(u'_h \& C) \oplus ((u'_l \gg 2) \& 3)]$, $u_l \leftarrow Q[((u'_h \ll 2) \& C) \oplus (u'_l \& 3)]$
 - 4: **return** $(u_h \ll 4) \oplus u_l$
-

7.3.1.2 A camada de permutação π

A permutação $\pi : \mathbb{M}_n \rightarrow \mathbb{M}_n$ efetua a troca de duas colunas da matriz passada como seu argumento, de acordo com a regra:

$$\pi(a) = b \Leftrightarrow b_{i,j} = a_{i,i \oplus j}, \quad 0 \leq i < 3, \quad 0 \leq j < n.$$

É fácil perceber que π é uma involução.

7.3.1.3 A camada de difusão linear θ

A camada de difusão $\theta : \mathbb{M}_n \rightarrow \mathbb{M}_n$ é um mapeamento linear baseado no código MDS $[6, 3, 4]$ com matriz geratriz $G_D = [I \ D]$, onde $D = I + 2A + 4B$, ou seja:

$$D = \begin{bmatrix} 3 & 2 & 2 \\ 4 & 5 & 4 \\ 6 & 6 & 7 \end{bmatrix},$$

de forma que

$$\theta(a) = b \Leftrightarrow b = D \cdot a.$$

Como D é auto-inversa, tem-se que θ é uma involução.

A utilização de matrizes circulantes (DAEMEN; KNUDSEN; RIJMEN, 1997; NIST, 2001a) não é adequada para a camada de difusão porque as mesmas não apresentam comportamento involutivo. Matrizes de Cauchy (YOUSSEF; TAVARES; HEYS, 1996; YOUSSEF; MISTER; TAVARES, 1997) poderiam ser uma opção, mas os coeficientes apresentados pelas mesmas são geralmente muito complexos, dificultando uma implementação eficiente. A escolha apresentada envolve os coeficientes mais simples possíveis, no sentido de obter polinômios apresentando baixo grau e também um reduzido Peso de Hamming.

7.3.1.4 A adição de chave $\sigma[k]$

A operação de adição de chave $\sigma[k] : \mathbb{M}_n \rightarrow \mathbb{M}_n$ consiste na adição byte a byte de uma chave $k \in \mathbb{M}_n$, ou seja:

$$\sigma[k](a) = b \Leftrightarrow b_{i,j} = a_{i,j} \oplus k_{i,j}, \quad 0 \leq i < 3, \quad 0 \leq j < n.$$

Esta operação de adição de chave é obviamente uma involução. É importante ressaltar que este mesmo mecanismo é utilizado também para a adição de constantes à chave criptográfica, durante o processo de escalonamento de chaves.

7.3.2 Algoritmo de evolução das chaves - versão 1

Uma primeira idéia de projeto do algoritmo de escalonamento de chaves, descrita a seguir, consiste no uso de uma estrutura semelhante àquela ditada pela própria Estratégia de Trilha Larga. A versão da cifra que adota o algoritmo de escalonamento descrito a seguir é denominada CURUPIRA-1.

7.3.2.1 Representação das chaves

Uma chave $\mathcal{K}$ de $48t$ bits ($2 \leq t \leq 4$), que pode ser interpretada como um vetor de bytes de tamanho $6t$, é representada internamente como uma matriz $K \in \mathbb{M}_{2t}$, de forma que:

$$K_{i,j} = \mathcal{K}[i + 3j], \quad 0 \leq i < 3, \quad 0 \leq j < 2t.$$

Deste modo, a chave K é mapeada por colunas (não por linhas).

7.3.2.2 Constantes de escalonamento

As *constantes de escalonamento* $q^{(s)}$, onde s indica o índice da constante (o round na qual ela será aplicada), são matrizes definidas de forma que $q^{(0)} = 0$ e, para $s > 0$ e

com $0 \leq j < 2t$:

$$q_{i,j}^{(s)} = \begin{cases} S[2t(s-1) + j], & \text{se } i = 0 \\ 0, & \text{senão} \end{cases}$$

A introdução destes elementos no escalonamento de chaves é importante para evitar o aparecimento de “pontos fixos” (valores de sub-chave que repetem-se continuamente ao longo do escalonamento) e, conseqüentemente, a existência de chaves fracas. Boas constantes de escalonamento não devem ser iguais para todos os bytes em um estado e também não devem ser iguais para todos os bits em um byte. Adicionalmente, elas devem assumir valores diferentes a cada round. A escolha atual respeita estas condições, tendo ainda uma vantagem extra: como as constantes são diretamente extraídas da própria S-Box adotada, evita-se a necessidade de armazenamento extra para as mesmas. Na realidade, exatamente t entradas da S-Box são usadas a cada round.

É importante observar que, apesar da simplicidade na escolha destas constantes, o fato das mesmas serem não-lineares pode ser considerada um benefício extra em termos de segurança. O próprio AES, por exemplo, adota uma solução menos complexa do que a do CURUPIRA, já que suas constantes de escalonamento resumem-se a um único bit ‘1’ deslocado consecutivamente para a esquerda (sendo, portanto, lineares). Desta forma, como não há problemas envolvidos na utilização dos valores da própria S-Box, a solução do CURUPIRA é ao mesmo tempo leve e segura.

7.3.2.3 O escalonamento de chaves ψ_s

O escalonamento de chaves é feito por uma transformação reversível, composta por uma adição de constante e duas operações lineares, ξ e μ . A transformação $\xi : \mathbb{M}_{2t} \rightarrow \mathbb{M}_{2t}$ é dada por:

$$\xi(a) = b \Leftrightarrow \begin{cases} b_{0,j} &= a_{0,j} \\ b_{1,j} &= a_{1,(j+1) \bmod 2t} \\ b_{2,j} &= a_{2,(j-1) \bmod 2t} \end{cases}$$

Ou seja, a primeira linha da matriz é mantida inalterada, a segunda linha é deslocada de uma posição para a esquerda e a terceira linha é deslocada de uma posição para a direita. Todas as rotações são cíclicas, ou seja, cada linha é tratada como se o seu último elemento fosse sucedido pelo primeiro.

Já a transformação $\mu : \mathbb{M}_n \rightarrow \mathbb{M}_n$ se escreve como:

$$\mu(a) = E \cdot a,$$

onde $E = I + cC$, c sendo a raiz cúbica da unidade: $c(x) = x^4 + x^3 + x^2 \in \text{GF}(2^8)$. A operação inversa é dada por $\mu^{-1}(a) = E^{-1} \cdot a$ onde $E^{-1} = I + (c + 1)C$.

Seja $a \in \mathbb{M}_{2t}$ e $\omega \equiv \mu \circ \xi$, onde ω^m denota a composição da operação ω consigo mesma por m vezes. Por inspeção direta, pode-se verificar que o período de ω sobre $\mathbb{M}_{2t}$ é igual $6t$, com $2 \leq t \leq 4$. Em outras palavras, $m = 6t$ é o menor inteiro positivo que satisfaz a relação $\omega^m(a) = a$, $\forall a \in \mathbb{M}_{2t}$. A idéia consiste no cálculo de ω^m em uma base de $\mathbb{M}_{2t}$, por exemplo $\{e^{(kl)} \mid e_{ij}^{(kl)} = \delta_{ki}\delta_{lj}\}$, e na verificação que $\omega^m(e^{(kl)}) \neq e^{(kl)}$ para $1 \leq m < 6t$ mas $\omega^m(e^{(kl)}) = e^{(kl)}$ para $m = 6t$.

Seja a constante de escalonamento acumulada $Q^{(s)} = \sum_{i=0}^s \omega^{s-i+1}(q^{(i)})$. Seja ainda $K \in \mathbb{M}_n$ a chave utilizada pela cifra. A primeira chave é definida como $K^{(0)} \equiv K$. A **função de escalonamento** $\psi_r : \mathbb{M}_n \rightarrow \mathbb{M}_n$ é usada para calcular a chave $K^{(r)}$, do r -ésimo round, a partir da chave $K^{(r-1)}$. Tal função é definida como $\psi_r \equiv \omega \circ \sigma[q^{(r)}]$, ou seja:

$$K^{(r)} = \psi_r(K^{(r-1)}) = \left(\bigcirc_{i=1}^r \omega \circ \sigma[q^{(i)}] \right)(K) = \omega^r(K) + Q^{(r)}.$$

Desta forma, a função de escalonamento deriva sub-chaves a partir das chaves de round $K^{(0)}$ a $K^{(m-1)}$. Percebe-se que uma aplicação extra de ω seguida da adição de $Q^{(m)}$ resultam na recuperação da chave original, ou seja: $\omega^m(K) = K = \omega(K^{(m-1)}) + \omega(Q^{(m-1)})$. Portanto, apenas $\omega(Q^{(m-1)})$ precisa ser armazenado para que seja possível realizar um **processamento seqüencial cíclico**: em todos os outros pas-

sos do escalonamento, exceto este último, as constantes $q^{(r)}$ descritas anteriormente podem ser usadas. Já a aplicação da constante $\omega(Q^{(m-1)})$ no último round leva à reinitialização da chave.

Além disto, a função de escalonamento inversa $\psi_r^{-1} : \mathbb{M}_n \rightarrow \mathbb{M}_n$ assume a forma:

$$\psi_r \equiv \sigma[q^{(r)}] \circ \omega^{-1} = \sigma[q^{(r)}] \circ \xi^{-1} \circ \mu^{-1}$$

Graças à escolha do valor da constante c , esta expressão torna-se particularmente simples de ser resolvida em razão da forma assumida pela matriz E^{-1} que compõe a operação μ^{-1} . Adicionalmente, a função ξ é facilmente inversível. Desta forma, com um custo computacional semelhante, é possível derivar as sub-chaves utilizadas pela cifra tanto na ordem direta quanto inversa.

O comportamento cíclico apresentado pelo escalonamento de chaves do CURUPIRA dispensa de imediato a necessidade de armazenamento de sub-chaves, durante ambos os processos de encriptação e decriptação. Como a cifra foi desenvolvida para utilização em redes nas quais a memória disponível é escassa, tal característica assume um papel muito importante quando a cifra é utilizada nestas plataformas. Dispositivos com maior capacidade de armazenamento (como servidores), por outro lado, podem pré-calcular estas chaves e armazená-las em uma tabela para pronto uso pelos diferentes blocos encriptados/decriptados, acelerando o processamento dos mesmos.

7.3.2.4 A seleção de chaves ϕ_r

As sub-chaves $\kappa^{(r)}$ efetivamente utilizadas a cada round do CURUPIRA são computadas (a partir da chave inicial K ou então de uma sub-chave $K^{(r)}$) por meio da **função de seleção de chaves**, $\phi_r : \mathbb{M}_n \rightarrow \mathbb{M}_n$, definida por:

$$\kappa^{(r)} = \phi_r(K) \Leftrightarrow \kappa_{0,j}^{(r)} = S[K_{0,j}^{(r)}] \text{ e } \kappa_{i,j}^{(r)} = K_{i,j}^{(r)} \text{ para } i > 0, 0 \leq j < 4.$$

Ou seja, efetua-se a substituição (com a S-Box previamente apresentada) dos bytes na primeira linha da chave, enquanto os outros bytes permanecem inalterados. Além disto, como o tamanho das chaves deve ser igual ao dos blocos utilizados pela cifra, as chaves são truncadas de forma a apresentar exatamente 3 linhas e 4 colunas.

Assim, a função de seleção introduz não-linearidade no escalonamento de chaves, garantindo a ocorrência de 8 aplicações da S-Box entre quaisquer duas sub-chaves e também $4(R + 1)$ aplicações da S-Box durante todo o processo de escalonamento, onde R corresponde ao número de rounds.

7.3.3 Algoritmo de evolução das chaves - versão 2

O algoritmo de escalonamento de chaves do CURUPIRA-1 é bastante conservador, buscando grande robustez contra ataques de chaves relacionadas. Entretanto, em cenários em que este tipo de ataque seja pouco viável, a operação de geração de chaves pode ser ainda mais rápida, tanto no sentido direto quanto inverso. Esta segunda proposta de algoritmo de escalonamento, que dá origem ao CURUPIRA-2, apresenta elementos essenciais para evitar chaves fracas e mesmo ataques de chaves relacionadas, apesar de possuir uma capacidade de difusão menor do que no caso anterior. Ela será descrita em detalhes nas próximas sessões.

7.3.3.1 Representação das chaves

Durante o processo de escalonamento, uma chave $\mathcal{K}$ de $48t$ bits ($2 \leq t \leq 4$) é representada internamente como um elemento K do corpo $\text{GF}(2^{48t}) = \text{GF}(2)/p_{48t}(x)$, onde $p(x)_{48t}$ é um pentanômio em $\text{GF}(2)$ escolhido de modo que x^8 é uma raiz primitiva de $p(x)$. Um elemento $u(x)$ deste corpo pode ser representados na forma de um vetor de bytes U_i , ou seja, $u = (U_{6t-1}, \dots, U_0)$, onde o índice 0 indica o byte menos significativo. A chave $\mathcal{K}$ é mapeada de forma direta, com seu primeiro byte assumindo a posição do byte K_{6t-1} .

A escolha de um pentanômio tem razões práticas, já que eles existem para todo t de interesse (IEEE, 2000), o mesmo não valendo necessariamente para trinômios (que levariam a um algoritmo mais rápido caso existissem). Por razões que ficarão mais claras na seção 7.3.3.3 a seguir, os pentanômios escolhidos foram:

- $p_{96}(x) = x^{96} + x^{16} + x^{13} + x^{11} + 1$;
- $p_{144}(x) = x^{144} + x^{56} + x^{53} + x^{51} + 1$;
- $p_{192}(x) = x^{192} + x^{43} + x^{41} + x^{40} + 1$.

7.3.3.2 Constantes de escalonamento

As **constantes de escalonamento** $q^{*(s)}$ (onde s indica o round em que ela é aplicada) são interpretadas como elementos do corpo $\text{GF}(2^{48t})$, e construídas a partir dos valores de saída da S-Box, de modo que $q^{*(0)} = 0$ e, para $s > 0$:

$$q_i^{*(s)} = \begin{cases} S[s-1], & \text{se } i = 6t-1 \\ 0, & \text{senão} \end{cases}$$

Assim, uma única saída da S-Box é mapeada no byte mais significativo de $q^{*(s)}$. Conforme será mostrado na próxima seção, o byte mais significativo é uma posição estratégica, fazendo com que a adição da constante afete exatamente 3 bytes da chave logo após sua aplicação.

7.3.3.3 O escalonamento de chaves Υ_s

O escalonamento de chaves envolve a transformação reversível $\aleph : \text{GF}(2^{48t}) \rightarrow \text{GF}(2^{48t})$, definida como $\aleph(u) \equiv u \cdot x^8$, e também a transformação auto-inversa $\eta : \text{GF}(2^{48t}) \rightarrow \text{GF}(2^{48t})$, definido de forma que, para os polinômios $u = (U_{6t-1}, \dots, U_0)$ e $v = (V_{6t-1}, \dots, V_0)$ em $\text{GF}(2^{48t})$:

$$v = \eta(u) \Leftrightarrow \begin{cases} V_i = U_{11-i} \oplus U_{12+i} & \text{se } 0 \leq i < 6t - 12; \\ V_i = U_i & \text{senão.} \end{cases}$$

Assim, a **função de escalonamento** $\Upsilon_r : \text{GF}(2^{48t}) \rightarrow \text{GF}(2^{48t})$ é definida como $\Upsilon_r(u) \equiv \eta \circ \aleph(u \oplus q^{*(r)})$, de modo que $K^{(0)} \equiv K$ e $K^{(r+1)} = \Upsilon_r(K^{(r)})$.

O transformação η é usada para prover maior difusão ao escalonamento quando chaves maiores do que 96 bits são utilizadas, fazendo com que os bytes mais significativos da chave sejam combinados com os menos significativos. Caso a mesma fosse removida, duas chaves de 144 bits distintas apenas na posição K_{11} gerariam subchaves cujos 12 bytes menos significativos seriam idênticos durante os primeiros 6 rounds, enquanto este comportamento seria observado por 12 rounds para chaves de 192 bits. Considerando que a função de seleção de chaves (descrita a seguir, na seção 7.3.3.4) atua somente sobre estes bytes menos significativos, tal comportamento seria inadequado em termos de segurança.

Já a escolha da operação $\aleph$ é particularmente interessante porque a mesma é bastante leve, especialmente em plataformas orientadas a bytes, conforme demonstra o teorema a seguir:

Teorema 1. *Seja $p(x) = x^n + x^{k_3} + x^{k_2} + x^{k_1} + 1$ um pentanômio primitivo de grau $n = bw$ em $\text{GF}(2)$ satisfazendo $k_3 > k_2 > k_1$, $k_3 - k_1 \leq w$ e também k_3 e/ou k_1 divisíveis por w . Então, a multiplicação por x^w em $\text{GF}(2^n) = \text{GF}(2)[x]/p(x)$ pode ser implementada usando não mais do que 5 XORs e 4 deslocamentos sobre palavras de w bits. Além disto, se 2×2^w bytes para armazenamento estiverem disponíveis, o custo fica inferior a 2 XORs sobre palavras de w bits e 2 buscas em tabelas.*

Demonstração Para $u = \bigoplus_{d=0}^{n-1} u_d x^d \in \text{GF}(2^n)$, seja $U_i = u_{wi+w-1}x^{w-1} + \dots + u_{wi}$, $i = 0, \dots, b-1$, de modo que $u = U_{b-1}x^{w(b-1)} + U_{b-2}x^{w(b-2)} + \dots + U_0$, que podem ser

escritos como $u = (U_{b-1}, \dots, U_0)$. Então, pode-se calcular $u \cdot x^w$ da seguinte forma:

$$\begin{aligned} & (U_{b-1}x^{w(b-1)} + U_{b-2}x^{w(b-2)} + \dots + U_0) \cdot x^w = \\ & U_{b-1}x^n + U_{b-2}x^{w(b-1)} + \dots + U_0x^w = \\ & U_{b-2}x^{w(b-1)} + \dots + U_0x^w + U_{b-1}(x^{k_3} + x^{k_2} + x^{k_1} + 1) = \\ & (U_{b-2}, \dots, U_0, U_{b-1}) \oplus U_{b-1}(x^{k_3} + x^{k_2} + x^{k_1}). \end{aligned}$$

Considere que $k_1 = wk$ para algum $k \geq 0$; para $k_3 = w$, com $k \geq 1$, a demonstração é feita de modo análogo. Tem-se que:

$$u \cdot x^w = (U_{b-2}, \dots, U_0, U_{b-1}) \oplus U_{b-1}(x^{k_3-k_1} + x^{k_2-k_1} + 1)x^{wk}.$$

Seja $gr(p)$ o grau do polinômio p . Como $gr(U_{b-1}) \leq w-1$ e $gr(x^{k_3-k_1} + x^{k_2-k_1} + 1) \leq w$, o seu produto é um polinômio de grau menor do que $2w-1$ e, assim, pode ser armazenado em apenas duas palavras de w bits, qualquer que seja U_{b-1} . Além disto, a multiplicação deste valor por x^{wk} corresponde a um simples deslocamento de k palavras para a esquerda. Então, pode-se escrever $u \cdot x^w = (U_{b-2}, \dots, U_k \oplus T_1[U_{b-1}], U_{k-1} \oplus T_0[U_{b-1}], \dots, U_0, U_{b-1})$, onde

$$\begin{aligned} T_0[U] & \equiv (U \ll (k_3 - k_1)) \oplus (U \ll (k_2 - k_1)) \oplus U; \\ T_1[U] & \equiv (U \gg (w - (k_3 - k_1))) \oplus (U \gg (w - (k_2 - k_1))). \end{aligned}$$

Os valores de T_1 e T_0 tanto podem ser calculados sob demanda ou pré-calculados e armazenados em duas tabelas contendo 2^w elementos. Por inspeção direta, conclui-se que o custo computacional resultante é aquele dado pelo teorema. $\square$

Usando este teorema para os polinômios escolhidos no CURUPIRA-2, pode-se avaliar o efeito da função $\aleph$ e de sua inversa:

$$p_{96}(x) = x^{96} + x^{16} + x^{13} + x^{11} + 1 :$$

$$(U_{11}, \dots, U_0) \cdot x^8 = (U_{10}, \dots, U_1 \oplus T_1[U_{11}], U_0 \oplus T_0[U_{11}], U_{11});$$

$$(U_{11}, \dots, U_0) \cdot x^{-8} = (U_0, U_{11}, \dots, U_2 \oplus T_1[U_0], U_1 \oplus T_0[U_0]);$$

$$p_{144}(x) = x^{144} + x^{56} + x^{53} + x^{51} + 1 :$$

$$(U_{17}, \dots, U_0) \cdot x^8 = (U_{16}, \dots, U_7, U_6 \oplus T_1[U_{17}], U_5 \oplus T_0[U_{17}], U_4, \dots, U_0, U_{17});$$

$$(U_{17}, \dots, U_0) \cdot x^{-8} = (U_0, U_{17}, \dots, U_8, U_7 \oplus T_1[U_0], U_6 \oplus T_0[U_0], U_5, \dots, U_1);$$

$$p_{192}(x) = x^{192} + x^{48} + x^{45} + x^{43} + 1 :$$

$$(U_{23}, \dots, U_0) \cdot x^8 = (U_{22}, \dots, U_6, U_5 \oplus T_1[U_{23}], U_4 \oplus T_0[U_{23}], U_3, \dots, U_0, U_{23}).$$

$$(U_{23}, \dots, U_0) \cdot x^{-8} = (U_0, U_{23}, \dots, U_7, U_6 \oplus T_1[U_0], U_5 \oplus T_0[U_0], U_4, \dots, U_1).$$

Qualquer que seja o tamanho da chave, T_0 e T_1 assumem a forma:

$$T_1[U] \equiv U \oplus (U \gg 3) \oplus (U \gg 5);$$

$$T_0[U] \equiv (U \ll 5) \oplus (U \ll 3).$$

A análise destas equações mostra que custo de $\aleph^{-1}$ é exatamente igual ao custo de $\aleph$ e, portanto, o custo de Υ_r é equivalente ao custo de $\Upsilon_r^{-1}(u) \equiv \aleph^{-1} \circ \eta(u) \oplus q^{*(r)}$. A desvantagem desta segunda versão do escalonamento é que não existe uma forma tão simples de reinicializar a chave. No entanto, em muitas aplicações, o reduzido número de operações envolvidas em ambos os sentidos do escalonamento pode ser uma vantagem mais significativa do que sua ciclicidade.

7.3.3.4 A seleção de chaves ϕ_r^*

As sub-chaves $\kappa^{(r)} \in \mathbb{M}_n$ utilizadas a cada round do CURUPIRA-2 são calculadas por meio da *função de seleção de chaves*, $\phi_r^* : \text{GF}(2^{48t}) \rightarrow \mathbb{M}_n$, definida por:

$$\kappa^{(r)} = \phi_r^*(K) \Leftrightarrow \begin{cases} \kappa_{i,j}^{(r)} = S[K_{i+3j}^{(r)}] & \text{se } i = 0; \\ \kappa_{i,j}^{(r)} = K_{i+3j}^{(r)} & \text{senão.} \end{cases}$$

Desta forma, a S-Box é aplicada apenas nos bytes da chave que serão combinados com a primeira linha do bloco, adicionando não-linearidade à cifra de modo idêntico ao que ocorre com a seleção de chaves ϕ_r da primeira versão do escalonamento do CURUPIRA. Os outros bytes da chave são usados diretamente e, novamente, as chaves são truncadas para 12 bytes: apenas os bytes menos significativos são efetivamente utilizados a cada round.

7.3.4 A cifra completa

O CURUPIRA é definido, para uma chave inicial $K \in \mathbb{M}_n$ e R rounds, como a transformação $\text{CURUPIRA}[K] : \mathbb{M}_4 \rightarrow \mathbb{M}_4$ dada por:

$$\text{CURUPIRA}[K] \equiv \sigma[\kappa^{(R)}] \circ \pi \circ \gamma \circ \left(\bigcirc_{r=1}^{R-1} \sigma[\kappa^{(r)}] \circ \theta \circ \pi \circ \gamma \right) \circ \sigma[\kappa^{(0)}].$$

À adição de chave inicial $\sigma[\kappa^{(0)}]$ dá-se o nome de *branqueamento* (ou *whitening*). A composição de operações $\rho[\kappa^{(r)}] \equiv \sigma[\kappa^{(r)}] \circ \theta \circ \pi \circ \gamma$ é chamada *função de round* para o r -ésimo round; por conveniência, o mapeamento $\rho'[\kappa^{(R)}] \equiv \sigma[\kappa^{(R)}] \circ \pi \circ \gamma$ é chamado de *última função de round*, apesar de não ser exatamente igual à função de round completa já que a operação θ não está presente.

Um total de $R + 1$ sub-chaves são necessárias para a operação da cifra. Como o escalonamento de chaves é cíclico e apresenta um período $m = 6t$, $2 \leq t \leq 4$, o número máximo de rounds que a cifra pode apresentar é $6t - 1$. O número exato de

rounds pode, entretanto, variar: para chaves de $48t$ bits, pode ser usado qualquer valor no intervalo entre $4t - 2 \leq R \leq 6t - 1$ (veja tabela 2). De qualquer forma, as sub-chaves de cada round sempre podem ser computadas de modo independente e em qualquer ordem desejada.

Tabela 2: CURUPIRA - número de rounds permitido para cada tamanho de chave

Tamanho da chave (bits)	min # rounds	max # rounds
96	10	11
144	14	17
192	18	23

7.3.5 A cifra inversa

A seguir é demonstrado que o CURUPIRA é uma cifra involutiva, de modo que a única diferença entre os processos de encriptação e decríptação reside no escalonamento de chaves.

Teorema 2. *Seja $\alpha[\kappa^{(0)}, \dots, \kappa^{(R)}]$ a operação de encriptação realizada pelo CURUPIRA sob a seqüência de chaves de round $\kappa^{(0)}, \dots, \kappa^{(R)}$, e sejam as chaves de decríptação definidas como $\bar{\kappa}^{(0)} \equiv \kappa^{(R)}$, $\bar{\kappa}^{(R)} \equiv \kappa^{(0)}$, e $\bar{\kappa}^{(r)} \equiv \theta(\kappa^{(R-r)})$ para $0 < r < R$. Desta maneira, tem-se que $\alpha^{-1}[\kappa^{(0)}, \dots, \kappa^{(R)}] = \alpha[\bar{\kappa}^{(0)}, \dots, \bar{\kappa}^{(R)}]$.*

Demonstração Por definição:

$$\alpha[\kappa^{(0)}, \dots, \kappa^{(R)}] = \sigma[\kappa^{(R)}] \circ \pi \circ \gamma \circ \left(\bigcirc_{r=1}^{R-1} \sigma[\kappa^{(r)}] \circ \theta \circ \pi \circ \gamma \right) \circ \sigma[\kappa^{(0)}].$$

Como cada função que compõe a expressão acima é uma involução, o inverso da mesma pode ser obtido pela aplicação de cada componente em ordem inversa. Entretanto, percebe-se que $\gamma \circ \pi = \pi \circ \gamma$, já que π move os elementos da matriz à qual é aplicado sem alterá-los, enquanto γ atua apenas em elementos individuais, independentemente de suas posições. Além disto, $\theta \circ \sigma[\kappa^{(r)}] = \sigma[\theta(\kappa^{(r)})] \circ \theta$, pois

$(\theta \circ \sigma[\kappa^{(r)}])(a) = \theta(\kappa^{(r)} + a) = \theta(\kappa^{(r)}) + \theta(a) = (\sigma[\theta(\kappa^{(r)})] \circ \theta)(a)$, qualquer que seja a entrada a . Portanto, pode-se escrever:

$$\alpha^{-1}[\kappa^{(0)}, \dots, \kappa^{(R)}] = \sigma[\kappa^{(0)}] \circ \left(\bigcirc_{r=1}^{R-1} \pi \circ \gamma \circ \sigma[\theta(\kappa^{(r)})] \circ \theta \right) \circ \pi \circ \gamma \circ \sigma[\kappa^{(R)}].$$

Substituindo $\bar{\kappa}^{(r)}$ na equação anterior e mudando ligeiramente os agrupamentos das operações, chega-se ao resultado desejado:

$$\alpha^{-1}[\kappa^{(0)}, \dots, \kappa^{(R)}] = \sigma[\bar{\kappa}^{(R)}] \circ \pi \circ \gamma \circ \left(\bigcirc_{r=1}^{R-1} \sigma[\bar{\kappa}^{(r)}] \circ \theta \circ \pi \circ \gamma \right) \circ \sigma[\bar{\kappa}^{(0)}] = \alpha[\bar{\kappa}^{(0)}, \dots, \bar{\kappa}^{(R)}].$$

□

7.4 Análise de Segurança

Para todos os possíveis tamanhos de chaves, tem-se como objetivo de segurança que o CURUPIRA seja ***K-Seguro*** e ***hermético*** conforme definido em (DAEMEN, 1995) e explicitados a seguir por razões de clareza:

- Uma cifra de bloco é ***K-segura*** se todas as possíveis estratégias de ataque contra ela requerem um esforço computacional e capacidade de armazenamento semelhantes à maioria das cifras de bloco possíveis com as mesmas dimensões (tamanho de bloco e chave), ou seja, ela não deve ser mais facilmente atacável do que cifras similares. Este deve ser o caso para qualquer modalidade de ataque e para qualquer chave utilizada.
- Uma cifra de bloco é dita ***hermética*** se ela não apresenta fraquezas ausentes na maioria das cifras de bloco com as mesmas dimensões de bloco e chave.

A seguir são apresentados os resultados da análise de segurança do CURUPIRA. Tal análise consiste na avaliação da robustez da cifra com relação às principais técnicas de criptoanálise conhecidas atualmente. Apesar de ser difícil definir com certeza se

esta é a melhor forma de se avaliar a segurança de uma cifra (PRENEEL et al., 2000), esta é uma abordagem comumente adotada no domínio da criptografia, aparecendo extensivamente durante a escolha do próprio AES.

7.4.1 Criptoanálise Diferencial e Linear

Conforme discutido na seção 5.7, a adoção de uma matriz MDS [6,3,4] para a camada θ faz com que toda característica diferencial ou aproximação linear de 4 rounds apresente ao menos $4^2 = 16$ S-Boxes ativas. Em consequência, usando a notação definida em 7.2.3, nenhuma característica diferencial de 4 rounds tem probabilidade de ocorrência maior do que $\delta_s^{16} = (2^{-5})^{16} = 2^{-80}$, enquanto nenhuma aproximação linear para 4 rounds tem uma correlação entrada-saída maior do que $\lambda_s^{16} = (2^{-2})^{16} = 2^{-32}$. Conforme discutido nas seções 5.2 e 5.3, ataques diferenciais ou lineares clássicos necessitariam de características diferenciais com probabilidade maiores do que $2^{1-n_b} = 2^{-95}$ ou correlações entre entrada e saída acima de $2^{-n_b/2} = 2^{-48}$ quando se consideram *todos* os rounds. Assim, a viabilidade destes ataques (bem como algumas variantes mais avançadas dos mesmos, como ataques diferencial-lineares) contra a cifra completa é virtualmente nula.

Ataques baseados em criptoanálise linear podem muitas vezes ser melhorados pela utilização de aproximações não-lineares (KNUDSEN; ROBSHAW, 1996). Todavia, o atual estado da arte destas técnicas parece limitá-las ao primeiro e/ou último round de uma aproximação linear. Esta limitação parece ainda mais difícil de se superar para cifras que usam S-Boxes altamente não-lineares, como é o caso do CURUPIRA.

7.4.2 Criptoanálise Diferencial Truncada

Um ataque diferencial truncado similar ao descrito em (DAEMEN; RIJMEN, 1998, seção 3.2 e Apêndice A) contra 7 rounds do BKSQ pode ser aplicado ao CURUPIRA reduzido ao mesmo número de rounds, usando as mesmas estruturas descritas no ata-

que original. Entretanto, assim como ocorre para o BKSQ, nenhum ataque diferencial truncado mais rápido do que uma busca exaustiva foi encontrado para 8 ou mais rounds do CURUPIRA.

7.4.3 Ataques por interpolação

Ataques por interpolação (cf. 4.3.4) geralmente dependem da presença de estruturas algébricas simples em alguns componentes da cifra (particularmente na S-Box), permitindo a construção de expressões de complexidade administrável. No entanto, este está longe de ser o caso da S-Box do CURUPIRA. A intrincada expressão algébrica envolvida na construção desta S-Box pseudo-aleatória (lembrando que o seu fator $v_S = 7$), combinada ainda com o efeito introduzido pela camada de difusão, aparentemente fazem com que este tipo de ataque seja impraticável para mais do que alguns poucos rounds da cifra proposta.

7.4.4 Chaves fracas e Chaves relacionadas

Chaves fracas tipicamente ocorrem em cifras nas quais as operações não-lineares dependem do valor atual da chave. Este não é o caso do CURUPIRA, onde as chaves são aplicadas por meio de operações de XOR e toda a não-linearidade encontra-se fixada na S-Box. Além disto, a não-linearidade das constantes de escalonamento utilizadas reduzem consideravelmente a probabilidade de aparecimento de pontos fixos no processo de escalonamento, diminuindo ainda mais a possibilidade de existência de chaves fracas.

Ataques de chaves relacionadas normalmente dependem da difusão lenta e/ou simetria durante o escalonamento de chaves. Entretanto, o escalonamento de chaves do CURUPIRA-1 foi projetado de acordo com a Estratégia de Trilha Larga, capaz de propagar as diferenças na chave inicial de maneira rápida, além e apresentar uma camada de difusão não-linear. Isto faz com que ataques de chaves relacionadas, incluindo va-

riantes como aquela descrita em (FERGUSON et al., 2000, seção 4), tenham uma baixa probabilidade de sucesso.

Já o escalonamento de chaves do CURUPIRA-2 apresenta uma difusão mais lenta já que, entre dois rounds quaisquer, a diferença em um único byte interno (i.e. em uma posição que é apenas permutadas como resultado da multiplicação por x^8) resulta na diferença de apenas 1 byte na sub-chave seguinte. Porém, a estrutura do algoritmo de evolução, em conjunto com a não-linearidade introduzida pela função de seleção de chave, ajudam a prevenir modalidades comuns de ataques de chave relacionada, como aqueles que envolvem diferenciais e rotações (cf. (KELSEY; SCHNEIER; WAGNER, 1996)). Além disto, aliado à grande capacidade de difusão da função de round do CURUPIRA, ele garante que cada byte da chave afetará um grande número de bytes do bloco.

Para chaves de tamanho maior do que o tamanho do bloco, é inevitável a existência de grupos de chaves que produzem resultados idênticos para pelo menos uma chave de round. Fica em aberto a possibilidade de determinar grupos de chaves capazes de produzir chaves de round idênticas para dois ou mais rounds consecutivos, principalmente no caso do CURUPIRA-1. Contudo, mesmo supondo sua existência, é difícil dizer como tais chaves poderiam ser usadas em um ataque de chaves relacionadas.

7.4.5 Ataque de Saturação

Ataques de Saturação, também conhecidos como Ataques Integrais, estão entre os ataques mais efetivos contra cifras que seguem a Estratégia de Trilha Larga, considerando um número reduzido de rounds. A Tabela 3 resume as complexidades de ataques de saturação contra o CURUPIRA, evidenciando esta modalidade de ataque deixa de ser interessante quando se considera mais do que 7 rounds da cifra. A forma como este ataque pode ser aplicado ao CURUPIRA é descrita em detalhes no Apêndice B.

Tabela 3: CURUPIRA: complexidade de ataques de saturação

rounds (n)	textos-claros	Encriptações de n -rounds
4	2^9	2^9
5	2^{11}	2^{35}
6	2^{27}	2^{37}
7	$2^{96} - 2^{87}$	2^{88}

7.4.6 O ataque Gilbert-Minier

O ataque Gilbert-Minier (GILBERT; MINIER, 2000) é capaz de quebrar 7 rounds do CURUPIRA com 2^{24} testes para uma coluna da chave do primeiro round $\times 2^{12}$ c -grupos $\times 48$ buscas em uma S-Box por entrada $\times 2^{64}$ entradas/tabelas $\times 2$ tabelas, ou aproximadamente 2^{110} buscas em uma S-Box (2^{104} encriptações de 7 rounds), além de 2^{32} textos claros escolhidos. É difícil dizer se a aceleração contra 7 rounds do RIJNDAEL com chaves de 128 bits pode ser modificada para funcionar contra 7 rounds do CURUPIRA com chaves de 96 bits. Caso isto seja possível, até 7 rounds da cifra poderiam ser quebrados ligeiramente mais rápido do que uma busca exaustiva.

7.4.7 Ataque de extensão geral

Qualquer ataque a n rounds de uma cifra pode ser estendido a $n + 1$ ou mais rounds simplesmente adivinhando a chave $\kappa^{(n+1)}$ por inteiro e efetuando um ataque a n -rounds da mesma (FERGUSON et al., 2000). Cada round extra aumenta a complexidade do ataque por um fator de 2^{96} buscas na S-Box. Como a complexidade de um ataque de saturação a 6 rounds do CURUPIRA (considerando o melhoramento proporcionado pela soma parcial) é de 5×2^{40} buscas na S-Box, uma extensão a 7 rounds custaria 5×2^{136} buscas, ou cerca de 2^{132} encriptações de 7 rounds. Caso isto seja possível, este ataque seria mais rápido do que uma busca exaustiva para chaves de 144 e 192 bits.

7.4.8 Ataques algébricos e outros ataques

Conforme discutido na seção 4.3.8, não há consenso em relação à efetividade de ataques algébricos como o método XSL (COURTOIS; PIEPRZYK, 2002). Em razão da natureza heurística destes ataques, a análise da complexidade teórica dos mesmos continua controversa, e a falta de evidências experimentais parece muitas vezes a negar sua viabilidade. Até a data em que este documento foi escrito, nenhum ataque deste tipo chegou a ser demonstrado, nem mesmo para versões simplificadas do AES, cuja S-Box apresenta uma estrutura algébrica bem menos complexa do aquela do CURUPIRA. Ainda, é importante ressaltar que esta mesma S-Box, em outro contexto, já foi motivo de estudos com a explícita intenção de testar sua resistência contra ataques algébricos (BIRYUKOV; DECANNIÈRE, 2003) e nenhuma evidência de fraqueza foi encontrada.

O Ataque Bumerangue mostra-se bastante efetivo contra cifras cuja robustez é diferente nos processos de encriptação e decríptação; este dificilmente é o caso do CURUPIRA, em razão de sua estrutura involutiva. Já a eficácia de *Slide Attacks* é reduzida pela assimetria introduzida no escalonamento das chaves, com as constantes utilizadas em ambas as versões propostas.

7.4.9 Conclusões sobre segurança

Conforme o resultado da análise de segurança acima, nenhuma técnica de criptoanálise conhecida pôde ser aplicada ao CURUPIRA completo com um resultado melhor do que uma busca exaustiva. De qualquer forma, é importante ressaltar que, como a cifra foi desenvolvida de acordo com a Estratégia de Trilha Larga, é de se esperar que a mesma apresente um elevado nível de segurança mesmo contra técnicas que não foram consideradas com detalhes neste documento. Esta afirmação tem base no que observa-se atualmente sobre o próprio AES: apesar de diversas tentativas de criptoanálise (muitas delas consistindo em variantes dos ataques descritos neste documento), até o momento nenhuma técnica foi capaz de colocar em xeque esta que é a mais

conhecida cifra seguidora de tal metodologia de projeto.

Um último ponto importante com relação à segurança do CURUPIRA diz respeito ao tamanho de chaves adotado. Considerando que a quantidade de informações trocadas em redes formadas por dispositivos com recursos restritos costuma ser reduzida em comparação a outras redes, o menor tamanho de chave adotado, de 96 bits, provê um nível de segurança que parece bastante adequado à maioria das aplicações nestas plataformas. Esta observação é particularmente verdadeira quando se considera que boa parte destas aplicações utilizam atualmente o Skipjack, que adota chaves de apenas 80 bits. Já o maior tamanho de chave permitido para o CURUPIRA, de 192 bits, pode ser considerado bem elevado para estas redes mesmo quando se considera um futuro bem distante. Além disto, a possibilidade de se utilizar um número variável de rounds adiciona flexibilidade à cifra, permitindo um ajuste mais fino de seu nível segurança dependendo dos requisitos da aplicação em questão.

7.5 Implementação e desempenho

Conforme brevemente discutido ao longo do texto, o CURUPIRA é bastante flexível em termos de implementação, existindo diversas possibilidades de otimização dependendo dos recursos disponíveis na plataforma alvo.

Para um processador de 8 bits com uma disponibilidade limitada de memória RAM, a camada de substituição não-linear pode ser feita byte a byte, seja utilizando uma tabela completa de 256 bytes ou por meio das mini-boxes P e Q (cf. 7.3.1.1). Isto também vale para a transformação $\sigma[k]$.

A geração de sub-chaves, por sua vez, pode ser feita sob demanda, dispensando a necessidade de armazenamento das chaves tanto no processo de encriptação quanto de decrptação. Para o CURUPIRA-1, além da chave atual, é necessário armazenar apenas o valor da constante de reinicialização da chave, $Q^{(m-1)}$ (cf. 7.3.2.3), que possui

exatamente o mesmo tamanho da chave utilizada. Já o CURUPIRA-2 não apresenta comportamento cíclico, mas o custo do cálculo de cada sub-chave de tamanho $6t$, $2 \leq t \leq 4$, em qualquer sentido, envolve uma permutação circular, $2+6(t-2)$ XORs e o cálculo de T_0 e T_1 . Estes últimos podem ser implementados na forma de tabelas ou computados sob demanda, com custo correspondente a 1 XOR + dois deslocamentos e 2 XORs + dois deslocamentos, respectivamente. Já a operação de permutação não precisa ser efetivamente implementada, já que bastaria armazenar o índice indicando o byte menos significativo da chave; este índice seria, então, atualizado a cada chamada de $\mathfrak{N}$ ou $\mathfrak{N}^{-1}$, simplificando a implementação. Portanto, a última sub-chave pode ser calculada a baixo custo no início do processo de decifração ou, caso haja memória disponível, isto pode ser feito logo na inicialização da cifra.

A multiplicação pelo polinômio $g(x) = x$ em $\text{GF}(2^8)$ é de central importância. Ela pode ser implementada como um deslocamento para a esquerda e um XOR condicional (para redução módulo 14D). Caso haja memória disponível, pode-se adotar uma implementação ainda mais eficiente pelo uso de uma tabela X de 256 bytes, com $X[u] \equiv x \cdot u$ (DAEMEN; RIJMEN, 2002, seção 4.1.1). Qualquer que seja a escolha, denota-se por $\text{xtimes}(u)$ o resultado de $x \cdot u(x)$.

Para o escalonamento do CURUPIRA-1, a multiplicação pelo polinômio $c(x) = x^4 + x^3 + x^2 \in \text{GF}(2^8)$ é igualmente importante. Denota-se por $\text{ctimes}(u)$ o resultado de $c(x) \cdot u(x)$. Ele pode ser implementado como $\text{ctimes}(u) \equiv \text{xtimes}(\text{xtimes}(\text{xtimes}(\text{xtimes}(u) \oplus u) \oplus u))$, custando então 2 XORs e 4 aplicações de xtimes .

Para as operações θ e ψ_s , é necessário implementar a multiplicação pelas matrizes D e E , respectivamente. O Algoritmo 2 calcula $D \cdot a$ para um vetor $a \in \mathbb{M}_1$ (ou seja, a coluna de uma matriz pertencente a $\mathbb{M}_n$) ao custo de 6 XORs e 2 operações de xtimes .

O Algoritmo 3 calcula $E \cdot a$ para um vetor $a \in \mathbb{M}_1$ pelo custo de 5 XORs e 1

Algorithm 2 Cálculo de $D \cdot a$

ENTRADA: $a \in \mathbb{M}_1$.

SAÍDA: $D \cdot a$.

```

1:  $v \leftarrow \text{xtimes}(a_0 \oplus a_1 \oplus a_2), w \leftarrow \text{xtimes}(v)$ 
2:  $b_0 \leftarrow a_0 \oplus v, b_1 \leftarrow a_1 \oplus w, b_2 \leftarrow a_2 \oplus v \oplus w$ 
3: return  $b$ 

```

operação de `ctimes`, ou 7 XORs e 4 operações de `xtimes`. Ele também calcula $E^{-1} \cdot a$ ao custo de um XOR extra.

Algorithm 3 Cálculo de $E \cdot a$ or $E^{-1} \cdot a$

ENTRADA: $a \in \mathbb{M}_1$.

ENTRADA: e , um flag Booleano indicando se $E \cdot a$ (**true**) ou $E^{-1} \cdot a$ (**false**) deve ser calculado.

```

1:  $v \leftarrow a_0 \oplus a_1 \oplus a_2$ 
2: if  $e$  then
3:    $v \leftarrow \text{ctimes}(v)$ 
4: else
5:    $v \leftarrow \text{ctimes}(v) \oplus v$ 
6: end if
7:  $b_0 \leftarrow a_0 \oplus v, b_1 \leftarrow a_1 \oplus v, b_2 \leftarrow a_2 \oplus v$ 
8: return  $b$ 

```

Dispositivos com processadores mais poderosos (de 32 bits, por exemplo) podem implementar as transformações $\sigma[k]$, θ , ψ_s e Υ_s de maneira mais eficiente. A adição de chaves $\sigma[k]$ pode ser feita coluna a coluna, exigindo apenas 4 operações de XOR ao invés de 12. Por outro lado, o resultado das operações θ e ψ_s pode ser armazenado em uma tabela que implementa a multiplicação pelas matrizes D e E . Uma outra tabela pode, ainda, conter o resultado da combinação da S-Box com a multiplicação por D , executando de uma só vez as operações γ e θ . A operação Υ_s também pode operar sobre grupos de alguns bytes, facilitando a operação de deslocamento; além disto, a multiplicação por x^8 pode ser implementada usando uma única tabela contendo o resultado de T_0 e T_1 , acelerando a redução polinomial. Adicionalmente, as sub-chaves podem ser pré-calculadas e armazenadas em uma tabela para pronto uso, aumentando a velocidade do processo de encriptação/decriptação de diversos blocos sob uma mesma chave.

7.5.1 Considerações preliminares

Uma característica importante do CURUPIRA é que sua função de round utiliza apenas operações bastante simples: XOR bit-a-bit, deslocamento de bits e buscas em tabelas. Desta forma, esta cifra evita, por projeto, operações algébricas complexas que poderiam levar a uma redução de seu desempenho em algumas plataformas.

A seguir encontra-se uma comparação inicial entre a eficiência do CURUPIRA e o Skipjack. Estas considerações preliminares, que servem para mostrar o potencial a cifra, serão complementadas pelos resultados experimentais apresentados no Capítulo 8.

O custo de encriptação no CURUPIRA com R rounds é de $3R - 1$ XORs, $3(R - 1)/2$ aplicações de `xtimes` e R buscas na S-Box por byte, sem contar os custos de escalonamento das chaves. Por exemplo, o CURUPIRA de 10 rounds com uma chave de 96 bits requer 29 XORs, 6 aplicações de `xtimes` e 10 buscas na S-Box por byte encriptado. Comparativamente, o Skipjack necessita de 48 XORs e 16 buscas na sua *F-Table* (uma tabela de substituição de 256 bytes), sem contar os incrementos de contador e as atualizações de índice de chaves durante o processo de escalonamento. Portanto, assumindo que o custo das operações básicas é aproximadamente o mesmo para cada byte em ambas as cifras, pode-se estimar que o CURUPIRA tem um custo de $45/64 \approx 70\%$ daquele apresentado pelo Skipjack.

O escalonamento de chaves do CURUPIRA-1 é assumidamente mais complexo do que aquele apresentado na maioria das outras cifras. Todavia, seu custo mantém-se abaixo de $2/3$ buscas na S-Box, $1/3$ aplicações de `ctimes` e 2 XORs por byte da chave e por round, além de dificultar vários tipos de ataques, (em especial ataques de chaves relacionadas) e ter a vantagem de um comportamento cíclico. Já o escalonamento do CURUPIRA-2 é bastante leve, exigindo menos do que $1/2$ buscas na S-Box, $3/4$ XOR, bem como $1/12$ cálculos de T_0 e T_1 por byte da chave e por round.

É importante ressaltar que a adoção do CURUPIRA também permite uma notável

redução no consumo de energia e no tempo de processamento para a autenticação de mensagens caso um MAC padrão como o CBCMAC, que requer uma encriptação completa por bloco de dados autenticado, seja substituído por um MAC da família AL-RED como o Pelican (DAEMEN; RIJMEN, 2005b), que requer apenas $4/R$ encriptações por bloco mais 2 encriptações completas para a mensagem inteira.

7.6 Resumo

Este capítulo apresentou o CURUPIRA, uma cifra de bloco de tamanho de chave variável desenvolvida especialmente para plataformas com recursos restritos, como redes de sensores. Desta forma, o algoritmo apresenta características bastante interessantes neste contexto, como uma estrutura involutiva e escalonamento cíclico de chaves.

O CURUPIRA foi desenvolvido conforme a Estratégia de Trilha Larga, apresentada no Capítulo 5. Aproveitando os conceitos desenvolvidos no citado capítulo e também no Capítulo 4, foi desenvolvida uma análise de sua segurança, que mostra a grande robustez da cifra às diversas técnicas de criptoanálise conhecidas atualmente. Esta é uma grande vantagem em relação a algumas cifras amplamente utilizadas em redes de sensores, como o Skipjack.

Ao mesmo tempo, este capítulo mostrou que o CURUPIRA apresenta grande flexibilidade de implementação. Suas características de projeto permitem, então, sua otimização em termos de ocupação de memória e velocidade de processamento de acordo com os recursos disponíveis na plataforma alvo. Assim, a cifra se propõe a atingir um elevado desempenho tanto quando se consideram máquinas poderosas (como computadores modernos) quanto dispositivos com recursos restritos. Esta característica será posta à prova no próximo capítulo, que mostra o resultado da análise de desempenho da cifra, comparando-a com outras cifras de importância neste contexto.

8 TESTES E COMPARAÇÃO COM OUTRAS CIFRAS

No capítulo anterior, foi dada uma descrição detalhada do CURUPIRA, incluindo uma discussão sobre formas de implementação de cada um dos seus blocos construtivos. Neste capítulo, a eficiência do algoritmo será posta à prova, tanto em plataformas com recursos restritos como em dispositivos poderosos. Sua ocupação de memória e velocidade de operação serão confrontados com aquelas do Skipjack e do AES.. O Skipjack foi escolhido em razão de sua ampla utilização em redes de sensores, tanto em conjunto com o TinySec (cf. Capítulo 6) quanto em outros protocolos como o Senssec (LI et al., 2005), o TinyKeyMan (LIU; NING; LI, 2003) e o MiniSec (LUK et al., 2007). Apesar de apresentar uma reduzida margem de segurança, seu sucesso deve-se ao bom desempenho apresentado em plataformas com recursos restritos, conforme resultados apresentados em (LAW; DOUMEN; HARTEL, 2006) e confirmados neste documento. Já a escolha do AES explica-se pela sua ampla utilização nas mais diversas aplicações e plataformas. Além disto, existem diversas implementações eficientes deste algoritmo em domínio público, o que facilita o desenvolvimento. Todos os algoritmos testados apresentam interfaces semelhantes, de modo a não influenciar na comparação.

As seguintes plataformas foram escolhidas como base para os testes:

- Plataforma de 8 bits: Microcontrolador RISC da série PIC18F8490 (MICROCHIP, 2006), com processador de 8MHz, 16KB de memória Flash e 768 bytes de RAM.
- Simulador de Sensor: Avrora versão 1.6.0 - Beta (TITZER; LEE; PALSBERG,

2004), simulando microcontrolador da série ATmega128 (ATMEL, 2007).

- Plataforma de 32 bits: Computador portátil Dell Inspiron 9100 equipado com processador Pentium 4 (3.2GHz) e 1GB de memória RAM.

O simulador Avrora foi escolhido pela sua facilidade de uso, permitindo uma segunda análise dos algoritmos testados com o microcontrolador PIC18F8490, interessante por apresentar capacidade pouco superior àquela do ATmega8535 (ATMEL, 2006). Este último conta com um processador de 4 MHz, 8 KB de memória Flash, 512 bytes de RAM e foi usado no projeto Smart Dust (HILL et al., 2000) (voltado a redes de sensores), donde o interesse da comparação.

8.1 Características das implementações

Durante o desenvolvimento, o maior foco foi dado à velocidade de operação de todas as cifras, já que o número de operações realizadas tem efeito direto sobre o consumo de energia dos dispositivos. Foi dada bastante atenção à utilização de memória (em especial memória RAM) mas, a não ser que a expansão do código se tornasse excessiva, em diversas ocasiões foram escolhidas estruturas mais rápidas em detrimento daquelas mais compactas, abordagem válida para todas as cifras consideradas.

As mesmas implementações em linguagem C foram testadas no simulador Avrora e no microcontrolador. Para o Avrora, foram utilizados o compilador `avr-gcc` e o *disassembler* `avr-objdump` (ambas ferramentas do pacote GNU) conforme recomendado pela sua documentação. Dentre as opções de otimização do compilador, foi escolhido o nível 2 (i.e. comando “`avr-gcc -O2`”), já que o mesmo apresentou os melhores resultados para todas as cifras testadas, levando a uma considerável compressão de código sem prejuízo de desempenho. Para o PIC, foi utilizado o MPLAB IDE v7.60, em conjunto com o compilador MPLAB C18. Este compilador fornece alguns recursos avançados de otimização, denominados *procedural abstraction* (agrupamento de por-

ções similares de código em funções, para compressão de código) e uso de *extended mode* (expansão do conjunto de instruções e modo de endereçamento adicional, funcionalidades não suportadas por alguns dispositivos).

Os algoritmos inicialmente desenvolvidos para plataformas de 8 bits baseiam-se fortemente no uso de matrizes e ponteiros. Contudo, durante os testes com os compiladores, observou-se que a velocidade de operações sobre estes tipos de variáveis é bem menor do que a operação sobre variáveis básicas, fato facilmente verificado quando se analisa o código em *Assembly* gerado pelas ferramentas. Por exemplo, enquanto uma operação de atribuição entre duas variáveis v_0 e v_1 do tipo *char* (i.e. $v_0 = v_1$) corresponde a uma única instrução, a mesma atribuição usando ponteiros para *char* p_0 e p_1 (i.e. $*p_0 = *p_1$) desdobra-se em mais de 15 instruções. Esta observação permitiu algumas otimizações importantes, em especial do CURUPIRA e do Skipjack, que tiveram sua velocidade expressivamente aumentada ao custo de uma pequena expansão no tamanho do código (já que alguns laços foram removidos). A razão pelo qual a mesma abordagem não foi intensamente utilizada no AES foi a acentuada expansão do código resultante, bem como sua modesta efetividade quando comparada ao algoritmo que usa chaves pré-calculadas. Portanto, como as velocidades obtidas com estas duas abordagens de programação foram bastante distintas, elas serão tratadas separadamente para garantir uma comparação justa.

Os detalhes de implementação de cada cifra testada são dados a seguir. É importante ressaltar que, para não estender desnecessariamente a discussão, apenas as versões mais relevantes serão analisadas com detalhes ao longo deste capítulo.

AES (8 bits): foi adotada a versão desenvolvida por Gustavo Martins (MARTINS, 2007), cujo código inclui o uso de matrizes e efetua o pré-cálculo das sub-chaves de round, garantindo uma velocidade bastante considerável para ambas as operações de encriptação e decríptação. São também utilizadas tabelas de 256 bytes, uma para o cálculo da S-Box e a outra para sua inversa. É importante ressaltar

que esta versão, doravante denominada AES_{128} , opera apenas com chaves de 128 bits. Isto permite a geração de um algoritmo mais compacto, principalmente no que diz respeito ao código do escalonamento de chaves. No caso do PIC, ambas as otimizações do compilador descritas anteriormente foram usadas, permitindo tanto uma redução na ocupação de memória quanto um aumento na velocidade da cifra.

CURUPIRA (8 bits): foram testadas diversas combinações das otimizações possíveis para o CURUPIRA, de modo a verificar a eficácia de cada uma delas:

1. Uma primeira variação refere-se ao número de tabelas utilizadas: ‘S’ refere-se à S-Box; ‘X’ refere-se à operação $xtimes$; ‘C’ refere-se à operação $ctimes$; ‘0’ e ‘1’ referem-se, respectivamente, às tabelas T_0 e T_1 da versão 2 do escalonamento de chaves. Todas estas tabelas ocupam 256 bytes, com exceção de T_0 , que ocupa apenas 32 bytes pois implementa apenas a operação $T'_0[U] = (U \ll 2) \oplus U$ e tem sua entrada inicialmente deslocada de 3 bits para a esquerda.
2. Uma segunda variação diz respeito à flexibilidade do tamanho de chave. O $CURUPIRA_c$ corresponde ao algoritmo completo, que aceita chaves de todos os tamanhos e, por esta razão, faz intenso uso de ponteiros e matrizes. Já a versão denominada $CURUPIRA_{96}$ aceita apenas chaves de 96 bits, permitindo otimizações como o mapeamento dos blocos de entrada em variáveis específicas da cifra, o que reduz bastante a necessidade destas representações de dados e laços para acessá-las.
3. Um terceiro fator levado em consideração são os recursos do MPLAB discutidos anteriormente. Para o $CURUPIRA_c$, apesar da considerável compactação da cifra (cerca de 25%), esta abordagem levou a uma redução de velocidade de até 45% e, por esta razão, apenas a versão que não utiliza estes recursos será considerada neste documento. Já o $CURUPIRA_{96}$ foi be-

neficiado pelos mesmos, dando origem a duas versões distintas da cifra: o *extended mode*, sozinho, permitiu a elevação da velocidade do algoritmo, enquanto seu uso em conjunto com a *procedural abstraction* levou à versão mais compacta da cifra. As cifras resultantes foram denominadas CURUPIRA₉₆^V e CURUPIRA₉₆^M, respectivamente. É importante notar que a simulação no Avrora não inclui otimizações distintas do compilador e, portanto, as versões mais rápida e veloz se confundem.

4. Finalmente, com o intuito de avaliar a complexidade do escalonamento de chaves, foram implementados não somente os escalonamentos 1 e 2, mas também versões da cifra nas quais esta operação foi removida, identificadas como CURUPIRA-0. Deste modo, a velocidade resultante é semelhante àquela que seria obtida com chaves pré-calculadas e elimina-se a influência do uso de tabelas para as operações $ctimes$, T_0 e T_1 .

Skipjack (8 bits): a implementação adotada busca grande velocidade de processamento sem elevar demasiadamente o consumo de memória RAM, de modo que não há expansão das chaves de round. Como a cifra opera com um único tamanho de chave, foram utilizadas otimizações de código semelhantes às presentes no CURUPIRA₉₆. Assim, com os mesmos recursos do compilador, foram obtidos o Skipjack^V e o Skipjack^M, as versões mais rápida e mais compacta da cifra, respectivamente. Para efeitos de comparação com o CURUPIRA_c, entretanto, também foi implementada uma versão do Skipjack que opera sobre ponteiros e matrizes. Esta versão foi denominada Skipjack_c em razão de sua semelhança, em termos de estrutura de código, com a versão completa da cifra proposta.

Já os algoritmos voltados à plataforma de 32 bits foram implementados em Java (JDK 1.5) usando o Netbeans IDE versão 5.0. Aproveitando a disponibilidade de códigos bastante otimizados tanto do AES quanto do Skipjack, foram utilizados algoritmos colocados em domínio público como base para os testes. O CURUPIRA, por sua vez, foi

desenvolvido segundo a especificação, adotando as otimizações propostas para plataformas com recursos abundantes. Mais especificamente:

AES (32 bits): foi adotada uma versão implementada por Paulo Barreto (BARRETO, 2003), a qual opera sobre colunas de 32 bits (ao invés de byte a byte) em cada bloco, acelerando os cálculos. As principais operações envolvidas no algoritmo são implementadas por meio de tabelas contendo resultados pré-calculados, totalizando 10 tabelas com 256 palavras de 32 bits cada uma. Além disto, as chaves de round são todas pré-calculadas na inicialização da cifra, ficando armazenadas em um vetor para fácil acesso.

CURUPIRA (32 bits): em sua maioria, as transformações que compõem o algoritmo atuam diretamente sobre colunas de 24 bits. Assim como o AES, esta versão pré-calcula as sub-chaves de round e as armazena em uma tabela. Adicionalmente, são utilizadas tabelas auxiliares que implementam a multiplicação pelas matrizes D e E , bem como para a combinação das operações θ e γ . Desta forma, a cifra usa exatamente o mesmo número de tabelas que o AES.

Skipjack (32 bits): foi adotada uma versão originalmente desenvolvida em linguagem C por Panu Rissanen e posteriormente otimizada por Mark Tillotson e Paulo Barreto (BARRETO, 1998), a qual foi adaptada para a linguagem Java durante este trabalho. Esta versão opera sobre palavras de 16 bits e, na inicialização, pré-calcula o resultado da *F-Table* combinada com a aplicação de chave, armazenando-os em uma matriz de dimensões 10 x 256.

8.2 Resultados obtidos: 8 bits

Como a maior parte dos testes com os algoritmos foi realizada usando o PIC18F8490, ele parece ser o melhor candidato para dar início às discussões. A velocidade de processamento foi medida usando rotinas de temporização do próprio

microcontrolador, apresentando-se na forma de pulsos de relógio. Deste modo, um tempo de processamento de 1000 corresponde a 1000 pulsos, ou a 1 segundo quando a velocidade do dispositivo é configurada para 1kHz.

A Tabela 4 mostra a ocupação de memória das diversas versões testadas do CURUPIRA, Skipjack e AES₁₂₈. Na memória ROM do dispositivos, são armazenadas apenas constantes, enquanto o algoritmo propriamente dito fica na memória de Código. Assim, a ocupação de memória ROM é proporcional ao número de tabelas. É interessante notar a influência da versão do escalonamento de chaves adotada, já que o CURUPIRA_c-2 mostra-se consideravelmente mais compacto do que o CURUPIRA_c-1, enquanto o inverso ocorre com o CURUPIRA₉₆ (em razão da remoção de laços no código).

Tabela 4: Ocupação de memória (em bytes) do CURUPIRA, Skipjack e AES₁₂₈ no PIC18F8490

Algoritmo	Escalonamento: 1			Escalonamento: 2		
	<i>Tabelas</i>	<i>ROM</i>	<i>Código</i>	<i>Tabelas</i>	<i>ROM</i>	<i>Código</i>
CURUPIRA _c	SXC	822	1444	SX01	800	1192
	SX	566	1490	SX0	544	1236
	SC	566	1462	SX1	768	1194
	S	310	1480	SX	512	1238
	XC	566	1608			
	X	310	1654			
	C	310	1628			
	-	54	1646			
CURUPIRA ₉₆ ^V	SXC	768	1372	SX01	800	1464
	SX	512	1444	SX0	544	1530
	SC	512	1350	SX1	768	1466
	S	256	1390	SX	512	1532
CURUPIRA ₉₆ ^M	SXC	768	1056	SX01	800	1102
	SX	512	1076	SX0	544	1190
	SC	512	1072	SX1	768	1104
	S	256	1092	SX	512	1192
Skipjack _c		256	1012			
Skipjack ^V		256	972			
Skipjack ^M		256	722			
AES ₁₂₈		512	1024			

Primeiramente, é necessário comparar as versões dos algoritmos que utilizam matrizes e ponteiros. Percebe-se que o espaço ocupado pelo CURUPIRA_c vai de 1700 a 2266

bytes, superiores aos 1268 bytes ocupados pelo Skipjack_c e os 1536 bytes ocupados pelo AES_{128} . É importante notar, entretanto, que o CURUPIRA_c suporta três tamanhos distintos de chave, enquanto os outros algoritmos não o fazem. Além disto, esta diferença de tamanho é compensada pela velocidade apresentada por esta versão da cifra proposta, como será mostrado adiante, o que se traduz em um reduzido consumo de energia.

Já a comparação entre o $\text{CURUPIRA}_{96}^{V,M}$ e o $\text{Skipjack}^{V,M}$ mostra, novamente, que este último se mantém mais compacto do que a cifra proposta. Como, em cada round, o algoritmo do CURUPIRA_{96} opera explicitamente sobre 24 variáveis distintas (12 para os bytes do bloco + 12 para os bytes da chave) e o Skipjack atua sobre apenas 18 (8 para o bloco e 10 para a chave), este resultado já era previsto. Além disto, o fato de o Skipjack não apresentar um algoritmo escalonamento de chaves permite um grau de compactação difícil de ser superado por uma cifra que adote uma solução de escalonamento provendo maior segurança.

Finalmente, apesar de não ter sido possível medir diretamente o uso de memória RAM, é de se esperar que a operação do CURUPIRA e do Skipjack exija uma quantidade de memória RAM bem menor do que no AES_{128} , já que este último armazena chaves pré-calculadas, enquanto os dois primeiros não o fazem. Já quando se comparam Skipjack e CURUPIRA, o esperado é que a cifra proposta consuma uma quantidade de memória RAM ligeiramente superior, devido aos maiores tamanhos de chave e bloco.

Tendo avaliado a ocupação de memória das cifras, pode-se então passar à sua análise de desempenho. A Figura 19 mostra a velocidade de encriptação do CURUPIRA_c-1 , considerando todos os tamanhos de chave e número de rounds permitidos. As curvas do processo de deciptação são semelhantes, apesar desta operação ter se mostrado de 2% a 6% mais lenta do que a encriptação para cada tamanho de chave e versão testada. A análise do gráfico mostra que o algoritmo para cálculo da S-Box é, de longe, o mais interessante de ser implementado na forma de uma tabela: isto leva a uma considerável

redução no número de operações da cifra sem grande impacto no tamanho do código, já que a quantidade total de memória (de Código) ocupada pelo Algoritmo 1, responsável pelo cálculo da S-Box (cf. seção 7.3.1.1), é semelhante ao espaço ocupado (na memória ROM) pela tabela que o substitui. Nota-se ainda que, em termos de desempenho, é interessante implementar *ctimes* e *xtimes*, nesta ordem de prioridade, na forma de tabelas. Apesar da maior velocidade envolvida, as curvas correspondentes ao $CURUPIRA_c-2$ apresentam comportamento análogo, mantendo também a relação entre as velocidades de encriptação e de decríptação. No entanto, os testes mostraram que a presença ou ausência das tabelas T_0 e T_1 tem efeito pouco significativo sobre sua velocidade final.

Figura 19: Tempo de encriptação de um único bloco de dados, em função do número de rounds, para o $CURUPIRA_c-1$ no PIC18F8490. São usados diferentes tamanhos de chave no gráfico: 96 bits para 10 e 11 rounds; 144 bits para 14 a 17 rounds; 192 bits para 18 a 23 rounds.

Baseado nestes resultados, foram eleitas as seguintes versões do $CURUPIRA$ para comparação com o desempenho do AES e do Skipjack:

$CURUPIRA_c-1$: Versões com as tabelas

1. SXC: é aquela que apresenta melhor desempenho, apesar de ocupar maior espaço na memória ($822+1444=2266$ bytes);
2. SC: apresenta uma boa relação memória/desempenho, ocupando cerca de 89% (2028 bytes) da memória usada pela versão SXC, contrabalançado pelo seu tempo de processamento $\approx 6\%$ superior a esta última;
3. S: versão que utiliza a menor quantidade de memória (1790 bytes, cerca de 22% a menos do que a versão SXC), apesar disto acarretar em um maior tempo de processamento ($\approx 30\%$ superior ao SXC).

CURUPIRA_c-2: Apenas a versão SX será considerada, já que ela está entre as versões mais rápidas e, ainda assim, mostra-se bastante compacta.

CURUPIRA₉₆-1,2: Para cada algoritmo de escalonamento, as mesmas versões descritas acima foram consideradas. Adicionalmente, distinguem-se as versões que ficaram mais compactas (M) e velozes (V) em função dos recursos do compilador.

CURUPIRA-0: Todas as versões testadas foram comparadas a suas equivalentes quando as chaves de round são pré-calculadas.

A Figura 20 mostra o tempo de encriptação, por byte, para o CURUPIRA_c, o AES₁₂₈ e o Skipjack_c. Observa-se que, enquanto estas duas últimas cifras têm desempenho semelhante, o CURUPIRA_c com chaves de 96 bits é consideravelmente mais rápido, tanto quando suas sub-chaves são pré-calculadas (até $\approx 60\%$ mais rápido) quanto quando elas são geradas no momento da utilização com o escalonamento 1 (até $\approx 20\%$ mais rápido) ou 2 ($\approx 45\%$ mais rápido). Além disto, para tamanhos de chave maiores, tanto o CURUPIRA_c-0 quanto o CURUPIRA_c-2 mostram-se mais rápidos do que as outras cifras consideradas. Apesar de bastante positivo, este resultado deve ser avaliado com cautela, já que o tempo medido inclui não somente as operações que compõem a cifra mas também diversas operações auxiliares, como a atualização de índices. Portanto,

Figura 20: Comparação entre as velocidades de encriptação, por byte, do CURUPIRA_c, AES₁₂₈ e Skipjack_c no PIC18F8490.

a comparação entre as versões que atuam explicitamente sobre cada variável é importante para verificar sua consistência.

A Figura 21 mostra o tempo de encriptação por byte, para o CURUPIRA₉₆^{V,M} e o Skipjack^{V,M}. A redução da influência de operações auxiliares mostra-se mais favorável ao Skipjack do que ao CURUPIRA mas, ainda assim, tanto o CURUPIRA₉₆-0 quanto o CURUPIRA₉₆-2 continuam ao menos 20% mais rápidos do que o Skipjack. Já o CURUPIRA₉₆^{V,M}-1 (SXC), mostra-se cerca de 13% mais lento do que o Skipjack^{V,M} em razão do escalonamento de chaves mais complexo (e seguro).

Por fim, é interessante notar que os cálculos preliminares da seção 7.5.1 não estão muito longe do resultado experimental, já que o tempo de encriptação do CURUPIRA₉₆^V-0 é cerca de 70% do tempo do Skipjack^V, enquanto esta relação se reduz a 20% quando se consideram o CURUPIRA₉₆^M-0 e o Skipjack^M.

Figura 21: Comparação entre as velocidades de encriptação, por byte, do $CURUPIRA_{96}^{V,M}$ e $Skipjack^{V,M}$ no PIC18F8490.

8.3 Resultados obtidos: Simulador de Sensor

Para verificar a influência da plataforma de testes sobre os resultados mostrados na seção anterior, a seguir são apresentados os resultados obtidos com o simulador Avrora, que simula outra plataforma de 8 bits. A memória ocupada pelos algoritmos considerados mais relevantes para a discussão é mostrada na Tabela 5.

Tabela 5: Ocupação de memória, em bytes, das cifras testadas no Simulador de Sensor.

Algoritmo	Escalonamento: 1			Escalonamento: 2		
	Versão	ROM	Código	Versão	ROM	Código
$CURUPIRA_c$	SXC	822	1648	SX01	800	1676
	SX	566	1684	SX0	544	1698
	SC	566	1672	SX1	768	1696
	S	310	1696	SX	512	1718
$CURUPIRA_{96}^V \equiv CURUPIRA_{96}^M$	SXC	768	1936	SX01	800	1806
	SX	512	1982	SX0	544	1826
	SC	512	1986	SX1	768	1826
	S	256	2016	SX	512	1846
$Skipjack_c$		256	940			
$Skipjack^V \equiv Skipjack^M$		256	1352			
AES_{128}		512	1406			

A análise desta tabela mostra um resultado salutar: dentre os algoritmos que utilizam matrizes e ponteiros, o $Skipjack_c$ é o único que apresentou redução no tamanho de código quando comparado ao resultado da compilação com o MPLAB. De qual-

quer forma, os resultados obtidos para o simulador confirmam aqueles apresentados no microcontrolador: o tamanho do código de todas as versões do CURUPIRA testadas continua superior ao do Skipjack e do AES.

Figura 22: Comparação entre as velocidades de encriptação, por byte, do CURUPIRA_c, AES₁₂₈ e Skipjack_c no Simulador de Sensor.

Com relação a desempenho, conforme mostrado na Figura 22, as velocidades do CURUPIRA_c e do Skipjack_c foram um pouco abaixo daquelas apresentadas no PIC, enquanto o AES₁₂₈ mostrou-se consideravelmente mais rápido do que anteriormente. Desta forma, enquanto as relações entre os tempos medidos para as duas primeiras cifras se mantiveram próximas àquelas encontrada para o PIC18F8490, o AES mostrou-se mais rápido do que o CURUPIRA_c-1, apesar de manter-se mais lento do que o CURUPIRA_c-0 e o CURUPIRA_c-2.

Já com relação aos testes das versões com reduzido uso de ponteiros e matrizes, os resultados foram um pouco menos favoráveis para o CURUPIRA₉₆: o mesmo código testado no microcontrolador apresentou um desempenho inferior quando testado no

simulador Avrrora, o contrário acontecendo com o Skipjack, conforme mostram os gráficos da Figura 23. Assim, o processamento de um byte com o CURUPIRA₉₆-0 foi apenas de 5% a 15% mais rápido do que o mesmo com o Skipjack^{V,M}. O desempenho do CURUPIRA₉₆-2, por sua vez, mostrou-se ligeiramente inferior ao desta última cifra.

Figura 23: Comparação entre as velocidades de encriptação, por byte, do CURUPIRA₉₆^{V,M} e Skipjack^{V,M} no Simulador de Sensor.

Apesar do possível questionamento sobre a influência do próprio simulador – que, conforme especificação, encontra-se em versão Beta – sobre os resultados obtidos, estes dados mostram que, apesar de poder variar de dispositivo para dispositivo, a velocidade da cifra proposta (especialmente quando se considera a versão 2 do escalonamento de chaves) é semelhante e potencialmente superior àquela do Skipjack.

8.4 Resultados obtidos: 32 bits

Os tempos de processamento na plataforma de 32 bits foram medidos usando o relógio do próprio sistema, para repetidas execuções de cada operação em questão: encriptação, decríptação e escalonamento de chaves. Para ambas as operações e cada tamanho de chave, o menor número de rounds do CURUPIRA foi adotado.

Antes de passar à comparação propriamente dita, entretanto, foi avaliada a relação entre o número de rounds do CURUPIRA e o tempo necessário para as operações de encriptação/decriptação (que usam exatamente o mesmo algoritmo) e escalonamento de chaves. Pelo gráfico na Figura 24, percebe-se que, nesta plataforma, a operação de escalonamento toma consideravelmente mais tempo para ser concluída do que a encriptação/decriptação de um único bloco, tanto para o CURUPIRA-1 quanto para o CURUPIRA-2. Isto pode ser considerado um ponto positivo, já que o cálculo das sub-chaves precisa ser feito uma única vez, na inicialização da cifra com uma determinada chave.

Figura 24: Desempenho em função do número de rounds adotado para o CURUPIRA na plataforma de 32 bits, normalizados para a escala 0 - 5000. Repare que são usados diferentes tamanhos de chave no gráfico: 96 bits para 10 e 11 rounds; 144 bits para 14 a 17 rounds; 192 bits para 18 a 23 rounds.

Os resultados da cifra proposta podem, então, ser comparados ao desempenho AES e ao Skipjack nesta plataforma. A Figura 25 mostra as curvas dos processos de encriptação e decriptação de cada uma destas cifras, para todos os tamanhos de chave permitidos (repare que, como o Skipjack usa apenas chaves de 80 bits, o mesmo

tempo de processamento é repetido 3 vezes). É interessante notar que, apesar de o CURUPIRA ser a única cifra involutiva dentre as analisadas, os testes mostram que todas elas apresentam uma velocidade de encriptação semelhante à de decifração. Por esta razão, para todas as cifras, as curvas destas duas operações se confundem. Apenas para efeitos de comparação, os gráficos mostram uma entrada adicional, denominada CURUPIRA', que corresponde ao CURUPIRA usando o mesmo número de rounds do AES.

Figura 25: Comparação entre as velocidades de encriptação/decifração do CURUPIRA, AES e Skipjack na plataforma de 32 bits. Para cada tamanho de chave, o menor número de rounds permitido foi usado. O CURUPIRA' corresponde ao CURUPIRA com o mesmo número de rounds do AES. Resultados normalizados para a escala 0 - 5000.

Os gráficos mostram que o AES é invariavelmente mais rápido do que o CURUPIRA e o Skipjack nesta plataforma, para todos os tamanhos de chave. Entretanto, percebe-se que a diferença com a cifra proposta deve-se basicamente ao número de rounds adotados, já que a curva correspondente ao CURUPIRA' é praticamente equivalente àquela do AES. Além disto, conforme discutido na seção 6.3.2, o tamanho de bloco do AES é considerado muito grande para redes de sensores, que costumam trafegar mensagens pequenas. Neste caso, o reduzido tamanho de bloco adotado pelo CURUPIRA pode

compensar, com vantagens, esta pequena diferença de velocidades.

O modesto resultado do Skipjack (quase de 3 vezes mais lento do que o CURUPIRA) pode parecer estranho à primeira vista, já que o desempenho desta cifra costuma ser a principal razão para sua adoção em redes de sensores. No entanto, isto se explica pela dificuldade de implementá-lo de modo a utilizar eficientemente os 32 bits disponíveis nesta plataforma, posto que suas operações são orientadas a palavras de 16 bits. Já o CURUPIRA e o AES conseguem fazê-lo de forma razoavelmente simples, pois suas transformações, apesar de orientadas a bytes, podem ser facilmente adaptadas para operar sobre 32 e 24 bits (colunas), respectivamente.

Figura 26: Comparação entre as velocidades de escalonamento de chaves do CURUPIRA, AES e Skipjack na plataforma de 32 bits. Para cada tamanho de chave, o menores número de rounds permitido foi usado. Resultados normalizados para a escala 0 - 5000.

Já o gráfico na Figura 26 mostra o desempenho das 3 cifras com relação ao escalonamento de chaves. O “escalonamento de chaves” do Skipjack, incluído nesta figura, corresponde aos pré-cálculos efetuados na inicialização da cifra, conforme explicado na seção 8.1).

De acordo com o gráfico, as operações realizadas na inicialização do Skipjack são cerca de 10 vezes mais demoradas do que ocorre com as outras cifras. Este fato é explicável pela quantidade de informação gerada, correspondente a uma matriz de dimensão 10×256 , enquanto o AES e o CURUPIRA geram vetores de $4(N + 1)$ posições, onde N corresponde ao número de rounds do escalonamento. Por outro lado, o complexo algoritmo envolvido no escalonamento de chaves do CURUPIRA-1, bem como a diferença no número de rounds adotado pela cifra e o truncamento da sub-chave para o tamanho do bloco, fazem com que seu desempenho seja de 25% a 115% inferior àquele do AES (para o menor e o maior tamanho de chaves, respectivamente). A curva correspondente ao CURUPIRA-2 mostra, contudo, que o algoritmo da segunda versão do escalonamento de chaves supera estas dificuldades, chegando a apresentar desempenho superior àquele obtido pelo escalonamento do AES. De qualquer forma, como o cálculo das chaves de round precisa ser feito apenas uma vez, na inicialização das cifras, a sua influência é reduzida em cenários nos quais não haja troca frequente de chaves.

8.5 Considerações finais sobre desempenho

Os testes com as diversas versões do CURUPIRA permitem concluir que a cifra proposta é bastante compacta e, principalmente, eficiente em plataformas com recursos restritos. Enquanto o AES é muitas vezes recomendado para dispositivos apresentam boa disponibilidade de memória e com aplicações que exigem segurança (LAW; DOUMEN; HARTEL, 2006, seção 5), nestas situações o CURUPIRA pode ser utilizado com vantagens, garantindo melhor desempenho e um nível de segurança considerável. Por outro lado, quando há pouca memória disponível, o Skipjack costuma ser adotado, já que realmente esta cifra apresenta bom desempenho e seu código é bastante compacto. Todavia, de acordo com os testes realizados, a cifra proposta apresenta-se como uma excelente alternativa: o CURUPIRA-1 é capaz de elevar consideravelmente o nível de segurança nestas situações, introduzindo um reduzido impacto em termos de desem-

penho e memória (em especial, memória RAM); já o CURUPIRA-2 é bastante seguro, especialmente em contextos pouco sujeitos a ataques de chaves relacionadas, e sua operação é potencialmente mais rápida do que aquela do Skipjack, resultando em uma cifra com baixíssimo consumo de energia. Além disto, caso haja espaço suficiente para o pré-cálculo de suas chaves, o uso do CURUPIRA torna-se ainda mais vantajoso para elevação do desempenho da encriptação dos dados e, conseqüentemente, para a redução do consumo de energia.

Os resultados para plataformas de 32 bits são igualmente animadores. O desempenho do CURUPIRA supera bastante o do Skipjack, tanto no que diz respeito à sua inicialização quanto aos processos de encriptação e deciptação. Além disto, o custo de encriptação/decriptação por byte e por round é semelhante àquele do AES, o que garante a equivalência das cifras quando os menores tamanhos de chave de ambas são adotados. Finalmente, apesar do escalonamento de chaves do CURUPIRA-1 ser assumidamente mais complexo e um pouco mais lento do que o do AES, o seu impacto é bastante reduzido nestas plataformas, já que normalmente as sub-chaves são todas pré-calculadas durante a inicialização da cifra.

9 CONCLUSÕES

O trabalho apresentado neste documento dá bases sólidas de alguns dos principais conceitos envolvidos no domínio da Criptografia, apresentados de forma sucinta nos Capítulos 2 e 3. Estes conceitos, aliados ao estudo de técnicas construtivas de cifras de bloco clássicas e do estado da arte da criptoanálise mostrados no Capítulo 4 são de fundamental importância para o completo entendimento da Estratégia de Trilha Larga descrita no Capítulo 5, uma metodologia de projeto robusta com relação à segurança de cifras de bloco. O próprio AES, construído de acordo com esta técnica, de sua adoção em 2001 até o presente momento resistiu muito bem à intensa análise de segurança perpetrada por diversos grupos de pesquisa em todo o mundo. A adoção da Estratégia de Trilha Larga como metodologia de projeto, portanto, mostra-se bastante interessante quando se consideram as necessidades de segurança de redes com recursos limitados, como redes de sensores. O protocolo Tinysec descrito no Capítulo 6 foi desenvolvido para prover segurança a esta classe de redes e, portanto, seu estudo apresenta diversas considerações quanto às suas limitações inerentes.

É neste contexto que foi desenvolvida uma cifra capaz de competir, muitas vezes em vantagem, com as soluções existentes para redes de sensores: o CURUPIRA, descrito no Capítulo 7. De acordo com sua análise de segurança, esta cifra é tão resistente à criptoanálise quanto se pode esperar, dadas as limitações assumidas durante o seu projeto. Além disto, a adoção da Estratégia de Trilha Larga permite provar rigorosamente sua segurança contra algumas das mais importantes famílias de ataques criptoanalíticos. Não obstante, os testes de desempenho realizados mostram que a cifra resultante

é compacta e, principalmente, eficiente tanto em plataformas de 8 bits (como é o caso de diversos dispositivos pouco poderosos, como sensores) quanto de 32 bits (como servidores na rede).

É importante ressaltar ainda que o CURUPIRA se mostra bastante flexível em termos de implementação, permitindo um aumento no seu desempenho ao custo de um maior consumo de memória (pelo uso de tabelas com resultados pré-calculados, por exemplo) ou uma redução de seu tamanho de código em troca de uma menor velocidade de processamento. Além desta facilidade de adaptação às mais diversas plataformas, a cifra permite 3 tamanhos de chave distintos (96, 144 ou 192 bits) e um número variável de rounds, garantindo também uma boa flexibilidade da cifra com relação a segurança. Adicionalmente, são propostas duas versões para seu algoritmo de escalonamento de chaves, ambas adequadas a sistemas com restrição de recursos, mas voltados a contextos um pouco distintos: o CURUPIRA-1 é voltado a situações nas quais, para obter um elevado nível de segurança (inclusive contra ataques de chaves relacionadas), é aceitável um ligeiro sacrifício em termos de desempenho; já o CURUPIRA-2 é mais adequado a sistemas menos sensíveis a ataques de chaves relacionadas e nos quais o desempenho é o fator preponderante, algo bastante factível com um pouco de planejamento da rede e com o uso de algoritmos seguros para geração de chaves. Como vantagens adicionais, o CURUPIRA apresenta uma estrutura involutiva quanto um escalonamento de chaves facilmente inversível, bem como ciclicidade no caso do CURUPIRA-1, características importantes para superar as limitações das plataformas às quais a cifra se destina.

Em resumo, suas propriedades fazem com que o CURUPIRA seja altamente recomendável para aumentar o nível de segurança (e, potencialmente, o desempenho) em redes de sensores e redes com limitação de recursos. É pelo conjunto de suas características que o artigo que descreve o CURUPIRA-1 foi escolhido para receber um prêmio de Menção Honrosa na 25^a edição de um dos simpósios mais tradicionais e conceituados do Brasil na área de redes de computadores, o Simpósio Brasileiro de Redes de

Computadores e Sistemas Distribuídos (BARRETO; SIMPLICIO, 2007).

9.1 Trabalhos futuros

O CURUPIRA é um bom exemplo de cifra capaz de prover confidencialidade aos dados em plataformas com recursos limitados. Entretanto, redes que utilizam tais dispositivos necessitam também de outros serviços de segurança. Dois serviços são de especial importância: (1) a garantia da integridade e autenticidade das mensagens trocadas e (2) o gerenciamento de chaves.

Pode-se garantir a integridade e a autenticidade de uma mensagem por meio do cálculo de seu *MACs* – *Message Authentication Code*, ou “Código de Autenticação de Mensagem”. Este é um serviço incluso no próprio TinySec, tanto no modo *TinySec-AE* quanto no modo *TinySec-Auth* (cf. Capítulo 6), que especifica o uso de uma variável do CBC-MAC (BELLARE; KILIAN; ROGAWAY, 2000) na qual o tamanho da mensagem é encriptado e combinado, via operação de XOR, com o primeiro bloco do texto claro. Esta pequena alteração é necessária porque a especificação original do CBC-MAC não é segura quando o tamanho das mensagens trocadas é variável (BELLARE; KILIAN; ROGAWAY, 2000, Seção 5). Deste modo, desde que a cifra de bloco subjacente seja segura, o MAC calculado também o será.

Contudo, conforme brevemente discutido na seção 7.5.1, existem alternativas ao CBC-MAC bastante superiores em termos de desempenho, como os MACs que seguem a estrutura ALRED (DAEMEN; RIJMEN, 2005a). Semelhantemente ao CBC-MAC, esta estratégia também utiliza-se de porções de código de uma cifra de bloco adjacente, introduzindo um reduzido impacto em termos de ocupação de memória. Além disto, seu uso é particularmente interessante quando a cifra subjacente segue a Estratégia de Trilha Larga, como é o caso do CURUPIRA, devido à elevada velocidade de difusão envolvida. Assim, desde que a cifra subjacente seja eficiente e segura, o al-

goritmo de MAC construído desta forma também o será. Como exemplo de função que adota tais características de projeto, pode-se citar o Pelican (DAEMEN; RIJMEN, 2005b), que usa o Rijndael como cifra subjacente. Outra alternativa possível para o cálculo do MAC é a adoção de uma estratégia semelhante à do Cipher-State Mode (LABORATORIES, 2004) que, apesar de bastante rápida, deve ser analisada com cuidado devido à sua incompatibilidade com cifras involutivas, conforme alegação de seus idealizadores. Um exemplo que se utiliza deste modo de operação para autenticação de mensagens é o ManTicore (BEAVER et al., 2003). Em resumo, o que se busca é um esquema de cálculo de MAC que não apenas seja bastante leve para ser implementado em sensores, mas também flexível o suficiente para permitir implementações otimizadas em plataformas mais poderosas, como servidores na rede alvo.

Já o gerenciamento de chaves representa um desafio à parte. Em alguns esquemas bastante difundidos para segurança a redes de sensores, como o TinySec, a resposta para este problema é evitada assumindo que, devido à reduzida vida útil dos nós, as chaves poderiam ser determinadas no momento em que o dispositivo fosse introduzido na rede, permanecendo inalteradas. Entretanto, esta abordagem é uma fonte em potencial de problemas de segurança, conforme discutido em diversos trabalhos que analisam esta questão mais profundamente (CAMTEPE; YENER, 2005; MALAN; WELSH; SMITH, 2004; WACKER et al., 2005). Uma abordagem interessante é o uso de esquemas de encriptação assimétrica baseados em identidades (BONEH; FRANKLIN, 2001), conhecidos pelo seu elevado desempenho. Assim, evitam-se os altos custos envolvidos na utilização de PKIs (*Public-Key Infrastructures*). Um exemplo bastante ilustrativo desta estratégia é o TinyTate (OLIVEIRA et al., 2007) que, apesar do baixo nível de segurança, demonstra o grande potencial desta classe de algoritmos. Portanto, por meio da seleção de parâmetros e algoritmos para cálculo de curvas elípticas e emparelhamentos bi-lineares, os quais devem ser bem adaptados a redes de sensores, é possível conceber esquemas de gerenciamento de chaves ao mesmo tempo leves e seguros.

Os trabalhos futuros incluem, portanto, a análise destes e outros esquemas de cálculo de MAC e gerenciamento de chaves. Desta forma, o que se busca é a construção de algoritmos que sejam ao mesmo tempo seguros e bem adaptados a plataformas com recursos limitados.

REFERÊNCIAS

ATMEL. *AVR 8-Bit RISC processor - ATmega8535 (90LS8535)*. [S.l.], 2006. Disponível em: <http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf>.

_____. *AVR 8-Bit RISC processor - ATmega128 e ATmega128L*. [S.l.], 2007. Disponível em: <http://www.atmel.com/dyn/resources/prod_documents/2467S.pdf>.

BARRETO, P. *The Skipjack block cipher*. 1998.
[Http://planeta.terra.com.br/informatica/paulobarreto/skipjack32.zip](http://planeta.terra.com.br/informatica/paulobarreto/skipjack32.zip).

_____. *The AES block cipher (Rijndael)*. 2003.
[Http://planeta.terra.com.br/informatica/paulobarreto/JEAX.zip](http://planeta.terra.com.br/informatica/paulobarreto/JEAX.zip).

BARRETO, P.; SIMPLICIO, M. CURUPIRA, a block cipher for constrained platforms. In: *Anais do 25º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2007*. SBC, 2007. v. 1, p. 61–74. ISBN 85-766-9106-X. Disponível em: <<http://www.sbr2007.ufpa.br/anais/2007/ST02%20-%2001.pdf>>.

BARRETO, P. S. L. M.; RIJMEN, V. The ANUBIS block cipher. In: *First open NESSIE Workshop*. Leuven, Belgium: NESSIE Consortium, 2000.

_____. The KHAZAD legacy-level block cipher. In: *First open NESSIE Workshop*. Leuven, Belgium: NESSIE Consortium, 2000.

BEAVER, C. et al. *ManTiCore: Encryption with joint cipher-state authentication*. 2003. [Http://eprint.iacr.org/2003/154.ps](http://eprint.iacr.org/2003/154.ps).

BELLARE, M.; KILIAN, J.; ROGAWAY, P. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, Academic Press, Inc., v. 61, n. 3, p. 362–399, 2000.

BELLARE, M.; ROGAWAY, P. Random oracles are practical: A paradigm for designing efficient protocols. In: *ACM Conference on Computer and Communications Security*. Fairfax, USA: ACM Press, 1993. p. 62–73.

BELLOVIN, S.; BLAZE, M. *Cryptographic modes of operation for the internet*. 2001. Second NIST Workshop on Modes of Operation. Disponível em: <citeseer.ist.psu.edu/bellovin01cryptographic.html>.

BIHAM, E.; BIRYUKOV, A.; SHAMIR, A. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In: *Advances in Cryptology – Eurocrypt’99*. [S.l.]: Springer, 1999. (Lecture Notes in Computer Science, v. 1592), p. 55–64.

- BIHAM, E.; SHAMIR, A. Differential cryptanalysis of DES-like cryptosystems. In: *Crypto '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1991. p. 2–21. ISBN 3-540-54508-5.
- BIRYUKOV, A.; DECANNIÈRE, C. Block ciphers and systems of quadratic equations. In: *Fast Software Encryption – FSE'2003*. [S.l.]: Springer, 2003. (Lecture Notes in Computer Science, v. 2887), p. 274–289.
- BIRYUKOV, A.; WAGNER, D. Slide attacks. In: *Fast Software Encryption – FSE'99*. Rome, Italy: Springer, 1999. (Lecture Notes in Computer Science, v. 1636), p. 245–259.
- BLAZE, M.; SCHNEIER, B. The MacGuffin block cipher algorithm. *Lecture Notes in Computer Science*, v. 1008, p. 97–110, 1995. Disponível em: <citeseer.ist.psu.edu/blaze95macguffin.html>.
- BOLAND, H.; MOUSAVI, H. Security issues of the ieee 802.11b wireless LAN. *Canadian Conference on Electrical and Computer Engineering*, v. 1, p. 333–336, 2004. ISSN 0840-7789. Disponível em: <<http://ieeexplore.ieee.org/iel5/9317/29618/01345023.pdf>>.
- BONEH, D.; FRANKLIN, M. Identity-based encryption from the Weil pairing. *Lecture Notes in Computer Science*, v. 2139, p. 213–??, 2001. Disponível em: <citeseer.ist.psu.edu/boneh01identitybased.html>.
- CAMTEPE, S. A.; YENER, B. *Key distribution mechanisms for wireless sensor networks: a survey*. [S.l.], 2005.
- COURTOIS, N.; PIEPRZYK, J. Cryptanalysis of block ciphers with overdefined systems of equations. In: *Advances in Cryptology – Asiacrypt'2002*. [S.l.]: Springer, 2002. (Lecture Notes in Computer Science, v. 2501), p. 267–287.
- DAEMEN, J. *Cipher and hash function design strategies based on linear and differential cryptanalysis*. Tese (Doctoral dissertation) — Katholiek Universiteit Leuven, March 1995.
- DAEMEN, J.; KNUDSEN, L. R.; RIJMEN, V. The block cipher SQUARE. In: *Fast Software Encryption – FSE'97*. Haifa, Israel: Springer, 1997. (Lecture Notes in Computer Science, v. 1267), p. 149–165.
- DAEMEN, J. et al. The NOEKEON block cipher. In: *First open NESSIE Workshop*. Leuven, Belgium: NESSIE Consortium, 2000.
- DAEMEN, J.; RIJMEN, V. The block cipher bksq. In: *Smart Card Research and Applications – CARDIS'98*. [S.l.]: Springer, 1998. (Lecture Notes in Computer Science, v. 1820), p. 236–245.
- _____. The wide trail design strategy. *Lecture Notes in Computer Science*, v. 2260, p. 222–239, 2001. ISSN 0302-9743. Disponível em: <<http://link.springer-ny.com/link/service/series/0558/papers/2260/22600222.pdf>>.

_____. *The Design of Rijndael: AES – The Advanced Encryption Standard*. Heidelberg, Germany: Springer, 2002. ISBN 3-540-42580-2.

_____. A new MAC construction ALRED and a specific instance ALPHA-mac. In: *FSE*. [S.l.: s.n.], 2005. p. 1–17.

_____. *The Pelican MAC Function*. 2005. Cryptology ePrint Archive, Report 2005/088. Disponível em: <<http://eprint.iacr.org/2005/088>>.

DIFFIE, W.; HELLMAN, M. E. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22, n. 6, p. 644–654, 1976. Disponível em: <citeseer.ist.psu.edu/diffie76new.html>.

FEISTEL, H. Cryptography and computer privacy. *Scientific American*, v. 228, n. 5, p. 15–23, 1973.

FERGUSON, N. et al. Improved cryptanalysis of RIJNDAEL. In: *Fast Software Encryption – FSE’2000*. New York, USA: Springer, 2000. (Lecture Notes in Computer Science, v. 1978), p. 213–230.

_____. Helix: Fast encryption and authentication in a single cryptographic primitive. In: *Fast Software Encryption*. Berlin, Germany: Springer-Verlag, 2003. (Lecture Notes in Computer Science, v. 2887), p. 330–346. Disponível em: <<http://www.macfergus.com/helix>>.

GILBERT, H.; MINIER, M. A collision attack on seven rounds of RIJNDAEL. In: *Third Advanced Encryption Standard Candidate Conference – AES3*. New York, USA: NIST, 2000. p. 230–241.

GREENWOOD, P. E.; NIKULIN, M. S. *A Guide to Chi-Squared Testing*. New York, NY: Wiley, 1996.

HALEVI, S.; COPPERSMITH, D.; JUTLA, C. Scream: a software-efficient stream cipher. In: *FSE ’02: Revised Papers from the 9th International Workshop on Fast Software Encryption*. London, UK: Springer-Verlag, 2002. p. 195–209. ISBN 3-540-44009-7. Disponível em: <citeseer.ist.psu.edu/halevi02scream.html>.

HEYS, H. M. A tutorial on linear and differential cryptanalysis. *Cryptologia*, v. 26, n. 3, p. 189–221, 2002. ISSN 0161–1194. Disponível em: <<http://citeseer.ist.psu.edu/443539.html>>.

HILL, J. et al. System architecture directions for networked sensors. In: *Architectural Support for Programming Languages and Operating Systems*. [s.n.], 2000. p. 93–104. Disponível em: <citeseer.ist.psu.edu/hill00system.html>.

IEEE. *Standard Specifications for Public-Key Cryptography – IEEE Std 1363:2000*. [S.l.], 2000.

JAKOBSEN, T.; KNUDSEN, L. R. The interpolation attack on block ciphers. In: *Fast Software Encryption – FSE’95*. Haifa, Israel: Springer, 1997. (Lecture Notes in Computer Science, v. 1267), p. 28–40.

JUNOD, P. *Statistical Cryptanalysis of Block Ciphers*. Tese (Doutorado) — Ecole Polytechnique Fédérale de Lausanne, 2005.

KARENOS, K.; KALOGERAKI, V. Real-time traffic management in sensor networks. In: *RTSS'06: Proceedings of the 27th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2006. p. 422–434. ISBN 0-7695-2761-2.

KARLOF, C.; SASTRY, N.; WAGNER, D. Tinysec: a link layer security architecture for wireless sensor networks. In: *2nd International Conference on Embedded Networked Sensor Systems – SenSys'2004*. Baltimore, USA: ACM, 2004. p. 162–175. ISBN 1-58113-879-2.

KELIHER, L.; MEIJER, H.; TAVARES, S. A new substitution-permutation network cryptosystem using key-dependent S-boxes. In: *Proceedings of Fourth International Workshop on Selected Areas in Cryptography (SAC'97)*. [S.l.: s.n.], 1997. p. 13–26.

KELLY, T. The myth of the skytale. *Cryptologia*, v. 22, n. 3, p. 244–260, 1998.

KELSEY, J.; SCHNEIER, B.; WAGNER, D. Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. *Lecture Notes in Computer Science*, v. 1109, p. 237–251, 1996. Disponível em: <<http://www.schneier.com/paper-key-schedule.pdf>>.

KNUDSEN, L. *DEAL: A 128-bit block cipher*. Norway, 1998. Disponível em: <citeseer.ist.psu.edu/knudsen98deal.html>.

KNUDSEN, L. R. Truncated and higher order differentials. In: *Fast Software Encryption – FSE'95*. New York, USA: Springer, 1995. (Lecture Notes in Computer Science, v. 1008), p. 196–211. Disponível em: <<http://citeseer.ist.psu.edu/knudsen95truncated.html>>.

KNUDSEN, L. R.; ROBSHAW, M. J. B. Non-linear approximations in linear cryptanalysis. In: *Advances in Cryptology – Eurocrypt'96*. [S.l.]: Springer, 1996. (Lecture Notes in Computer Science, v. 1070), p. 224–236.

LABORATORIES, S. N. *Submission to NIST: Cipher-State (CS) Mode of Operation for AES*. 2004. [Csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/cs/cs-spec.pdf](http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/cs/cs-spec.pdf).

LAI, X. *On the Design and Security of Block Ciphers*. Hartung-Gorre, Germany: Konstanz, 1992.

LAI, X.; MASSEY, J. L.; MURPHY, S. Markov ciphers and differential cryptanalysis. In: *Advances in Cryptology – Eurocrypt'91*. Brighton, UK: Springer, 1991. (Lecture Notes in Computer Science, v. 547), p. 17–38.

LAW, Y. W.; DOUMEN, J.; HARTEL, P. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, v. 2, n. 1, p. 65–93, 2006. ISSN 1550-4859.

LEVIS, P. et al. TinyOS: An operating system for wireless sensor networks. In: _____. *Ambient Intelligence*. [S.l.]: Springer-Verlag, 2004.

LI, T. et al. *SenSec Design*. [S.l.], February 2005.

LIDL, R.; NIEDERREITER, H. *Finite Fields*. 2nd. ed. Cambridge, UK: Cambridge University Press, 1997. (Encyclopedia of Mathematics and its Applications, 20).

LIU, D.; NING, P.; LI, R. Establishing pairwise keys in distributed sensor networks. In: *CCS'03: Proceedings of the 10th ACM conference on Computer and communications security*. [S.l.]: ACM, 2003. p. 52–61. ISBN 1-58113-738-9.

LUCKS, S. Attacking seven rounds of RIJNDAEL under 192-bit and 256-bit keys. In: *Third Advanced Encryption Standard Candidate Conference – AES3*. New York, USA: NIST, 2000. p. 215–229.

LUK, M. et al. Minisec: A secure sensor network communication architecture. In: *IPSN'07: Proceedings of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007. p. 479–488. ISBN 978-1-59593-638-X.

MACWILLIAMS, F. J.; SLOANE, N. J. A. *The theory of error-correcting codes*. [S.l.]: North-Holland Mathematical Library, 1977.

MALAN, D. J.; WELSH, M.; SMITH, M. A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography. In: *Proceedings of The First IEEE International Conference on Sensor and Ad Hoc Communications and Networks (IEEE SECON)*. [S.l.: s.n.], 2004. p. 71–80.

MARTINS, G. Y. *Projeto de um dispositivo de autenticação e assinatura*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, 2007.

MATSUI, M. Linear cryptanalysis method for DES cipher. In: *Advances in Cryptology - Eurocrypt '93*. Lofthus, Norway: Springer-Verlag, 1993. (Lecture Notes in Computer Science, v. 765), p. 62–73. Disponível em: <http://homes.esat.kuleuven.be/~abiryuko/Cryptan/matsui_des.PDF>.

_____. New block encryption algorithm MISTY. In: *Fast Software Encryption – FSE'97*. [S.l.]: Springer, 1997. (Lecture Notes in Computer Science, v. 1267), p. 54–68.

MATSUI, M.; YAMAGISHI, A. A new method for known plaintext attack of FEAL cipher. In: *Eurocrypt*. [s.n.], 1992. p. 81–91. Disponível em: <<http://link.springer.de/link/service/series/0558/bibs/0658/06580081.htm>>.

MICROCHIP. *PIC18F8490 Datasheet*. [S.l.], 2006. Disponível em: <<http://www.chipcatalog.com/Microchip/PIC18F8490.htm>>.

MIRZA, F. *Block ciphers and cryptanalysis*. [S.l.], 1998. Disponível em: <<http://citeseer.ist.psu.edu/266201.html>>.

NIST. **Federal Information Processing Standard (FIPS PUB 46) – Data Encryption Standard (DES)**. [S.l.], January 1977. Disponível em: <<http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>>.

_____. **Federal Information Processing Standard (FIPS 197) – Advanced Encryption Standard (AES)**. [S.l.], November 2001. Disponível em: <<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>>.

_____. **Special Publication SP 800-38A – Recommendations for Block Cipher Modes of Operation, Methods and Techniques**. [S.l.], December 2001. Disponível em: <<http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>>.

_____. **Federal Information Processing Standard (FIPS 180-2) – Secure Hash Standard (SHS)**. [S.l.], August 2002. Disponível em: <<http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf>>.

_____. **Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher – Special Publication 800-67**. [S.l.], March 2004. Disponível em: <<http://csrc.nist.gov/publications/nistpubs...67/SP800-67.pdf>>.

NSA. **Skipjack and KEA Algorithm Specifications, version 2.0**. [S.l.], May 1998. Disponível em: <<http://csrc.nist.gov/CryptoToolkit/skipjack/skipjack.pdf>>.

NYBERG, K. Linear approximations of block ciphers. In: **Advances in Cryptology - Eurocrypt '94: Workshop on the Theory and Application of Cryptographic Techniques**. [S.l.]: Springer-Verlag, 1995. (Lecture Notes in Computer Science, v. 950), p. 439–444.

OLIVEIRA, L. et al. **TinyTate: Identity-Based Encryption for Sensor Networks**. 2007. Cryptology ePrint Archive, Report 2007/020. Disponível em: <<http://eprint.iacr.org/>>.

POSCHMANN, A. et al. A family of light-weight block ciphers based on des suited for rfid applications. In: **Workshop on RFID Security 2006**. Graz, Austria: [s.n.], 2006. Disponível em: <<http://events.iaik.tugraz.at/RFIDSec06/Program/index.htm>>.

PRENEEL, B. et al. **Comments by the NESSIE Project on the AES Finalists**. [S.l.], 2000. Disponível em: <https://www.cosic.esat.kuleuven.be/nessie/deliverables/D4_NessieAESInput.pdf>.

RIJMEN, V.; PRENEEL, B. Cryptanalysis of mcguffin. In: **Fast Software Encryption**. [s.n.], 1994. p. 353–358. Disponível em: <citeseer.ist.psu.edu/rijmen95cryptanalysis.html>.

RIVEST, R. L. The RC5 encryption algorithm. In: **Fast Software Encryption – FSE'94**. [S.l.]: Springer, 1995. (Lecture Notes in Computer Science, v. 1008), p. 86–96.

RIVEST, R. L.; SHAMIR, A.; ADELMAN, L. **A Method for obtaining Digital Signatures and Public-Key Cryptosystems**. [S.l.], 1977. 15 p. Disponível em: <citeseer.ist.psu.edu/rivest78method.html>.

ROSE, G.; HAWKES, P. *Turing, a fast stream cipher*. 2002. Cryptology ePrint Archive, Report 2002/185. Disponível em: <<http://eprint.iacr.org/>>.

SCHÄFER, G. *Security in Fixed and Wireless Networks: An Introduction to securing data communications*. [S.l.]: John Wiley & Sons, 2004. ISBN 0470863706.

SCHNEIER, B.; KELSEY, J. Unbalanced Feistel networks and block cipher design. *Lecture Notes in Computer Science*, v. 1039, p. 121–144, 1996. Disponível em: <citeseer.ist.psu.edu/article/schneier96unbalanced.html>.

SELctimescUK, A. A. On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, v. 21, n. 1, p. 131–147, 2008. Disponível em: <<http://dx.doi.org/10.1007/s00145-007-9013-7>>.

SHANNON, C. E. Communication theory of secrecy systems. *Bell System Technical Journal*, v. 28, p. 656–715, 1949.

SHIMIZU, A.; MIYAGUCHI, S. Fast data encipherment algorithm FEAL. In: *Eurocrypt*. [s.n.], 1987. p. 267–278. Disponível em: <<http://link.springer.de/link/service/series/0558/bibs/0304/03040267.htm>>.

SIDDIQUE, S. M.; AMIR, M. Gsm security issues and challenges. *7th ACIS International Conference on Software Engineering, AI, Networking, and Parallel/Distributed Computing*, p. 413–418, 2006. Disponível em: <<http://ieeexplore.ieee.org/iel5/9317/29618/01345023.pdf>>.

THORSTEINSON, P.; GANESH, G. *NET Security and Cryptography*. [S.l.]: Prentice Hall PTR, 2003. ISBN 0-13-100851-X.

TITZER, B.; LEE, D.; PALSBERG, J. *Avrora Scalable Simulation of Sensor Networks with Precise Timing*. 2004. Center for Embedded Network Sensing Posters - Paper 93. Disponível em: <<http://compilers.cs.ucla.edu/emsoft05/TitzerLeePalsberg05.pdf>>.

VAUDENAY, S. An experiment on DES statistical cryptanalysis. In: *ACM Conference on Computer and Communications Security*. [s.n.], 1996. p. 139–147. Disponível em: <citeseer.ist.psu.edu/vaudenay95experiment.html>.

VENUGOPALAN, R. et al. Encryption overhead in embedded systems and sensor network nodes: modeling and analysis. In: *Cases '03: Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. New York, NY, USA: ACM Press, 2003. p. 188–197. ISBN 1-58113-676-5.

WACKER, A. et al. A new approach for establishing pairwise keys for securing wireless sensor networks. In: *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2005. p. 27–38. ISBN 1-59593-054-X.

WAGNER, D. The boomerang attack. In: *Fast Software Encryption – FSE'99*. Rome, Italy: Springer, 1999. (Lecture Notes in Computer Science, v. 1636), p. 156–170.

WHEELER, D. J.; NEEDHAM, R. M. TEA, a tiny encryption algorithm. *Lecture Notes in Computer Science*, v. 1008, p. 363–377, 1995. Disponível em: <citeseer.ist.psu.edu/article/wheeler95tea.html>.

WILLIAMSON, M. J. *Non-Secret Encryption Using a Finite Field*. [S.l.], 1974. Disponível em: <<http://www.cesg.gov.uk/site/publications/media/secenc.pdf>>.

YANG, B. Y.; CHEN, J. M. All in the XL family: Theory and practice. In: *ICISC*. [s.n.], 2004. p. 67–86. Disponível em: <http://dx.doi.org/10.1007/11496618_7>.

YOUSSEF, A. M.; MISTER, S.; TAVARES, S. E. On the design of linear transformations for substitution permutation encryption networks. In: *Selected Areas in Cryptography – SAC’97*. [S.l.: s.n.], 1997. (Proceedings), p. 40–48.

YOUSSEF, A. M.; TAVARES, S. E.; HEYS, H. M. A new class of substitution-permutation networks. In: *Selected Areas in Cryptography – SAC’96*. [S.l.: s.n.], 1996. (Proceedings), p. 132–147.

APÊNDICE A - PROJETO DAS MATRIZES *D E E*

Conforme discutido na seção 7.2.2 as matrizes D e E assumem papéis importantes no CURUPIRA. Este apêndice discute de que forma o seu projeto atende aos requisitos impostos inicialmente para a cifra, o comportamento involutivo da função de encriptação e a ciclicidade do escalonamento de chaves. Desta forma, as seções a seguir têm como objetivo apresentar a forma como o projeto destas matrizes é capaz de satisfazer estas propriedades.

A.1 Matriz D : Comportamento Involutivo

Conforme discutido no final da seção 4.1, que discute o projeto de Redes de Substituição-Permutação, algumas cifras modernas adotam funções de round compostas apenas de operações auto-inversas, de forma que as mesmas passem a apresentar um comportamento involutivo. Além disto, conforme discutido na seção 5.7, uma forma bastante interessante de se implementar a transformação de difusão θ exigida pela Estratégia de Trilha Larga (veja seção 5.6.1) é por meio da multiplicação dos blocos por matrizes MDS. É desta forma que a matriz D , utilizada para implementar a camada de difusão linear do CURUPIRA, apresenta logo de início dois requisitos básicos: ser MDS e involutiva. Além disto, por razões de desempenho, é interessante que a mesma apresente os coeficientes mais simples possíveis, de forma a facilitar os cálculos envolvidos na multiplicação.

Assim, o projeto da matriz D consiste simplesmente na busca de coeficientes que satisfazem estas propriedades, partindo-se de uma matriz 3×3 em $\text{GF}(2^m)$ da forma mais geral possível, ou seja:

$$D = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Para que D seja involutivo, é então necessário que:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Desta forma, chega-se ao seguinte conjunto de equações não-lineares:

$$\left\{ \begin{array}{l} (1) \quad a^2 + bd + cg = 1 \\ (2) \quad ad + de + fg = 0 \\ (3) \quad ag + dh + gi = 0 \\ (4) \quad ab + be + ch = 0 \\ (5) \quad bd + e^2 + fh = 1 \\ (6) \quad bg + eh + hi = 0 \\ (7) \quad ac + bf + ci = 0 \\ (8) \quad dc + ef + fi = 0 \\ (9) \quad cg + fh + i^2 = 1 \end{array} \right.$$

Das equações (1), (5) e (9), tem-se que $i^2 + a^2 + e^2 = 1$. Como $(x + y)^2 = x^2 + y^2$ (lembrando que a operação de soma corresponde a um XOR bit-a-bit), esta equação é equivalente a $i + a + e = 1$. Substituindo esta igualdade nas equações de (1) a (9) e

reagrupando os parâmetros, chega-se a:

$$\left\{ \begin{array}{lcl} (1') & a^2 + bd + cg & = 1 \\ (2') & fg & = d(a+e) \\ (3') & dh & = g(1+e) \\ (4') & ch & = b(a+e) \\ (5') & bd + e^2 + fh & = 1 \\ (6') & bg & = h(1+a) \\ (7') & bf & = c(1+e) \\ (8') & dc & = f(1+a) \\ (9') & a + e + i & = 1 \end{array} \right.$$

Na verdade, este sistema contém equações dependentes entre si. Por exemplo, multiplicando (8') e (4') tem-se que $fb(1+a)(a+e) = dcch$; aplicando (3') e (7') nas porções esquerda e direita da equação, respectivamente, chega-se a $(1+a)(a+e) = cg$. Analogamente, a multiplicação de (7') e (2') seguida de (6') e (8') leva a $(1+e)(a+e) = fh$. A soma destes dois resultados leva a $cg + fh = a^2 + e^2$, exatamente o mesmo resultado que se obtém da soma entre as equações (1') e (5'), que não haviam sido utilizadas anteriormente.

A resolução completa deste sistema de equações pode ser uma tarefa bastante demorada. Entretanto, como existem graus de liberdade disponíveis, pode-se fazer uma simplificação de modo que $b = c, d = f, g = h$. Esta eliminação de 3 graus de liberdade removem operações de multiplicação no sistema, fazendo com que a matriz D assumia a forma:

$$D = \begin{bmatrix} a+1 & a & a \\ e & e+1 & e \\ a+e & a+e & a+e+1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + a \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} + e \cdot \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Ou seja, $D = I + \alpha A + \beta B$. É fácil verificar que qualquer matriz D com esta forma

é involutiva, já que $D^2 = (I + \alpha A + \beta B)^2 = I^2 + A^2 + AB + BA + B^2$ e, por simples inspeção, percebe-se que $A^2 = B^2 = AB = BA = O$, onde O é a matriz nula. Além disto, conforme explicado na seção 7.2.2, os menores coeficientes α e β possíveis para que uma matriz desta forma seja MDS são $\alpha = 2$ e $\beta = 4$, chegando-se então à matriz D utilizada:

$$D = \begin{bmatrix} 3 & 2 & 2 \\ 4 & 5 & 4 \\ 6 & 6 & 7 \end{bmatrix}$$

A.2 Matriz E : Escalonamento Cíclico

O projeto da matriz E vem da necessidade de uma matriz facilmente inversível. Uma primeira idéia, que se mostrou bastante frutífera, foi a busca de uma matriz E da forma $E = I + cC$ e $E^{-1} = I + dC$, ou seja, uma matriz igual à identidade I exceto por uma pequena perturbação dada pela matriz C , e cuja inversa assume uma forma semelhante. A matriz C e os coeficientes c e d que satisfazem estas condições podem, então, ser calculados da seguinte forma:

$$E \cdot E^{-1} = I(I + cC) \cdot (I + dC) = II + (c + d)C + cdC^2 = I$$

Esta equação sugere uma simplificação bastante interessante: a adoção de uma matriz idempotente, ou seja, uma matriz que satisfaz a propriedade $C^2 = C$. Desta forma, chega-se a:

$$I + (c + d)C + cdC = I(c + d + cd)C = O$$

Na equação anterior, O representa a matriz nula. Assim, uma segunda restrição necessária para garantir que a matriz E seja diferente da identidade, é fazer com que C seja não-singular (i.e. $C \neq O$). A matriz na qual todos os elementos são iguais a '1', além de satisfazer às condições impostas, apresenta uma forma bastante simples e,

portanto, foi escolhida. Desta forma, chega-se à seguinte relação entre escalares:

$$c + d + cd = 0d(1 + c) = cd = c/(1 + c)$$

Este resultado sugere que o parâmetro c deve ser escolhido de modo que $d = c/(1 + c)$ exista (i.e. $c \neq 1$) e assuma a forma mais simples possível, facilitando o cálculo de E^{-1} . É por esta razão que o valor de c foi escolhido conforme mostrado na seção 7.2.2, que impõe como requisito adicional que E seja MDS. Para que esta última condição seja satisfeita, deve-se então adotar um valor de $c \neq 0, 1$.

Tendo estes requisitos iniciais em mente, o cálculo do valor exato de c foi descrito de forma bastante detalhada na própria seção 7.2.2. Na verdade, descobriu-se que uma forma particularmente simples de d poderia ser obtida caso c fosse escolhido como a raiz cúbica da unidade (i.e. $c^3 = 1$). Esta observação foi possível graças à utilização de uma técnica de fatoração de polinômios bastante conhecida: $x^n - 1 = (x - 1)(x^{n-1} + x^{n-2} + \dots + x + 1)$. Como soma e subtração em $\text{GF}(2^8)$ se resumem à operação de XOR, pode-se escrever para $n = 3$: $c^3 + 1 = (c + 1)(c^2 + c + 1)$. Desta forma, adotando-se $c^3 = 1$, tem-se que:

$$c^3 = 1 \Rightarrow c^2 = 1/c$$

$$c^3 + 1 = 0 \Rightarrow (c + 1)(c^2 + c + 1) = 0 \Rightarrow c^2 + c + 1 = 0 \text{ (já que, por projeto, } c \neq 1)$$

$$c^2 = c + 1 \Rightarrow 1/(c + 1) = 1/c^2 = c$$

$$\underline{c/(c + 1) = c^2 = c + 1}$$

Conforme desejado, esta escolha para o valor de c leva a uma forma particularmente simples para E^{-1} , a saber $E^{-1} = I + (c + 1)C$. É importante ainda observar que a forma apresentada pela matriz E obtida desta maneira é bastante interessante por permitir o escalonamento cíclico de chaves, com um número adequado de rounds, quando sua aplicação é intercalada com um deslocamento cíclico das linhas da matriz alvo.

APÊNDICE B - ATAQUE DE SATURAÇÃO APLICADO AO CURUPIRA

Este apêndice descreve o Ataque de Saturação aplicado ao CURUPIRA. Ao longo da discussão, a^R será usado para indicar o estado de entrada do round R , enquanto b^R denota o estado de saída do round R e k^R denota a sub-chave deste round.

Conforme descrito na seção 4.3.5, o Ataque de Saturação consiste na utilização de conjuntos de blocos de texto claro com ao menos 1 posição saturada, os quais recebem o nome de Integrais.

B.1 Comportamento para 3 rounds

Para o ataque básico ao CURUPIRA, as integrais são compostas por 2^8 blocos, nos quais a posição (0,0) assume todos os valores possíveis e as outras posições se mantêm constantes. A adição de chave inicial não afeta a estrutura da Integral, pois apenas muda o valor das posições constantes, mantendo saturada a posição (0,0). A Figura 27 ilustra esta propriedade.

O primeiro round da cifra tem o seguinte efeito sobre a^1 : a operação γ altera os elementos individualmente, porém não altera a estrutura da integral (lembrando que a S-Box é bijetora); π apenas altera a posição de alguns dos elementos da integral, mas a posição saturada não é afetada; já θ causa a difusão da posição saturada por toda a coluna 0, fazendo então com que cada um de seus elementos apresente todos os valores possíveis; a aplicação da chave k^1 não altera esta estrutura. Portanto, o efeito

Figura 27: Comportamento do CURUPIRA para 3 rounds

do primeiro round é apenas de espalhar a saturação de uma única posição para todos os elementos da coluna 0 no estado b^1 .

Já o segundo round aplicado a $b^1 \equiv a^2$ tem o seguinte efeito: novamente γ não altera a estrutura da integral; π permuta as posições $a_{1,0}^2$ com $a_{1,1}^2$ e $a_{2,0}^2$ com $a_{2,2}^2$; a aplicação de θ leva agora à saturação das 3 primeira colunas do estado, situação que não é alterada pela adição da sub-chave k^2 .

Por fim, o terceiro round da cifra aplicado a $b^2 \equiv a^3$ leva à seguinte situação: γ não afeta esta estrutura; π faz com que as posições não saturadas da cifra passem a ser (0,3), (1,2) e (2,1); como agora existem ao menos 2 posições saturadas em todas as colunas, uma nova aplicação de θ em geral destrói a estrutura da integral, que passa a apresentar uma estrutura aparentemente aleatória. Contudo, como θ combina linearmente (com XOR) os elementos da coluna, uma característica importante se conserva: o **balanço**, o que significa que a soma de todos os elementos de uma determinada posição da integral é nula. Esta propriedade não é afetada pela adição da sub-chave k^3 , já que um mesmo valor de chave é somado exatamente 2^8 vezes no cálculo do balanço de cada posição do estado, anulando-se.

B.2 O ataque básico de 4 rounds

O ataque de saturação mais básico ao CURUPIRA se estende a 4 rounds da cifra, bastando para isto adicionar um round ao estado $b^3 \equiv a^4$ descrito anteriormente. Como este round final não inclui a transformação θ , tem-se que $b^4 = \pi(\gamma(a^4)) \oplus k^4$, o que pode ser reescrito como $\gamma(\pi(b^4) \oplus \pi(k^4))$. A Figura 28 ilustra o ataque.

Figura 28: Ataque de saturação a 4 rounds do CURUPIRA

O atacante já tem acesso ao valor de $b_{i,j}^4$ para toda posição (i,j) . Para descobrir o valor de k^4 , basta então “chutar” o valor correspondente de $k_{i,j}^4$ e calcular o $a_{i,j}^4$, resultante para todos os textos claros da integral. Caso a chave testada seja a correta, a soma de todos os valores de $a_{i,j}^4$ será necessariamente 0, em função do balanço. Já para uma chave incorreta, o valor da soma será nulo com uma probabilidade de aproximadamente 2^{-8} . Como, para cada posição, existem 2^8 valores de chave possíveis, aproximadamente 1 chave passará pelo teste. Repetindo-se o processo para uma outra integral (basta alterar algum dos valores nas posições fixas), o valor correto de $k_{i,j}^4$ é filtrado com uma probabilidade esmagadora: a probabilidade de uma chave incorreta passar pelo teste é de 2^{-16} . Na verdade, não é necessário testar todos os valores de chaves para esta segunda integral, mas apenas o pequeno sub-conjunto filtrado pela primeira integral.

Portanto, a complexidade total do ataque de saturação a 4 rounds do CURUPIRA é de

2 integrais completas (2^9 textos claros) e 2^8 testes de sub-chave para cada posição da chave e para cada um dos textos claros. Desde que implementado de forma adequada, o custo da decifração parcial para 256 textos claros pode ser comparada a uma única encriptação completa.

B.3 Adicionando um round no final

O ataque básico de 4 rounds pode ser estendido pela adição de um round extra no final da cifra. Neste caso, será o 5º e não mais o 4º round que não apresentará a camada de difusão θ . Observa-se que os elementos $b_{i,i}^5$, $0 \leq i \leq 2$ dependem exclusivamente da primeira coluna de a^5 , que por sua vez depende exclusivamente dos elementos $a_{i,i}^4$, $0 \leq i \leq 2$. Tendo isto em mente, o ataque consiste em “chutar” todos os 3 valores de $k_{i,i}^5$ e calcular $a^5 = \gamma(\pi(b^5) \oplus \pi(k^5))$ para estas posições, ou seja, $a_{i,0}^5 = S[b_{i,i}^5 \oplus k_{i,i}^5]$. Em seguida, como $a^4 = \gamma(\pi \circ \theta(b^4) \oplus \pi \circ \theta(k^4))$, o byte $a_{i,0}^4$ é calculado a partir do valor de b^4 encontrado anteriormente e pelo “chute” do valor de $\pi \circ \theta(k^4)$ na posição $(i, 0)$ correspondente. Pode-se então prosseguir de maneira semelhante ao ataque de 4 rounds usando esta posição $(i, 0)$. A Figura 29 ilustra o ataque.

Figura 29: Ataque de saturação a 5 rounds do CURUPIRA

Deste modo, o ataque recupera 3 bytes da sub-chave do último round e informação parcial sobre a sub-chave do penúltimo round, já que apenas 1 byte de $\pi \circ \theta(k^4)$ precisa

ser calculado. O restante dos bytes pode ser recuperado simplesmente repetindo-se o processo anterior para posições diferentes do estado b^5 . Todavia, 3 aplicações do método seguidas de uma busca exaustiva final para os bytes restantes é normalmente uma abordagem mais eficiente para recuperar a sub-chave k^5 completa.

Como o método consiste no teste de $2^{8 \cdot 3} \cdot 2^8$ chaves e a probabilidade de que uma chave incorreta leve à saturação da posição testada é de 2^{-8} por integral, são necessárias 5 integrais para filtrar as chaves corretas ($2^{32} \cdot (2^{-8})^5 \ll 1$). O custo do ataque é então de $5 \cdot 2^8 \approx 2^{11}$ textos claros e 2^{32} testes de chave para cada um deles. Considerando a aproximação anterior de que a decifração parcial para um conjunto de 256 textos corresponde a uma encriptação completa e que a repetição do processo para um conjunto 2^8 vezes menor de chaves é bem mais rápida do que o processo inicial, a complexidade computacional é calculada como $2^{32} \cdot 2^{11} \cdot 2^{-8} = 2^{35}$ encriptações completas.

B.4 Adicionando um round no início

A idéia básica consiste na escolha de 2^8 textos claros de forma que a posição $b_{0,0}^1$ fique saturada, prosseguindo então com o ataque de 5 rounds descrito na seção anterior. Para isto, é necessário construir uma integral de 2^{24} textos claros cujos elementos da diagonal principal, $a_{i,i}^1$, com $0 \leq i \leq 2$, estejam saturados, além de “chutar” os bytes correspondente da sub-chave k^1 . Como os elementos da primeira coluna de b^1 dependem exclusivamente destas posições de a^1 , é possível filtrar 2^8 textos claros para os quais o byte $b_{0,0}^1$ está saturado e as outras posições assumem valores constantes. Para cada uma das 2^{24} chaves k^1 testadas, procede-se simplesmente com o ataque de 5 rounds descrito na seção anterior. A Figura 30 ilustra este ataque.

Há, entretanto, uma maneira mais eficiente de adicionar um round ao início da cifra. Os 2^{24} textos claros anteriores podem ser vistos como 2^{16} integrais de 2^8 textos

Figura 30: Ataque de saturação a 6 rounds do CURUPIRA

claros, cada qual com uma única posição saturada em b^1 . Como o XOR dos bytes em qualquer posição de a^5 é nulo para cada uma das 2^{16} integrais, o mesmo é válido quando se considera o XOR para todas estas integrais. Portanto, o valor de $a_{i,j}^5$ pode ser obtido da mesma forma que era feito com a^4 no caso do ataque a 5 rounds do CURUPIRA: basta “chutar” 3 bytes da sub-chave k^6 e 1 byte de $\pi \circ \theta(k^5)$, efetuando o teste sobre os 2^{24} textos (ao invés de apenas 2^8) para checar o balanço. Como cada teste filtra da ordem de 2^8 chaves incorretas, são necessários 5 grupos de 2^{24} textos claros para recuperar 3 bytes da última sub-chave.

A complexidade total do ataque é de $5 \cdot 2^{24} \approx 2^{27}$ textos claros e 2^{32} testes de chave para cada um deles. Considerando as mesmas condições discutidas para o ataque de 5 rounds, a complexidade do ataque descrito é da ordem de $2^{32} \cdot 2^{27} \cdot 2^{-8} = 2^{51}$ encriptações completas.

B.5 Soma-parcial aplicada a 6 rounds

Existe ainda um modo de aumentar a velocidade do ataque a 6 rounds, ao custo de uma maior utilização de memória, usando a técnica da *Soma Parcial* (FERGUSON et al., 2000). Denotando por c_0, c_1, c_2 as posições de b^6 usadas no ataque da seção anterior e

por k_0, k_1, k_2, k_3 os bytes de chave testados, pode-se escrever:

$$a_{i,j}^5 = S[(\bigoplus_{t=0}^2 S_t[c_t \oplus k_t]) \oplus k_3] \quad (\text{B.1})$$

Onde S_t são S-boxes bijetoras consistindo da aplicação de S seguida pela multiplicação por um elemento da matriz D , como resultado da transformação θ . Pode-se então denotar por $x_n = \bigoplus_{t=0}^n S_t[c_t \oplus k_t]$ a **soma parcial** no cálculo de $a_{i,j}^5$. Como o cálculo do balanço envolve o XOR de todos os $a_{i,j}^5$ para uma determinada posição (i, j) , é fácil perceber que valores idênticos de $x_n, c_{n+1}, \dots, c_2$ cancelam-se mutuamente. Portanto, pode-se reduzir o tempo gasto para calcular o balanço “chutando” os valores de k_0 e k_1 e contando o número de ocorrências da dupla (x_1, c_2) sobre o conjunto de textos cifrados usados no ataque; em seguida “chuta-se” k_2 e conta-se a ocorrência de x_2 para todos os valores de (x_1, c_2) calculados na etapa anterior; finalmente, basta “chutar” k_3 e calcular a soma sobre todos os valores de x_2 , aplicar a S-Box aos mesmos e então calcular o XOR do resultado. Como apenas a paridade da ocorrência é importante, a contagem pode ser feita módulo 2, ou seja, podem ser usados $2^{16} + 2^8$ contadores de 1 bit para armazenar as contagens.

O esforço total usando este método é menor do que aquele da seção B.4. Na primeira etapa do ataque são testados 2^{16} pares (k_0, k_1) sobre 2^{24} textos cifrados. Para cada (k_0, k_1) , na segunda etapa são testados 2^8 valores de k_2 para sobre os 2^{16} valores possíveis de (x_1, c_2) . Na terceira etapa, para cada tripla (k_0, k_1, k_2) são testados 2^8 valores de k_4 sobre os 2^8 valores de x_2 . O custo do processo é então de 2^{40} aplicações da equação B.1 para cada uma das 5 integrais, o que resulta em $5 \cdot 2^{40}$ indexações da S-Box ou cerca de 2^{37} encriptações completas de 6 rounds.

B.6 Soma-parcial aplicada a 7 rounds

A extensão do ataque para 7 rounds do CURUPIRA usando a técnica conhecida como *Herds Attack* (algo como “ataque de rebanho”) é bem mais complexa do que nos casos anteriores, sendo difícil afirmar com certeza se a mesma é aplicável. Caso o seja, seu custo envolve um total de $2^{96} - 2^{87}$ textos claros e um esforço de cerca de 2^{88} encriptações de 7 rounds, bem como uma enorme quantidade de memória para armazenar todos os textos, e contagens parciais. Apesar de se tratar de um resultado teórico interessante, como o ataque envolve a encriptação de cerca de 99.6% de todos os possíveis textos claros, este ataque tem um reduzido interesse prático: se um atacante fosse capaz de obter esta imensa quantidade de pares de texto claro/cifrado, seria mais viável para ele construir uma lista contendo todos estes pares e simplesmente utilizá-la para decriptar 99.6% dos textos sem nenhum conhecimento da chave utilizada, com um esforço computacional muito inferior a 2^{88} encriptações.

APÊNDICE C - DETALHES ADICIONAIS DA S-BOX

Conforme discutido anteriormente, a S-box adotada pelo CURUPIRA S-box é a mesma utilizada pelas cifras ANUBIS (BARRETO; RIJMEN, 2000a) e KHAZAD (BARRETO; RIJMEN, 2000b). A seguir, são dados alguns detalhes extras sobre sua construção e características

C.1 Formas de implementação

Existem basicamente duas formas de calcular esta S-Box. O primeiro modo é por meio das mini-boxes P e Q que constituem a cifra completa, solução já apresentada na seção 7.3.1.1 por meio do Algoritmo 1. O outro modo é utilizando uma tabela completa, que ocupa 256 bytes de memória, dada pela Tabela 6.

C.2 Segurança

A S-Box do CURUPIRA apresenta $\delta_S = 2^{-5}$, $\lambda_S = 2^{-2}$, e $\nu_S = 7$ e, desta forma, previne ataques diferenciais, lineares e de interpolação. Ela foi também concebida de forma que $S[S[x]] = x$, $\forall x \in \text{GF}(2^8)$, ou seja, de modo a apresentar um comportamento involutivo.

Tabela 6: A S-Box completa CURUPIRA

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