DÉCIO LUIZ GAZZONI FILHO

PROJETO, ANÁLISE E IMPLEMENTAÇÃO DE PRIMITIVAS CRIPTOGRÁFICAS SIMÉTRICAS EFICIENTES USANDO A ESTRATÉGIA DE TRILHA LARGA

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

São Paulo
Edição Original: 2007
Edição Revisada: 2008

DÉCIO LUIZ GAZZONI FILHO

PROJETO, ANÁLISE E IMPLEMENTAÇÃO DE PRIMITIVAS CRIPTOGRÁFICAS SIMÉTRICAS EFICIENTES USANDO A ESTRATÉGIA DE TRILHA LARGA

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:

Prof. Dr. Paulo S. L. M. Barreto

São Paulo
Edição Original: 2007
Edição Revisada: 2008

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência do seu orientador.

São Paulo, 27 de março de 2008.

Assinatura do autor

Assinatura do orientador

FICHA CATALOGRÁFICA

Gazzoni Filho, Décio Luiz

Projeto, Análise e Implementação de Primitivas Criptográficas Simétricas Eficientes Usando a Estratégia de Trilha Larga. São Paulo, Edição Original: 2007, Edição Revisada: 2008.

129 p.

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

1. Criptologia. 2. Algoritmos. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais (PCS). II. t.

AGRADECIMENTOS

Aos meus pais, pelo apoio moral e financeiro durante a elaboração desta dissertação.

Ao meu orientador, prof. Paulo Barreto, pelo apoio técnico e paciência infinita durante esta jornada.

Ao prof. Vincent Rijmen, por seu papel no desenvolvimento da estratégia de trilha larga, e pela colaboração no trabalho desenvolvido nesta dissertação.

A Jéssica Rangel, pelo carinho e amizade, ambos sinceros e inabaláveis.

RESUMO

Estendemos o trabalho de Vincent Rijmen e Joan Daemen na estratégia de trilha larga, uma metodologia de projeto para primitivas criptográficas simétricas eficientes e demonstravelmente resistentes às técnicas de criptanálise diferencial e linear. Preocupamo-nos principalmente com a melhoria na eficiência de primitivas projetadas de acordo com a estratégia de trilha larga. Investigamos duas linhas distintas de pesquisa: a aplicabilidade da técnica de bitslicing à implementação em software de primitivas baseadas na estratégia de trilha larga; e o projeto de S-boxes estruturadas com implementação eficiente em hardware e bitslicing, e especificamente, o uso de S-boxes invariantes por rotação, que exibem propriedades vantajosas para implementação. Também implementamos e otimizamos algumas primitivas criptográficas em plataformas de software selecionadas, para substanciar e aprimorar as afirmações de eficiência da estratégia de trilha larga. Ademais, aplicamos nosso conhecimento e técnicas propostas ao projeto de novas primitivas criptográficas altamente eficientes, em particular a função de hash MAELSTROM-0 e a cifra de bloco legada FUTURE.

ABSTRACT

We extend the work of Vincent Rijmen and Joan Daemen on the Wide Trail strategy, a design methodology for symmetric-key cryptographic primitives which are efficient and provably secure against differential and linear cryptanalysis. We concern ourselves mainly with improving the efficiency of primitives designed according to the Wide Trail strategy. To that end, we investigate two distinct lines of research: the applicability of the bitslicing technique to the software implementation of primitives based on the Wide Trail strategy; and the design of structured S-boxes with efficient implementation in hardware and bitslicing, and specifically, the use of rotation-symmetric S-boxes, which exhibit advantageous implementation properties. We also perform general implementation and optimization work on selected software platforms, to further realize the claims of efficiency of the Wide Trail strategy. Additionally, we apply our expertise and proposed techniques to the design of new highly-efficient cryptographic primitives, in particular the hash function MAELSTROM-0 and the legacy-level block cipher FUTURE.

RÉSUMÉ

On apporte des améliorations au travail de Vincent Rijmen et Joan Daemen dans le stratégie de piste large, une technique de conception de primitives cryptographiques symétriques efficientes et à l'abri de cryptanalyse différentielle et linéaire. On s'occupe surtout de l'améloriation de la vitesse des primitives conçus selon la stratégie de piste large. À cette fin, on examine deux lignes distinctes de recherche: la valeur de la technique de bitslicing sur l'implémentation en logiciel des primitives que sont basés sur la stratégie de piste large; et le conception des S-boxes structurées pour l'implémentation efficient en hardware et bitslicing, et spécifiquement, l'emploi des S-boxes invariables pour la rotation, qui ont des propriétés avantageuses pour l'implémentation. On fait aussi des implémentations et optimisations des primitives de chiffrement dans quelques plateformes de logiciel, pour confirmer et améliorer des affirmations de vitesse de la stratégie de piste large. Finalement, on applique notre expérience et les techniques proposées pour la conception de nouvelles primitives cryptographiques de haute vitesse, en particulier la fonction de hachage MAELSTROM-0 e le algorithme de chiffrement par bloc de héritage FUTURE.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

1 INTRODUÇÃO

De acordo com (MENEZES; OORSCHOT; VANSTONE, 1997), criptografia é o "estudo de técnicas matemáticas relacionadas a aspectos da segurança da informação como privacidade, integridade de dados, autenticação de entidades e autenticação de fontes de dados". A pesquisa em criptografia subdivide-se em diversos campos, cada um buscando melhorias em um ou mais dos objetivos acima. Dentre estes campos, a presente dissertação preocupa-se especificamente com a *criptografia de chave secreta* ou *criptografia de chave simétrica*, e mais especificamente ainda, com o projeto de *cifras de bloco*. Avanços no projeto de cifras de bloco resultam diretamente em melhoras no objetivo de privacidade listado acima, e indiretamente nos demais através da aplicação de cifras de bloco na construção de *funções de hash*. Tomados em conjunto, cifras de bloco e funções de hash são empregados na imensa maioria dos protocolos criptográficos em uso corrente; mesmo a chamada *criptografia de chave pública*, capaz de prover funcionalidades inexistentes na criptografia de chave secreta, ainda é implementada em conjunção com esta última em sistemas reais, por razões de desempenho.

A preocupação primária desta dissertação é com a *eficiência* dos criptossistemas projetados. Desde a popularização dos computadores, o projeto de cifras completamente seguras é possível a princípio; mesmo o Data Encryption Standard (DES) (NIST, 1999), projetado há três décadas e considerado ultrapassado devido ao uso de chaves curtas (56 bits), pode ser empregado

de maneira segura através da concatenação de três aplicações da cifra (o chamado Triple-DES ou 3DES). Desta maneira, a pesquisa em cifras de bloco tornou-se um exercício tanto de matemática como de engenharia, utilizando artifícios matemáticos para o projeto de cifras compactas e eficientes. Um dos grandes frutos dessa pesquisa num passado recente é o desenvolvimento da *estratégia de trilha larga* (RIJMEN, 1997; DAEMEN, 1995; DAEMEN; RIJMEN, 2001a; DAEMEN; RIJMEN, 2001b), uma metodologia formal para o projeto de cifras eficientes e demonstravelmente resistentes aos principais ataques criptanalíticos conhecidos. O reconhecimento da importância desta estratégia veio com um concurso do governo americano para escolha de um novo padrão (Advanced Encryption Standard ou AES) (NIST, 2001) para substituir o DES; este concurso foi vencido pela cifra Rijndael, projetada de acordo com a estratégia de trilha larga.

Sendo assim, decidimos por enfocar esta dissertação na estratégia de trilha larga, buscando novas construções mais eficientes para alguns de seus componentes. Quanto a isso, destacamos as seguintes contribuições:

- a descoberta de S-boxes invariantes por rotação (Capítulo 4), uma classe de S-boxes em que entradas relacionadas por uma rotação de bits produzem saídas relacionadas pela mesma rotação, e que apresenta propriedades vantajosas de implementação. Esta descoberta resultou na submissão de um artigo em co-autoria com Paulo S. L. M. Barreto e Vincent Rijmen ao periódico Information Processing Letters, que foi aceito e aguarda publicação;
- o trabalho na implementação de cifras de trilha larga usando bitslicing (Seção 3.1.3), uma técnica de implementação em software que pode ser exprimida naturalmente em processadores SIMD (Single Instruction Multiple Data), a exemplo das extensões vetoriais presentes em todos

os processadores modernos de alto desempenho.

Um objetivo secundário da dissertação é o projeto de novas cifras de bloco e funções de hash utilizando a estratégia de trilha larga; destacamos:

- o projeto da função de hash MAELSTROM-0 (Capítulo 5), uma evolução da função WHIRLPOOL, que gerou um artigo em co-autoria com Paulo S. L. M. Barreto e Vincent Rijmen, agraciado com o prêmio de melhor artigo do SBSEG 2006, e que forma a base de nossa futura submissão para o concurso Advanced Hash Standard (AHS) do governo americano, um concurso nos moldes do já citado AES;
- o projeto da cifra de bloco FUTURE (Capítulo 6), uma cifra com parâmetros legados (blocos de 64 bits e chaves de 128 bits) especialmente compacta e eficiente para implementação em hardware, e com vantagens para implementação em software em dispositivos embarcados como microcontroladores e smart cards, e que gerou um artigo em co-autoria com Paulo S. L. M. Barreto, Jorge Nakahara e Paris Kitsos, em processo de escrita quando da finalização deste trabalho.

2 A ESTRATÉGIA DE TRILHA LARGA

Conforme mencionado no Capítulo 1, o projeto de sistemas criptográficos deve levar em conta tanto a segurança como a eficiência dos sistemas projetados. Neste sentido, uma das maiores contribuições recentes no campo de criptografia simétrica foi a introdução da estratégia de trilha larga (RIJMEN, 1997; DAEMEN, 1995; DAEMEN; RIJMEN, 2001a; DAEMEN; RIJMEN, 2001b), que une demonstrações de segurança contra os principais ataques criptanalíticos conhecidos a uma estrutura eficientemente implementável numa ampla gama de plataformas. Este capítulo descreve a teoria da estratégia de trilha larga, enquanto o Capítulo 3 descreve a implementação eficiente de primitivas projetadas de acordo com esta estratégia.

O presente capítulo é organizado da seguinte maneira: a Seção 2.1 introduz a cifra Data Encryption Standard (DES) (NIST, 1999), a mais importante e mais difundida cifra simétrica quando da concepção da estratégia de trilha larga, e sobre a qual foram montados os dois ataques criptanalíticos de maior importante, os ataques diferencial (Seção 2.2) e linear (Seção 2.3); é instrutivo considerar a aplicação de criptanálise diferencial e linear sobre a cifra para a qual ambos foram concebidos. Uma das contribuições da estratégia de trilha larga foi o emprego de novos formalismos para descrever estes dois ataques, enfatizando pontos em comum entre os ataques, o que permite a concepção de uma estratégia unificada para prevenir a aplicação bem-sucedida destes ataques; estes formalismos são brevemente descritos na Seção 2.4. Neste

ponto, torna-se possível introduzir a estratégia de trilha larga na Seção 2.5: seus conceitos, uma visão geral da estratégia, e uma das construções possíveis que atende aos requisitos da estratégia de trilha larga e é, ao mesmo tempo, eficiente (construção esta que é empregada com poucas modificações em todos os projetos de acordo com a estratégia de trilha larga em existência). Para ilustrar concretamente o projeto de primitivas utilizando a estratégia de trilha larga, a Seção 2.6 discute as decisões de projeto por trás da mais ilustre representante desta estratégia, a cifra Rijndael (AES). Um sumário do capítulo é feito na Seção 2.7.

2.1 A cifra Data Encryption Standard (DES)

Em 1973, o National Bureau of Standards americano (atualmente conhecido como NIST, ou National Institute of Standards and Technology) solicitou propostas para elaboração de um padrão de cifras de bloco, a ser adotado como padrão governamental para cifração de informação importante (porém não-confidencial). Nesta primeira rodada, todas as propostas foram recusadas, e uma segunda solicitação foi feita em 1974. Desta vez, uma submissão de um time de criptógrafos da IBM, incluindo Horst Feistel e Don Coppersmith, foi considerada aceitável. Após algumas modificações pela National Security Agency americana (NSA), a cifra foi publicada como Data Encryption Standard (NIST, 1999). Embora tenha sido publicado como um padrão para uso governamental, o respaldo do governo americano e da NSA garantiram sua adoção em aplicações civis. Apesar de ser atualmente considerado inseguro devido ao uso de uma chave muito curta (56 bits), e de ter sido oficialmente substituído pelo AES após a publicação deste como padrão em 2001, o DES ainda é amplamente usado.

2.1.1 Estrutura do DES

O DES é uma *cifra de bloco iterada* baseada na *estrutura de Feistel* (portanto, é uma *cifra de Feistel*), e sua função de rodada é uma *rede de substituição-permutação*. Estes termos são definidos a seguir.

Definição 1. *Uma cifra de bloco iterada é uma cifra de bloco definida por repetidas aplicações de uma* função de rodada *relativamente simples. Para garantir decifração única, a função de rodada deve ser bijetora.*

Definição 2. *Uma S-box é uma função sobrejetora (e possivelmente bijetora), que não necessariamente possui uma representação algébrica de fácil computação, de maneira que é concretamente implementada através de uma tabela de substituição.*

Definição 3. *Uma rede de substituição-permutação é uma topologia para funções de rodada, composta de uma aplicação de S-boxes a grupos de bits distintos da entrada, seguida por uma permutação de bits (sem restringir-se aos agrupamentos do primeiro passo).*

Definição 4. *Uma cifra de Feistel é uma cifra de bloco iterada, em que uma mensagem de $2t$ bits é dividida em duas metades (L_0, R_0) de t bits cada, e mapeada para um texto cifrado (R_r, L_r) através da iteração de r funções de rodada. A i-ésima rodada mapeia $(L_{i-1}, R_{i-1}) \rightarrow (L_i, R_i)$ de acordo com $L_i = R_{i-1}$ e $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$, onde K_i é a i-ésima subchave, derivada a partir da chave K da cifra, e F é uma função qualquer (não necessariamente bijetora) que produz valores de t bits. Este processo é ilustrado graficamente na Figura 1.*

O DES é composto de 16 rodadas, mais duas permutações no começo e no fim, a *permutação inicial* e a *permutação final* (que não possuem significância criptográfica). Uma vantagem de empregar a estrutura de Feistel é a

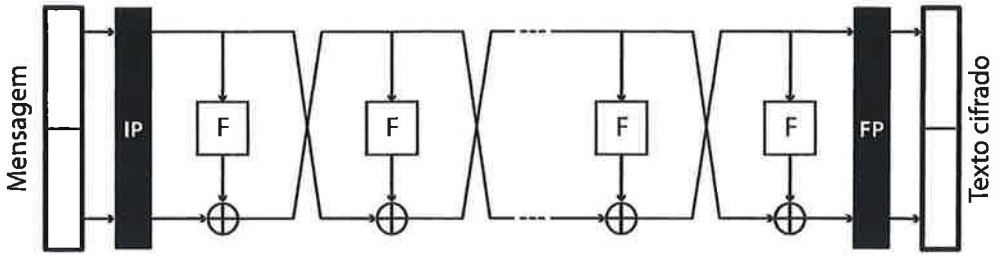

Figura 1: Estrutura de Feistel.

simetria, que permite o uso de algoritmos idênticos para cifração e decifração (o que economiza área em implementações e hardware, e memória de código em implementações em software), a menos do escalonamento de chaves, que também é o mesmo mas deve ser aplicado em ordem reversa (as subchaves são aplicadas como parte da função F). Isto é devido ao uso da operação XOR, que é auto-inversa, para misturar os valores. Por exemplo, dado L, é feito um XOR com $F(R)$ para obter $L \oplus F(R)$; esta operação pode ser cancelada por uma segunda aplicação de XOR com $F(R)$, obtendo-se L novamente. Assim, aplicando-se as subchaves em ordem reversa na decifração, a primeira rodada da decifração cancela o efeito da última rodada de cifração, a segunda rodada de decifração cancela o efeito da penúltima rodada de decifração, e assim por diante. As permutações inicial e final também são inversas uma da outra, e seus efeitos se cancelam.

2.1.2 A função F

O núcleo do DES é a função F, ilustrada na Figura 2. Esta função é essencialmente uma rede de substituição-permutação (Definição 3), e pode ser decomposta em quatro partes principais:

1. **Expansão**: o bloco é expandido de 32 para 48 bits, através da duplicação de alguns bits;

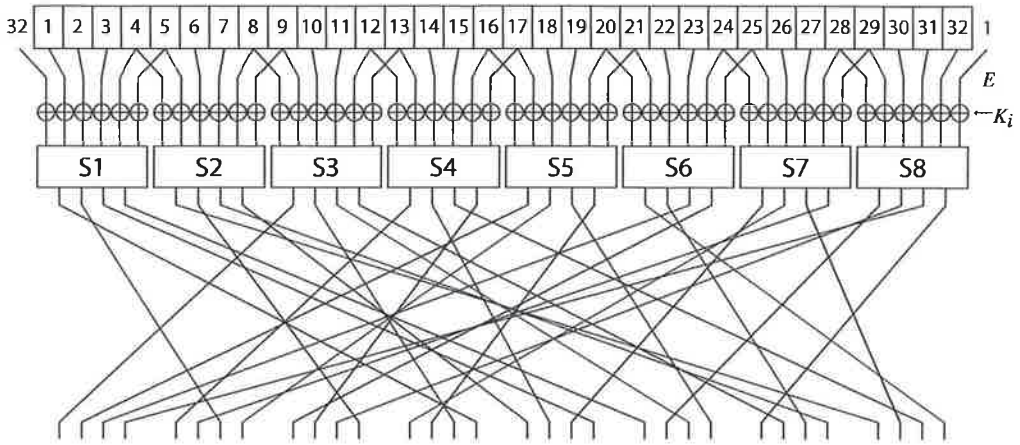

Figura 2: A função F do DES.

2. **Aplicação de subchaves**: XOR do bloco expandido com a subchave (também de 48 bits);

3. **S-boxes**: aplicação de S-boxes não lineares (8 S-boxes 6×4 distintas), produzindo um resultado de 32 bits;

4. **Permutação**: permutação de bits.

Cada passo tem uma finalidade específica, embora algumas escolhas de projeto só fiquem claras conhecendo os critérios de projeto não publicados pela IBM na época, para proteger a cifra contra criptanálise diferencial (COPPERSMITH, 1994), um ataque não publicado na literatura aberta naquele tempo. Apesar disso, alguns objetivos são claros para cada passo. O primeiro passo garante que alguns bits afetem duas S-boxes ao mesmo tempo, ao invés de uma única; isto se mostrará especialmente útil quando os ataques diferenciais forem considerados. O segundo passo modifica os bits do bloco de acordo com a chave; sem este passo, a cifração de um dado bloco de texto seria sempre igual, independente da chave. O terceiro passo, aplicação de S-boxes, é o único passo não linear da cifra, sem o qual a cifra seria linear e sua quebra seria trivial. As S-boxes estão sujeitas a diversos critérios de projeto (por

exemplo, mudar um bit na entrada deve mudar pelo menos dois bits na saída), inclusive alguns não publicados na época, que são listados em (COPPERSMITH, 1994). O quarto passo, permutação de bits, faz com que bits próximos numa rodada estejam distantes na rodada seguinte. Na Seção 2.2, outros papéis desses passos serão revelados, com o objetivo de proteger contra criptanálise diferencial, um dos critérios de projeto não-publicados quando da adoção do DES, a pedido da NSA.

2.1.3 O escalonamento de chaves

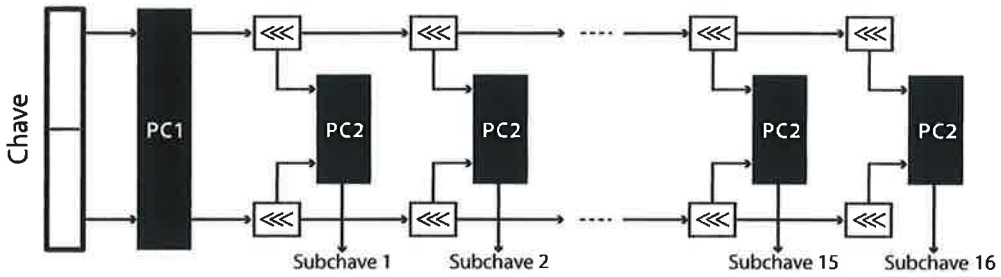

Figura 3: O escalonamento de chaves do DES.

O escalonamento de chaves do DES é ilustrado na Figura 3. Inicialmente, são selecionados 56 dos 64 bits da chave, que são divididos em duas metades de 28 bits cada. A cada rodada do escalonamento de chaves, as duas metades são submetidas a uma rotação para esquerda de 1 ou 2 bits (o valor depende da rodada), e 48 dos 56 bits são selecionados e permutados para uso como subchave da rodada correspondente da cifração ou decifração do algoritmo. Um detalhe interessante é que, ao final do escalonamento de chaves, as duas metades voltam ao estado inicial (correspondendo a uma rotação total de 28 bits). Assim, para realizar o escalonamento de chaves em ordem reversa, necessário para o algoritmo de decifração, basta substituir as rotações à esquerda por rotações à direita, e não aplicar rotação alguma durante a primeira rodada.

2.2 Criptanálise diferencial

A técnica de criptanálise diferencial foi publicada na literatura aberta pela primeira vez em 1990 por Biham e Shamir (BIHAM; SHAMIR, 1991; BIHAM; SHAMIR, 1992); porém, após a publicação da técnica, Don Coppersmith revelou (COPPERSMITH, 1994) que a técnica já era conhecida na IBM em 1974, durante o projeto do DES, mas que não foi revelada a pedido da NSA, por ser uma técnica extremamente poderosa, e sua publicação poderia colocar a segurança nacional em risco. Coppersmith inclusive revelou os reais critérios de projeto das S-boxes e da permutação de bits da função F do DES, para proteger o mesmo contra ataques diferenciais, e independente das afirmações dos projetistas, a resistência exemplar do DES à técnica de criptanálise diferencial é evidência muito forte de que seus projetistas conheciam esta técnica. O melhor ataque diferencial contra o DES (BIHAM; SHAMIR, 1992) exige 2^{47} textos escolhidos, o que torna o ataque logisticamente bem mais complexo que busca exaustiva, embora esta última exija até 2^{55} operações utilizando a propriedade de complementaridade do DES (na média, metade deste valor). Ainda assim, a existência de um ataque mais eficiente que busca exaustiva é indesejável e o DES é considerado quebrado.

Técnicas típicas de criptanálise consideram a operação da cifra sobre mensagens individuais, ou as estatísticas de um conjunto de mensagens. A criptanálise diferencial concentra-se ao invés na análise de *pares* de mensagens, e especificamente a *diferença* entre eles, e como essa diferença é alterada pela aplicação dos diferentes componentes da cifra.

Definição 5. *Sejam duas mensagens m_1, m_2. Define-se a diferença Δm com relação a uma certa operação (será adotada a operação $\oplus$ no restante deste trabalho, embora outras operações sejam mais úteis em outros contextos)*

como

$$\Delta m = m_1 \oplus m_2.$$

A vantagem de estudar diferenças é que Δm é invariante (ou modificado de maneira simples e previsível) por muitos dos componentes do DES e de outras cifras; em outros componentes, a propagação de diferenças ocorre com certas probabilidades. Em particular, como será visto, diferenças são invariantes mediante a aplicação de subchaves através de XOR. Consideremos o efeito de cada estrutura do DES sobre diferenças. Primeiramente, na estrutura de Feistel, onde as duas metades L, R são misturadas através da operação XOR, sejam L_1, L_2 e R_1, R_2 os componentes em cada caso, com diferenças $\Delta L = L_1 \oplus L_2$ e $\Delta R = R_1 \oplus R_2$, temos:

$$\Delta L \oplus \Delta R = (L_1 \oplus L_2) \oplus (R_1 \oplus R_2) = (L_1 \oplus R_1) \oplus (L_2 \oplus R_2) = \Delta(L \oplus R).$$

Assim, se ΔL e ΔR são conhecidos, então $\Delta(L \oplus R)$ pode ser determinado, sem incerteza alguma. Em particular, se $R_1 = R_2$ (resp. $L_1 = L_2$), temos que $\Delta R = 0$ (resp. $\Delta L = 0$) e portanto $\Delta L \oplus \Delta R = \Delta L$ (resp. $\Delta L \oplus \Delta R = \Delta R$). Nos componentes da função F, temos:

1. **Expansão**: a expansão apenas duplica alguns bits do bloco, realizando a mesma operação sobre qualquer entrada; assim, o efeito sobre a diferença é previsível, consistindo na duplicação dos mesmos bits da diferença;

2. **Aplicação de subchaves**: considere um par de blocos expandidos x_1, x_2 com diferença Δx. A aplicação da subchave k em ambos os blocos não tem efeito algum sobre Δx; de fato,

$$\Delta(x \oplus k) = (x_1 \oplus k) \oplus (x_2 \oplus k) = (x_1 \oplus x_2) \oplus (k \oplus k) = x_1 \oplus x_2 = \Delta x.$$

3. **Permutação**: a permutação reordena os bits da mesma maneira, independente da entrada; o efeito sobre a diferença é previsível, consistindo na reordenação dos bits da diferença seguindo o mesmo padrão.

Observa-se que as operações listadas até agora modificam as diferenças de maneira previsível, e se fossem as únicas operações do DES, o mesmo seria quebrado trivialmente pelo estudo das diferenças. A única operação que adiciona alguma incerteza é a aplicação de S-boxes: uma dada diferença de saída na S-box pode ser provocada por mais que uma diferença de entrada, cada uma com uma determinada probabilidade. A única diferença de entrada que garantidamente se propaga para uma diferença fixa é 0; se as entradas são iguais, as saídas também serão. Pode-se construir uma *tabela de distribuição de diferenças* da S-box, contando para cada diferença de entrada a quantidade de vezes que uma determinada diferença de saída é produzida. Assumindo que os dados são distribuídos de forma aleatória, a estatística produzida é a probabilidade de propagação de uma diferença. Por exemplo, para a primeira S-box do DES, a diferença de entrada $\Delta I = 34\mathrm{h}$ produz a diferença de saída $\Delta O = 02\mathrm{h}$ em $16/64 = 1/4$ dos casos.

Segue que o segredo de um ataque diferencial no DES é encontrar uma diferença de entrada que se propague com grande probabilidade para uma determinada diferença de saída através da cifra quase inteira, conhecido como uma *característica*. Para maximizar as probabilidades, deve-se garantir que o máximo possível de diferenças (porém não todas) sejam 0, já que estas propagam com probabilidade 1. Alguns critérios de projeto do DES tem objetivo específico de dificultar ataques diferenciais: por exemplo, um dos critérios de projeto das S-boxes é que a probabilidade de propagação de todas as diferenças não exceda $1/4$, enquanto um dos critérios de projeto da permutação é que os 4 bits de saída de cada S-box afetem, na próxima rodada, 6 diferentes

S-boxes (lembrando que alguns bits são duplicados pelo passo de expansão). Assim, uma dada S-box *ativa*, ou seja, com diferença não-nula, poderá forçar diferenças não-nulas em até 6 outras S-boxes.

Resta saber como extrair informações sobre a chave utilizando este ataque. Na verdade, o que se obtém são bits da subchave da última rodada; no caso do DES, esses bits possuem uma relação direta com a chave em si, e os bits restantes da chave podem ser descobertos por busca exaustiva. Devido à estrutura do DES, é possível calcular, para a última rodada de cifração, os bits que serão misturados com a subchave e fornecidos às S-boxes. No sentido oposto, dada a diferença de saída da característica, lista-se as diferenças de entrada admissíveis para esta diferença de saída (nem todas as diferenças de entrada produzem uma certa diferença de saída). Juntando estas duas informações, é possível listar os bits da subchaves que geram as entradas necessárias para produzir a diferença de saída especificada. Certos pares, chamados de *pares incorretos*, não seguirão a característica especificada e sugerirão subchaves aleatórias, enquanto que os *pares corretos*, que seguem a característica, sempre sugerirão a subchave correta (além de outras erradas). Assim, a subchave correta será sugerida mais frequentemente que as subchaves erradas; a frequência exata depende da probabilidade da característica e do número de bits da subchave a serem contados – quanto mais bits, maior o espaço de subchaves, 'espalhando' mais os erros e garantindo que a subchave correta se sobressairá mais facilmente.

Um projetista que deseje proteger uma cifra de estrutura similar ao DES contra criptanálise diferencial deve tomar duas medidas principais: limitar a probabilidade de propagação de diferenças nas S-boxes, e garantir o máximo possível de S-boxes ativas por rodada. A primeira propriedade depende da escolha de boas S-boxes, mas a segunda exige uma construção apropriada

da cifra inteira, a exemplo de construção proposta pela estratégia de trilha larga.

2.3 Criptanálise linear

A técnica de criptanálise linear foi publicada por Mitsuru Matsui em 1992 (MATSUI, 1994b; MATSUI, 1994a). Ao contrário do ataque diferencial, não há evidência de que o DES tenha sido reforçado contra esse ataque; observa-se que mudanças aleatórias simples no DES enfraquecem-no contra criptanálise diferencial, mostrando o cuidado que os projetistas tiveram em protegê-lo contra este ataque, porém o mesmo não ocorre com relação a criptanálise linear. Apesar disso, o melhor ataque linear contra o DES, descrito nos artigos citados, não é muito mais eficiente que o melhor ataque diferencial, exigindo 2^{43} textos conhecidos.

A idéia do ataque é tentar aproximar a operação da cifra por uma expressão linear.

Definição 6. *Seja $A[i]$ o i-ésimo bit de um valor A. Uma* paridade *de A, associada a um* padrão de seleção *de bits $i_1, i_2, \ldots, i_n$, é definida como*

$$A[i_1, i_2, \ldots, i_n] = A[i_1] \oplus A[i_2] \oplus \ldots \oplus A[i_n].$$

Definição 7. *Uma expressão linear para uma cifra relaciona paridades especificadas da mensagem P, texto cifrado C e chave K da cifra. Matematicamente,*

$$P[i_1, i_2, \ldots, i_a] \oplus C[j_1, j_2, \ldots, j_b] = K[k_1, k_2, \ldots, k_c]. \tag{2.1}$$

Se uma expressão linear estivesse sempre correta para uma escolha apropriada dos conjuntos de bits, a expressão linear seria uma representação alternativa da cifra (ao menos para os bits em questão) e a cifra seria trivialmente

quebrada. Da mesma forma, se a expressão estivesse sempre errada, então uma expressão similar, em que um dos lados da equação é negado, também seria uma representação alternativa da cifra. Assim, espera-se de uma cifra forte que a probabilidade da expressão estar certa ou errada seja aproximadamente igual a $1/2$ em cada caso, ou seja, que a expressão seja tão ineficaz em prever o comportamento da cifra quanto um evento aleatório como o lançamento de uma moeda.

Definição 8. *A eficácia de uma expressão linear, válida com probabilidade p, é dada por seu desvio da probabilidade ideal $1/2$; matematicamente,*

$$\epsilon = |p - 1/2|.$$

Este valor é tomado em módulo; o sentido do desvio é irrelevante para a eficácia.

Se o criptanalista obtiver uma expressão linear para a cifra com um valor 'significativo' de ϵ, a cifra pode ser quebrada. (Como exemplo, a melhor expressão para 14 rodadas do DES possui $\epsilon = 1,19 \times 2^{-21}$.)

Para montar ataques lineares bem sucedidos, é necessário um método para obtenção de expressões lineares eficazes. No caso do DES, observa-se que todas as operações da cifra, à exceção das S-boxes, são lineares; a incorporação destas operações no DES não reduz a eficácia das expressões lineares construídas. Resta saber como construir expressões lineares eficazes para as S-boxes. Para tanto, constrói-se uma *tabela de aproximações lineares para a S-box*, contendo todos os padrões de seleção de bits para a entrada e saída da S-box, e para cada um, listando o valor de ϵ (obtido por enumeração dos casos possíveis). Analogamente à técnica de criptanálise diferencial, cada S-box ativa exige uma aproximação linear (contribuindo para a redução de ϵ), enquanto que S-boxes inativas não afetam a eficácia da apro-

ximação; assim, a minimização do número de S-boxes ativas é crucial para o sucesso do ataque.

A princípio, uma expressão como (2.1) fornece um único bit de informação sobre a chave, a saber, a paridade do conjunto de bits selecionados para a chave. Uma modificação do procedimento permite a extração de mais bits de informação, o que torna o ataque prático. Ao invés de construir uma aproximação linear para a cifra inteira, é feita uma aproximação para a cifra sem a última rodada. Essa aproximação emprega alguns bits da subchave da última rodada; o criptanalista efetua uma decifração parcial da última rodada, considerando apenas as S-boxes ativas e as subchaves associadas a essas S-boxes. Como as subchaves são a priori desconhecidas, as decifrações são realizadas para todas as subchaves possíveis, e para cada subchave testada, um contador associado é incrementado caso a aproximação linear empregada esteja correta para esta combinação de mensagem, texto cifrado e subchave conjecturada. Para as subchaves incorretas, a aproximação linear é inválida e espera-se uma proporção aproximadamente igual de aproximações corretas ou incorretas, enquanto que para a subchave correta, há um desvio sistemático de ϵ na probabilidade de a expressão estar correta. Obviamente que, devido à pequena magnitude de ϵ numa cifra bem projetada, e à variância inerente a qualquer processo aleatório, espera-se que seja necessária uma grande quantidade de textos para observar o desvio; de fato, uma decisão confiável só pode ser tomada após o processamento de uma quantidade de textos da ordem de $1/\epsilon^2$.

No projeto de uma nova cifra, a proteção contra criptanálise linear, em analogia com a proteção contra criptanálise diferencial, pode ser obtida através de duas medidas: limitando a magnitude dos desvios na tabela de aproximação linear das S-boxes e garantindo o máximo de S-boxes ativas por rodada; am-

bas as medidas são adotadas pela estratégia de trilha larga.

2.4 Arcabouços formais para criptanálise diferencial e linear

Uma das contribuições da estratégia de trilha larga é um novo formalismo para os métodos de criptanálise diferencial e linear, que trata ambos os métodos de maneira similar, até parcialmente unificando os mesmos, e remove algumas aproximações e hipóteses heurísticas dos formalismos originais. Este formalismo será exposto aqui como preparação para a Seção 2.5, que ilustra a estratégia de trilha larga e justifica as decisões de projeto com base neste formalismo.

2.4.1 Criptanálise diferencial

O formalismo para criptanálise diferencial utilizado na estratégia de trilha larga é similar ao desenvolvido anteriormente na literatura, porém algumas modificações e definições extras são necessárias.

Seja um par de vetores de n bits a, a^* com diferença (sob a operação XOR) $a' = a \oplus a^*$, e seja uma função h tal que $b = h(a), b^* = h(a^*)$ e $b' = b \oplus b^*$. Diz-se que a diferença a' é propagada para a diferença b' através de h. Deve-se notar que em geral, a' pode se propagar para diversas possibilidades de b', dependendo das escolhas para a e a^* (mantendo a' fixo ainda) e não apenas de a'. A cada propagação de diferenças possível, é associada uma probabilidade, definida a seguir. Observa-se que, para certos pares (a', b'), essa probabilidade pode ser nula, e diz-se então que a' não pode se propagar para b' através de h.

Definição 9. *Uma probabilidade de propação de diferenças* $Pr^h(a', b')$ *é defi-*

nida como

$$Pr^h(a', b') = 2^{-n} \sum_a \delta(h(a) \oplus h(a \oplus a') \oplus b'). \tag{2.2}$$

onde δ é a função de Kronecker

$$\delta(x) = \begin{cases} 1 & \text{se } x = 0, \\ 0 & \text{caso contrário.} \end{cases}$$

Intuitivamente, (2.2) conta quantas vezes uma diferença b' dada coincide com a diferença de saída produzida pelo par de entradas a e $a^* = a \oplus a'$, para todas as 2^n possibilidades para a, e divide por este números de possibilidades. Admitindo uma diferença fixa a' e que a seja sorteado de acordo com uma distribuição uniforme, então $Pr^h(a', b')$ é a probabilidade de que $b' = h(a) \oplus h(a^*)$.

Uma métrica relacionada, porém mais útil para a discussão sobre a estratégia de trilha larga, é o *peso* de uma propagação de diferenças.

Definição 10. *O peso de uma propagação de diferenças (a', b') é o negativo do logaritmo (em base 2) da probabilidade de propagação de diferenças. Matematicamente,*

$$w(a', b') = -\log_2 Pr^h(a', b').$$

Intuitivamente, pode-se pensar no peso de uma propagação de diferenças como o número de bits de informação que essa propagação de diferenças fornece sobre a.

Introduz-se agora o conceito crucial de *trilha diferencial*.

Definição 11. *Seja $\beta : GF(2)^n \to GF(2)^n$ uma função sobre vetores de n bits, que pode ser descrita como uma composição de r funções (por exemplo, r funções de rodada de uma cifra): $\beta = \rho^{(r)} \circ \cdots \circ \rho^{(1)}$. Uma trilha diferencial A*

consiste de uma sequência de $r + 1$ *padrões de diferenças*

$$Q = (q^{(0)}, q^{(1)}, \ldots, q^{(r-1)}, q^{(r)}),$$

onde a trilha é seguida se, para $q^{(i)} = q_1^{(i)} \oplus q_2^{(i)}$, *temos*

$$\begin{aligned}
\rho^{(1)}(q_1^{(0)}) \oplus \rho^{(1)}(q_2^{(0)}) &= q^{(1)}, \\
\rho^{(2)}(q_1^{(1)}) \oplus \rho^{(2)}(q_2^{(1)}) &= q^{(2)}, \\
&\ldots \\
\rho^{(r-1)}(q_1^{(r-2)}) \oplus \rho^{(r-1)}(q_2^{(r-2)}) &= q^{(r-1)}, \\
\rho^{(r)}(q_1^{(r-1)}) \oplus \rho^{(r)}(q_2^{(r-1)}) &= q^{(r)},
\end{aligned}$$

Essa trilha é composta por r passos diferenciais $(q^{(i-1)}, q^{(i)})$, *cada qual com probabilidade de propagação* $Pr^{\rho^{(i)}}(q^{(i-1)}, q^{(i)})$.

A uma trilha diferencial, associa-se sua probabilidade de ocorrência, que é o número de textos de entrada $a^{(0)}$ tais que a trilha diferencial é seguida, dividida pelo número de possibilidades para $a^{(0)}$.

Um ataque de criptanálise diferencial sucede se é possível propagar uma diferença $(q^{(0)}, q^{(r)})$ com alta probabilidade. Esta propagação não ocorre por uma única trilha, mas sim por todas as trilhas com diferença inicial $q^{(0)}$ e diferença final $q^{(r)}$, embora geralmente o criptanalista concentre-se numa trilha específica, com probabilidade mais alta que outras trilhas de mesma diferença inicial e final, e desconsidere o efeito das demais trilhas. Mesmo assim, deve-se definir a probabilidade de propagação de uma diferença pela cifra inteira como a soma das probabilidades para todas as trilhas possíveis, ou seja,

$$\Pr(a', b') = \sum_{q^{(0)}=a', q^{(r)}=b'} \Pr(Q). \tag{2.3}$$

Analogamente ao caso de propagação de diferenças, pode-se definir o peso de uma trilha diferencial.

Definição 12. *O peso de uma trilha diferencial Q é a soma dos pesos de seus passos diferenciais, ou seja,*

$$w(Q) = \sum_{i=1}^{r} w^{\rho(i)}(q^{(i-1)}, q^{(i)}).$$

2.4.2 Criptanálise linear

No caso da criptanálise linear, o formalismo usado para a mesma na estratégia de trilha larga é bastante distinto do formalismo amplamente adotado na literatura, utilizando a noção de *correlação* ao invés de probabilidade e desvio de uma aproximação linear.

Definição 13. *A correlação entre duas funções booleanas $f(a)$ e $g(a)$ é definida como*

$$C(f,g) = 2Pr(f(a) = g(a)) - 1.$$

Intuitivamente, a correlação de duas funções booleanas é a contagem de quantas vezes estas funções coincidem, ajustada para o intervalo $[-1,+1]$, onde -1 indica correlação negativa máxima ($f(a) \neq g(a)$ sempre), $+1$ indica correlação positiva máxima ($f(a) = g(a)$ sempre), e 0 indica correlação nula ($f(a) = g(a)$ em exatamente metade dos casos, e $f(a) \neq g(a)$ na outra metade).

Como no caso de propagações de diferenças, pode-se definir um *peso* de uma correlação.

Definição 14. *O peso de uma correlação $C(f,g)$ é o negativo do logaritmo (em base 2) de sua amplitude em módulo. Matematicamente,*

$$w(f,g) = -\log_2 |C(f,g)|.$$

O conceito de paridade e padrão de seleção de bits da Definição 6 pode ser formalizado de maneira diferente. Para um padrão de seleção de bits, associa-se um vetor booleano w com bits 1 nas posições selecionadas e 0 nas posições não selecionadas. Assim, a paridade de um vetor a pode ser calculada pelo produto interno $w^T a$, onde w^T é a transposição de w. Fica claro então que o número de paridades possíveis é o número de vetores possíveis w, que é 2^n se o vetor possui n componentes.

É possível agora definir o conceito de *trilha linear*.

Definição 15. *Seja $\beta, r, \rho^{(r)}, \ldots, \rho^{(1)}$ como na Definição 11. Uma trilha linear U consiste de uma sequência de $r+1$ padrões de seleção*

$$U = (u^{(0)}, u^{(1)}, \ldots, u^{(r-1)}, u^{(r)}),$$

composto de r passos lineares $(u^{(i-1)}, u^{(i)})$, cada qual com correlação

$$C(u^{(i-1)^T} a, u^{(i)^T} \rho^{(i)}(a)).$$

A contribuição de correlação C_p de uma trilha linear é o produto da correlação de todos os seus passos:

$$C_p(U) = \prod_{i=1}^{r} C(u^{(i-1)^T} a, u^{(i)^T} \rho^{(i)}(a)).$$

Analogamente ao peso de uma correlação, pode-se definir o peso de uma trilha linear.

Definição 16. *O peso de uma trilha linear U é a soma dos pesos de seus passos lineares, ou seja,*

$$w(Q) = \sum_{i=1}^{r} w^{\rho(i)}(u^{(i-1)^T} a, u^{(i)^T} \rho^{(i)}(a)).$$

2.5 A estratégia de trilha larga

A estratégia de trilha larga (RIJMEN, 1997; DAEMEN, 1995; DAEMEN; RIJMEN, 2001a; DAEMEN; RIJMEN, 2001b) é uma metodologia de projeto de cifras de bloco eficientes e demonstravelmente resistentes a criptanálise diferencial e linear. O nome da estratégia refere-se ao fato que, nas cifras projetadas de acordo com esta estratégia, as trilhas diferenciais e lineares são 'largas', no sentido de se estenderem a uma grande quantidade de S-boxes ativas (termo definido na Seção 2.5.2.3); a presença de muitas S-boxes ativas é uma das características principais da estratégia. Os fundamentos e critérios de projeto desta estratégia serão discutidos ao longo do restante do capítulo. Esta seção é fortemente inspirada no Capítulo 9 de (DAEMEN; RIJMEN, 2001a), de maneira que muitas referências a este material ao longo do capítulo foram omitidas.

2.5.1 Um modelo para cifras de bloco resistentes a criptanálise diferencial e linear

Nesta seção, será discutido o modelo para cifras de bloco adotado na estratégia de trilha larga, e como este modelo é uma restrição do modelo geral para uma cifra de bloco. Será argumentado que cifras dentro deste modelo podem ser analisadas mais facilmente com relação à sua resistência a ataques diferenciais e lineares.

Em geral, uma cifra de bloco é uma função, parametrizada por uma chave k, que converte blocos de mensagens de tamanho n_b bits para blocos de texto cifrado do mesmo tamanho. Uma cifra de bloco iterada (Definição 1) é da forma

$$\beta[k] = \rho^{(r)}[k^{(r)}] \circ \cdots \circ \rho^{(1)}[k^{(1)}], \tag{2.4}$$

onde $\rho^{(i)}$ é a função de rodada relativa à i-ésima rodada, e $k^{(i)}$ é a subchave

da i-ésima rodada. Neste modelo, a função de rodada é a mesma para todas as rodadas, e diz-se então que há uma única função de rodada. As subchaves são calculadas a partir da chave principal através do escalonamento de chaves da cifra.

2.5.1.1 Cifras de bloco com alternância de chaves

Uma restrição ao modelo geral é a propriedade de *alternância de chaves*.

Definição 17. *Uma cifra de bloco com alternância de chaves é uma cifra de bloco iterada, com as seguintes propriedades adicionais:*

- **alternância**: *a função de rodada da cifra é decomposta em duas partes, uma função de rodada propriamente dita, independente da chave, e a aplicação da subchave em questão.*
- **aplicação de chaves por XOR**: *as subchaves são aplicadas ao estado da cifra através da função XOR.*

Matematicamente, restringe-se (2.4) à forma

$$\beta[k] = \sigma[k^{(r)}] \circ \rho^{(r)} \circ \sigma[k^{(r-1)}] \circ \cdots \circ \sigma[k^{(1)}] \circ \rho^{(1)} \circ \sigma[k^{(0)}].$$

Uma vantagem de cifras com alternância de chaves é que a análise da resistência à criptanálise diferencial e linear é mais simples, como ficará claro em seguida. Pode-se adiantar que isto é devido ao isolamento entre a função de rodada propriamente dita (que obrigatoriamente deve conter alguma forma de não-linearidade) da aplicação de subchaves, e que esta aplicação é feita através de uma operação linear, ou seja, pode ser ignorada na construção de um ataque diferencial ou linear – efetivamente, o efeito da chave pode ser

ignorado na construção destes ataques. Note que isto não facilita um ataque (no sentido de reduzir seu fator de trabalho), apenas a construção e análise do mesmo.

2.5.1.2 A estrutura de rodada $\gamma\lambda$

Na estratégia de trilha larga, a função de rodada é restrita à composição de dois passos invertíveis:

- γ: uma transformação não-linear local (local no sentido de que bits próximos na saída só são afetados por bits próximos na entrada – especificamente, bits próximos referem-se a bits no mesmo *grupo*, expressão definida a seguir);
- λ: uma transformação linear que provê alta difusão[1].

Matematicamente, a função de rodada ρ é dada por

$$\rho = \lambda \circ \gamma.$$

A construção mais simples para γ é pelo uso de S-boxes invertíveis. Nesta construção, os bits do vetor de entrada a são particionados em n_t *grupos* $a_i \in \mathrm{GF}(2)^m$ (ou seja, de m bits cada) com $i \in \mathcal{I}$, onde $\mathcal{I}$ é o *espaço de índices*. O tamanho de bloco da cifra é relacionado com m e n_t por $n_b = mn_t$. A maioria dos projetos baseados na estratégia de trilha larga (incluindo a cifra Rijndael) tomam $m = 8$, mas este não é um requerimento; por exemplo, FUTURE (Capítulo 6) possui $m = 4$.

Esta estrutura já impõe uma primeira restrição sobre o estado da cifra:

[1] Difusão pode ter diversos sentidos nesse contexto; o sentido empregado na estratégia de trilha larga é definido na Seção 2.5.3.

Definição 18. *O estado de uma cifra com estrutura de rodada $\gamma\lambda$, composto por n_b bits, será considerado como n_t grupos de m bits cada, com n_t, m tais que $n_b = mn_t$.*

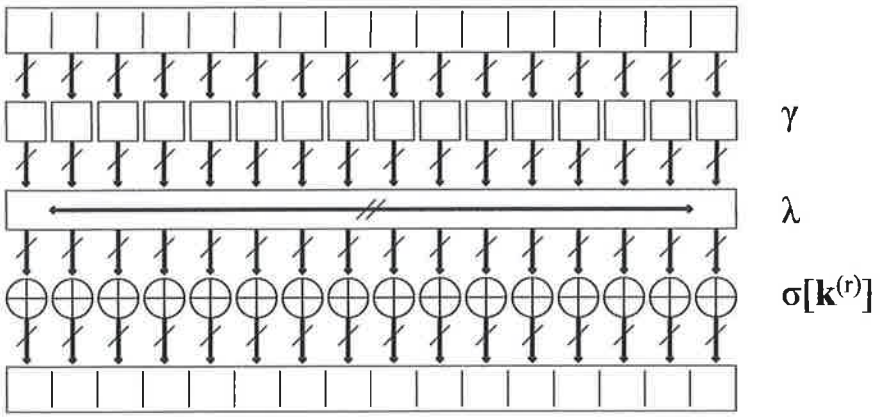

Figura 4: Estrutura de rodada $\gamma\lambda$.

É possível usar S-boxes diferentes para grupos diferentes, mas isto não resulta em uma melhoria quantificável da segurança da cifra, e tem a desvantagem de exigir mais espaço para armazenamento de tabelas em software, e restringir o tamanho de implementações compactas em hardware. Portanto, admite-se daqui em diante que será usada a mesma S-box para todos os grupos.

A transformação linear λ também opera sobre grupos, de modo que cada grupo na saída é uma função linear de alguns dos (ou todos os) grupos da entrada. A nível de bit, λ pode ser especificada como uma matriz binária M de dimensão $n_b \times n_b$:

$$\lambda : b = \lambda(a) \Leftrightarrow b = Ma.$$

λ também pode ser especificada a nível de grupo. Tipicamente, na estratégia de trilha larga, os grupos são considerados como elementos de GF(2^m).

2.5.2 Propagação de diferenças e correlações em cifras com alternância de chaves

Nesta seção, será feita uma análise de como diferenças (em um ataque diferencial) ou correlações (em um ataque linear) se propagam através de uma cifra que se encaixa no modelo estudado. É recomendada uma consulta à Seção 2.4 para familiarização com a notação e os conceitos empregados.

2.5.2.1 Obtenção de baixas probabilidades de propagação de diferenças

Para montar um ataque diferencial com sucesso, o criptanalista deve fornecer uma diferença de entrada que se propaga para uma diferença de saída através da cifra quase inteira, com uma probabilidade bastante superior a 2^{-n_b} (caso contrário, é mais simples obter o livro de códigos completo da cifra). O projetista da cifra deve então construir a função de rodada e especificar um número de rodadas, de modo que não existam trilhas diferenciais com probabilidade superior a 2^{-n_b}.

No entanto, isto não significa que uma diferença de entrada não possa se propagar para uma diferença de saída com alta probabilidade; de fato, isto deve ocorrer, uma vez que uma diferença de entrada i' deve se propagar para alguma diferença de saída o', e mesmo no caso ideal em que as diferenças fossem uniformemente distribuídas, estas já ocorreriam com probabilidade 2^{-n_b}; na prática, existem diferenças que ocorrem com probabilidade muito maior. Isto não é incompatível com a afirmação do parágrafo anterior: em geral, uma diferença de entrada não se propaga para uma diferença de saída através de uma única trilha, mas sim de diversas trilhas, e as probabilidade destas trilhas são somadas conforme a equação (2.3), sem impor restrições sobre a probabilidade de uma trilha específica.

Deve-se lembrar que a propagação de uma diferença não depende apenas da diferença em si, e sim dos elementos do par que gera aquela diferença. Portanto, a aplicação de subchaves na cifra, embora não influencie as diferenças em si, influencia os elementos componentes das diferenças, e assim, uma dada trilha pode ser seguida para um valor da chave mas não para outro valor diferente. A identificação e uso destas trilhas dependentes de chaves num ataque diferencial é praticamente inviável mesmo quando a chave é conhecida, e mais ainda quando não é. Já trilhas fixas e independentes de chaves podem estar sujeitas à restrição do primeiro parágrafo. Conclui-se então que restringir a probabilidade associada às trilhas diferenciais é uma boa estratégia.

2.5.2.2 Obtenção de baixas amplitudes de correlação

Para montar um ataque linear com sucesso, o criptanalista deve fornecer uma paridade de entrada correlacionada a uma paridade de saída através da cifra quase inteira, com uma amplitude de correlação bastante superior a $2^{-n_b/2}$ (caso contrário, é mais simples obter o livro de códigos completo da cifra). O projetista da cifra deve então construir a função de rodada e especificar um número de rodadas, de modo que não existam trilhas lineares com contribuição de correlação superior a $2^{-n_b}/n_k$, onde n_k é o número de bits da chave.

No entanto, isto não significa que não existam paridades de entrada correlacionadas a paridades de saída com altas amplitudes; de fato, pelo teorema de Parseval (DINIZ; SILVA; NETTO, 2002), para uma dada paridade de saída, a soma do quadrado das amplitudes das correlações com as paridades de entrada deve ser 1, e mesmo no caso ideal em que a paridade de saída seja igualmente correlacionada a todas as 2^{n_b} paridades de entrada, as amplitudes já seriam $2^{-n_b/2}$; na prática, existem correlações que ocorrem com amplitude muito maior. Isto não é incompatível com a afirmação do parágrafo anterior:

em geral, uma paridade de entrada não é correlacionada com uma paridade de saída através de uma única trilha, mas sim pela interferência construtiva de diversas trilhas com a mesma paridade de entrada e saída.

No entanto, a interferência construtiva de trilhas impõe uma relação linear sobre os bits da chave, e portanto as trilhas que contribuem para uma determinada correlação para um valor da chave muito provavelmente não contribuirão para um valor diferente da chave. A identificação e uso destas trilhas dependentes de chaves num ataque linear é praticamente inviável mesmo quando a chave é conhecida, e mais ainda quando não é. Já trilhas fixas e independentes de chaves podem estar sujeitas à restrição do primeiro parágrafo. Conclui-se então que restringir a probabilidade associada às trilhas lineares é uma boa estratégia.

2.5.2.3 Trilhas largas

Quando uma diferença de saída ou padrão de seleção de saída num grupo ou S-box é não-nula, diz-se que esse grupo ou S-box está *ativo(a)*. O número de grupos ativos numa diferença ou padrão de seleção é dito o *peso do grupo*, e é denotado por $w_b(a)$[2]. Quando aplicado a um padrão de diferença a', $w_b(a')$ é o número de grupos ativos em a', e quando aplicado a um padrão de seleção v, $w_b(v)$ é o número de grupos ativos em v. O peso de grupo de uma trilha é a soma dos pesos de grupo para os padrões de diferença ou amplitudes de correlação dos componentes da trilha. Um limite inferior para o peso de uma trilha diferencial ou linear é o produto do peso de grupo pelo peso mínimo (diferencial ou linear) da S-box, dado pelo mínimo, tomado sobre todas as diferenças (resp. correlações) possíveis, do peso diferencial (resp. linear) da S-box.

[2]O índice b refere-se a *bundle*, o termo em inglês para 'grupo'.

Existem dois caminhos possíveis para eliminar trilhas diferenciais ou lineares de baixo peso:

1. Escolher S-boxes com altos pesos diferenciais ou lineares mínimos;
2. Projetar a função de rodada de maneira a garantir altos pesos mínimos de grupo.

Pode-se mostrar que o limite superior para o peso diferencial de uma S-box invertível $m \times m$ é $m-2$, e para o peso de correlação, $m/2$ (DAEMEN; RIJMEN, 2001a). Por este motivo, para tomar o primeiro caminho sugerido, é necessário despender recursos em S-boxes grandes. Na estratégia de trilha larga, toma-se o segundo caminho, despendendo recursos no passo linear para garantir altos pesos mínimos de grupo.

2.5.3 Difusão

Difusão é o termo usado por Claude Shannon para indicar o espalhamento de informação (SHANNON, 1949). Pode-se dizer que a transformação γ realiza uma espécie de difusão entre os bits de um grupo; porém, esta difusão é confinada a cada grupo e não provê nenhuma interação inter-grupo. Na estratégia de trilha larga, esse tipo de difusão é irrelevante, e o que interessa é apenas a difusão que aumenta o peso mínimo de grupo de trilhas diferenciais e lineares, que é realizada somente por λ.

A difusão de uma trilha através de uma única rodada de uma cifra é igual ao peso de grupo do estado da cifra naquela rodada, e um limite inferior para este peso é 1 – certamente é possível ter um único grupo ativo numa dada rodada. Já para duas rodadas, é necessário somar o peso de grupo na entrada da primeira rodada e da segunda rodada. É possível obter um limite

inferior para a difusão nas cifras baseadas na estrutura $\gamma\lambda$, mas para tanto, é necessário primeiro definir um conceito preciso de difusão.

2.5.3.1 O número de ramificação

Pode-se agora definir o conceito de *número de ramificação*, fazendo a distinção entre o número de ramificação diferencial e linear.

Definição 19. *O número de ramificação diferencial de uma transformação ϕ é dado por*

$$\mathcal{B}_d(\phi) = \min_{a,a*\neq a}\{w_b(a \oplus a*) + w_b(\phi(a) \oplus \phi(a*))\}.$$

Em particular, para uma transformação linear λ, temos que $\lambda(a) \oplus \lambda(a*) = \lambda(a \oplus a*)$, e assim

$$\mathcal{B}_d(\phi) = \min_{a'\neq 0}\{w_b(a') + w_b(\lambda(a'))\}.$$

Definição 20. *O número de ramificação linear de uma transformação ϕ é dado por*

$$\mathcal{B}_l(\phi) = \min_{\alpha,\alpha^*,C(\alpha^T x,\alpha*^T\phi(x))\neq 0}\{w_b(\alpha) + w_b(\alpha*)\},$$

onde α, β variam sobre os possíveis padrões de seleção de bits. Em particular, para uma transformação linear λ, existe uma matriz M tal que $\lambda(x) = Mx$, e assim

$$\mathcal{B}_l(\phi) = \min_{\alpha\neq 0}\{w_b(\alpha) + w_b(M^T\alpha)\}.$$

Observa-se uma certa simetria entre as definições para o caso em que a transformação é linear: o número de ramificação diferencial para uma transformação λ especificada por uma matriz M é equivalente ao número de ramificação linear para uma transformação especificada por M^T. Em geral, os números de ramificação diferencial e linear podem ser diferentes, mas uma condição suficiente para que sejam iguais é adotar uma matriz simétrica (em que $M = M^T$).

Muitas das discussões seguintes são válidas tanto para números de ramificação diferencial quanto linear, e será feita então referência apenas ao número de ramificação, sem especificar o tipo, e usada a notação $\mathcal{B}$.

Seja n_α o total de grupos no estado da cifra. Um limite superior para o número de ramificação é o caso em que todos os n_α grupos estão ativos na saída, e somando ao mínimo de um grupo ativo na entrada, obtém-se

$$\mathcal{B}(\phi) \leq n_\alpha + 1. \tag{2.5}$$

Algumas propriedades de interesse sobre números de ramificação:

- Pela simetria dos termos na Definição 19, tomando o mínimo sobre $a' = \phi(a)$ e $b' = \phi(b)$ (uma permutação do conjunto original), fica claro que $\mathcal{B}_d(\phi^{-1}) = \mathcal{B}_d(\phi)$. Um argumento semelhante garante que $\mathcal{B}_l(\phi^{-1}) = \mathcal{B}_l(\phi)$;
- Um padrão diferencial ou padrão de seleção a não é afetado pela aplicação de subchaves em uma cifra com alternância de chaves, e portanto seu peso de grupo $w_b(a)$ não é afetado;
- A transformação γ, operando sobre grupos individuais, não pode transformar um grupo ativo em inativo ou vice-versa, e portanto não afeta o peso de grupo w_b.

Aplicando estas probabilidades ao peso de grupo de uma transformação de rodada ρ que segue a estrutura $\gamma\lambda$, obtém-se que o número de ramificação (linear ou diferencial) de ρ é igual ao da transformação linear λ apenas.

2.5.3.2 Propagação de trilhas em duas rodadas

O seguinte teorema ilustra a utilidade do número de ramificação, através de sua relação com o número de grupos ativos numa trilha. O teorema é válido

tanto para o número de ramificação diferencial como o linear.

Teorema 1. *Para uma cifra de bloco com alternância de chaves e estrutura de rodada $\gamma\lambda$, um limite inferior para o número de grupos ativos de qualquer trilha sobre duas rodadas é o número de ramificação de λ.*

Demonstração. O número de grupos ativos em uma trilha de duas rodadas é a soma do número de grupos ativos na entrada da primeira rodada e na entrada da segunda rodada (após a aplicação da função de rodada $\rho = \sigma[k] \circ \lambda \circ \gamma$), ou seja,

$$w_b(a) + w_b(\rho(a)). \tag{2.6}$$

Pelas propriedades da seção anterior, $\sigma[k]$ e γ não afetam o número de ramificação de ρ; portanto $\mathcal{B}(\rho) = \mathcal{B}(\lambda)$. Como por definição $\mathcal{B}(\rho)$ é o valor mínimo de (2.6), a afirmação do teorema fica provada. □

2.5.4 Estruturas eficientes para cifras com alternância de chaves

Uma estratégia sugerida pelo Teorema 1 é empregar uma transformação λ com alto número de ramificação. No entanto, tais transformações possuem alto custo computacional. Ao invés disso, propõe-se o uso de uma estrutura com duas funções de rodada distintas, que produz melhores limites inferiores para trilhas ao longo de 4 rodadas:

$$\rho^a = \theta \circ \gamma, \tag{2.7}$$

$$\rho^b = \Theta \circ \gamma. \tag{2.8}$$

Nesta estrutura, impõe-se uma restrição extra sobre o estado da cifra, originalmente definido na Definição 18.

Definição 21. *O estado da cifra para a estrutura de rodada $\rho^a\rho^b$ é definido*

como o estado da cifra na estrutura $\gamma\lambda$, adicionando-se a restrição que os grupos são dispostos em uma matriz.

2.5.4.1 A transformação de difusão θ

θ é uma transformação que agrupa os grupos do estado da cifra em uma série de colunas, através de uma partição Ξ do espaço de índices $\mathcal{I}$. Uma coluna é denotada por ξ e o número de colunas por n_Ξ. A coluna que contém o índice i é denotada por $\xi(i)$, e o número de índices em uma coluna ξ por n_ξ.

A transformação θ é local, porém local a colunas e não apenas grupos (como no caso de γ). Internamente a cada coluna, grupos são combinados de forma linear. Matematicamente,

$$\theta : b = \theta(a) \Leftrightarrow b_i = \bigoplus_{j \in \xi(i)} C_{i,j} a_j.$$

Uma maneira alternativa de representar a transformação é considerar o vetor a_ξ formado pelos grupos com índices em ξ e uma matriz C_ξ de dimensão $n_\xi \times n_\xi$. Temos

$$\theta : b = \theta(a) \Leftrightarrow b_\xi = C_\xi a_\xi.$$

A difusão nesta transformação é restrita apenas às colunas, por ser local a estas, e portanto seu custo de implementação tem o potencial de ser bem mais baixo.

Por analogia a grupos ativos, pode-se definir a idéia de *colunas ativas*, denotadas por w_s.

O número de ramificação de θ é o número de ramificação mínimo de suas transformações componentes. Um limite superior para o número de ramificação de θ é obtido pela aplicação de (2.5) às transformações componentes de

θ, definidas pelas matrizes C_ξ, e é dado por

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

Um corolário imediato é que a menor coluna é o fator limitante do número de ramificação.

Sobre θ, temos o seguinte lema de propagação.

Lema 1. *O peso de grupo de qualquer trilha de duas rodadas cuja primeira rodada emprega a estrutura $\gamma\theta$ possui limite inferior $N\mathcal{B}(\theta)$, onde N é o número de colunas ativas na entrada da segunda rodada.*

2.5.4.2 A transformação linear Θ

Θ é uma transformação que mistura grupos através de colunas, com o objetivo de prover difusão inter-colunas. Matematicamente, Θ é descrita por

$$\Theta : b = \Theta(a) \Leftrightarrow b_i = \bigoplus_j C_{i,j} a_j.$$

O critério de projeto de Θ é ter um alto número de ramificação em relação à partição de colunas, o *número de ramificação de coluna*, denotado por $\mathcal{B}^c(\theta)$.

2.5.4.3 Propagação de trilhas em quatro rodadas

Combinando o número de ramificação de grupo de θ e o número de ramificação de coluna de Θ, é possível provar um resultado sobre o peso de grupo de uma trilha de quatro rodadas em cifras com esta estrutura.

Teorema 2. *Para uma cifra de bloco com alternância de chaves e estrutura de duas rodadas dada por (2.7) e (2.8), o limite inferior para o peso de grupo de uma trilha de 4 rodadas da forma $\rho^b \circ \rho^a \circ \rho^b \circ \rho^a$ é $\mathcal{B}(\theta)\mathcal{B}^c(\theta)$.*

2.5.4.4 Uma construção eficiente para Θ

Ao contrário de θ, Θ não opera isoladamente sobre cada coluna, e por isso pode ter alto custo de implementação. Uma construção eficiente para Θ pode ser dada em termos de θ e uma transformação de transposição de grupos denotada por π:

$$\Theta = \pi \circ \theta \circ \pi.$$

π é definida como

$$\pi : b = \pi(a) \Leftrightarrow b_i = a_{p(i)},$$

onde $p(i)$ é uma permutação do espaço de índices $\mathcal{I}$. Observa-se que π apenas rearranja a ordem dos grupos, sem alterar o estado de um grupo de ativo para inativo ou vice-versa, e portanto não afeta o número de ramificação de outra transformação quando composta com ela. A função de π é prover difusão inter-coluna, e intuitivamente espera-se que isso imponha o critério que π distribua os grupos de cada coluna para o máximo possível de colunas diferentes. Esta idéia é formalizada no conceito de *difusão ótima*, que ocorre quando cada grupo de uma coluna é distribuído para uma coluna diferente. Nota-se que o termo *difusão ótima* refere-se à difusão de grupos pelo estado da cifra, e não difusão de bits individuais.

Definição 22. *π possui difusão ótima se e somente se*

$$\forall i, j \in \mathcal{I}, i \neq j : \xi(i) = \xi(j) \Rightarrow \xi(p(i)) \neq \xi(p(j)).$$

Nota-se que o conceito de difusão ótima impõe uma restrição sobre a estrutura da cifra: devem existir pelo menos tantas colunas quanto grupos na maior coluna. Se π possui difusão ótima, pode-se mostrar que o número de ramificação de coluna de Θ é limitado inferiormente pelo número de ramificação de θ.

Lema 2. *Se π é uma transposição de grupos com difusão ótima, o número de ramificação de coluna de $\pi \circ \theta \circ \pi$ é limitado inferiormente pelo número de ramificação de grupo de θ.*

Este lema justifica a nomenclatura de difusão ótima para π, no sentido que o uso de π com difusão ótima garante o aproveitamento de todo o potencial de difusão de θ, o que não ocorreria caso π não apresentasse esta propriedade.

2.5.4.5 Uso de funções de rodada idênticas

Uma observação interessante é que π comuta com γ; de fato, π apenas permuta a posição dos grupos, enquanto que γ atua isoladamente sobre os grupos. Seja uma sequência de duas rodadas $\rho^b \circ \rho^a$; temos

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

Definindo

$$\rho^c = \theta \circ \pi \circ \gamma,$$

temos que

$$\rho^b \circ \rho^a = \rho^c \circ \rho^c.$$

Portanto, é possível usar uma única função de rodada ao invés de duas funções distintas, com a estrutura ilustrada na Figura 5. Nota-se que a escolha de π na figura é apenas uma das escolhas possíveis com difusão ótima, em particular a escolha da cifra Rijndael; outras podem ser empregadas. O uso de uma única função de rodada permite reduzir o tamanho de código em software, produzir implementações mais compactas em hardware, etc.

Essa é a estrutura usada na cifra Rijndael, nas funções de hash WHIRLPOOL e MAELSTROM-0 (Capítulo 5), na cifra FUTURE (Capítulo 6), entre outras. Finalmente, combinando o Teorema 2, o Lema 2 e os resultados dessa seção, pode-se provar o seguinte teorema.

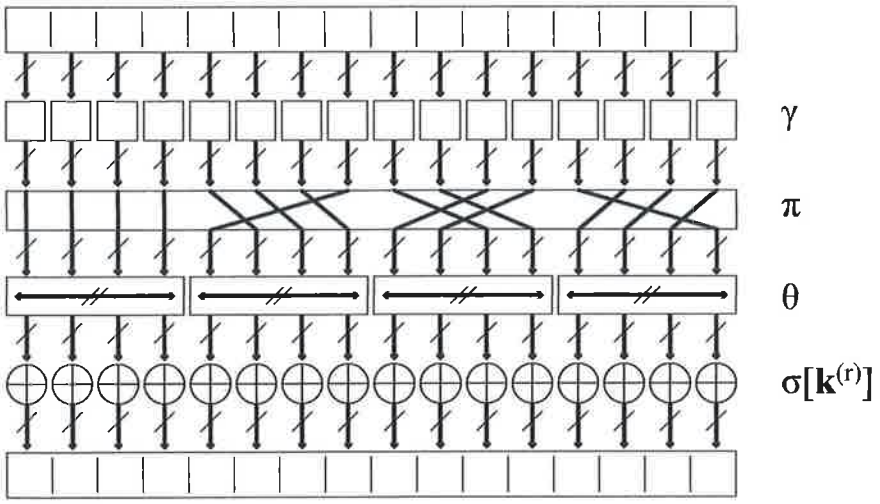

Figura 5: Estrutura de rodada $\gamma\pi\theta$.

Teorema 3. *Para uma cifra de bloco com alternância de chaves, com uma estrutura de rodada $\gamma\pi\theta$ e π com difusão ótima, o número de S-boxes ativas numa trilha de 4 rodadas é no mínimo $\mathcal{B}(\theta)^2$.*

2.6 Exemplo de aplicação da estratégia de trilha larga: a cifra Rijndael (AES)

A cifra projetada pela estratégia de trilha larga de maior importância e visibilidade é Rijndael, devido à sua escolha como Advanced Encryption Standard (AES) do governo americano. Nesta seção, serão justificadas as decisões de projeto de Rijndael, como uma cifra restrita pelos parâmetros impostos pelo concurso do AES (em particular, tamanho de bloco de 128 bits e tamanho de chaves de 128, 192 ou 256 bits). O material é baseado principalmente no capítulo 3 de (DAEMEN; RIJMEN, 2001a).

2.6.1 Parâmetros da cifra

Uma das decisões mais importantes num projeto baseado na estratégia de trilha larga é o tamanho de grupo a ser empregado. Uma vez que o tamanho das S-boxes é o tamanho de grupo, esta decisão determina a quantidade de recursos necessários para implementar as S-boxes, e também o peso de grupo mínimo das S-boxes, que determina os requerimentos de difusão da camada linear da cifra, assim como a quantidade de rodadas necessárias para garantir a segurança do algoritmo.

Como mencionado na Seção 2.5.2.3, uma S-box $m \times m$ possui peso diferencial máximo $m - 2$ e peso linear máximo $m/2$. Isto limita o tamanho mínimo das S-boxes a 3×3 para garantir uma propagação de diferenças não-determinística através da S-box.

Implementações em hardware não favorecem nenhum tamanho de grupo particular, deixando a decisão a cargo da quantidade de recursos que o projetista deseja alocar para as S-boxes; porém, em software, conjuntos de instruções usados na prática trabalham com o byte (grupo de 8 bits) como unidade básica de acesso à memória; assim, lidar com bytes, ou seus múltiplos e divisores, exige menos manipulações de operandos, como deslocamentos e aplicações de máscaras de bits, para colocar os dados nos formatos desejados. Assim, é desejável restringir os tamanhos de grupo possíveis a 4, 8, 16, 24, ... bits.

Observa-se que a quantidade de recursos despendidos com S-boxes cresce de maneira exponencial com o tamanho da S-box, enquanto os pesos diferencial e linear variam de maneira linear; esta propriedade limita a utilidade de S-boxes maiores no projeto de cifras eficientes. Ademais, os limites da tecnologia no momento do projeto restringem o tamanho das S-boxes

passíveis de implementação prática – por exemplo, a tecnologia disponível na época do projeto de Rijndael, e mesmo no presente (2007), impede o uso de S-boxes de 16 bits, uma vez que tabelas de $16 \times 2^{16}/8 = 128$ KB são inviáveis em plataformas embarcadas, e terão efeitos detrimentais mesmo em PCs e servidores, por excederem o tamanho das memórias cache de primeiro nível. Mesmo um valor intermediário de 12 bits exigiria 6-8 KB de armazenamento, que não deixa de ser inviável em muitos sistemas embarcados.

Desta maneira, a decisão sobre o tamanho de grupo de Rijndael ficou restrita aos valores de 4 e 8 bits. O tamanho de grupo de 8 bits apresenta algumas vantagens, a saber:

- tamanho natural para operações em software;
- ganhos significativos em segurança (proporcionalmente à quantidade extra de recursos utilizados) pelo uso de S-boxes 8×8 em relação a S-boxes 4×4, o que reduz o ônus sobre a camada de difusão;
- para um tamanho de bloco de 128 bits, produz um número de grupos igual a 16, o que permite a disposição do estado da cifra em uma matriz quadrada 4×4, que apresenta melhor difusão que matrizes não-quadradas.

Por julgar-se que a implementação de S-boxes 8×8 era viável nas plataformas desejadas, em termos da quantidade de recursos usados, e devido às vantagens listadas, os projetistas optaram por um tamanho de grupo de 8 bits. Portanto, a aritmética de Rijndael é realizada em GF(2^8), e a representação escolhida para este corpo, que é irrelevante para a segurança da cifra, é $\mathrm{GF}(2)/(x^8 + x^4 + x^3 + x + 1)$.

Como mencionado, para obter difusão ótima na camada linear, o estado da cifra é disposto em uma matriz 4×4, por ser o arranjo que maximiza a

difusão para o caso de 16 grupos. Nota-se que a definição original de Rijndael permitia tamanhos de bloco de 128–256 bits em incrementos de 32 bits, o que é obtido pela variação no número de colunas da matriz, mantendo o número de linhas fixo em 4; apesar de a matriz possuir maiores dimensões, o Teorema 3 não dita um aumento no número de S-boxes ativas por rodada, mas devido ao aumento no tamanho do espaço de mensagens e textos cifrados, é necessário aumentar a quantidade de S-boxes ativas ao longo da cifra inteira, o que só pode ser obtido aumentando o número de rodadas da cifra. Os valores exatos de números de rodadas, e justificativas para a escolha dos mesmos, serão dados mais à frente.

2.6.2 A S-box de Rijndael

Uma vez que a S-box é de tamanho 8×8, é sabido que o peso diferencial máximo é $8 - 2 = 6$ e o peso linear máximo é $8/2 = 4$ (para S-boxes invertíveis 8×8, como de interesse para Rijndael, o peso linear máximo é limitado a 3). Um critério de projeto sensível é obter uma S-box com pesos máximos. Também é interessante incluir algum tipo de estrutura na S-box, por exemplo para simplificar implementações em hardware. Com isto em mente, os projetistas de Rijndael decidiram por uma S-box baseada em inversão em $GF(2^8)$, uma operação com boas propriedades de não-linearidade, atingindo os pesos diferenciais e lineares máximos.

Devido à descrição algébrica relativamente simples das cifras baseadas na estratégia de trilha larga, é desejável incluir alguma forma de proteção contra ataques algébricos (por exemplo, ataques de interpolação). No caso da S-box de Rijndael, é feita uma composição da transformação de inversão em $GF(2^8)$ com uma transformação afim invertível, que não impacta as propriedades de não-linearidade da S-box, mas que torna sua expressão algébrica

bastante complexa. Em hardware, esta operação não exige muitos recursos extras, enquanto que em software, a complexidade de implementação é a mesma para qualquer escolha de S-box, devido ao uso de tabelas para implementação da mesma.

2.6.3 A camada linear de Rijndael

Devido à disposição do estado da cifra em uma matriz 4×4, o limite superior para o número de ramificação da transformação θ é $\mathcal{B}(\theta) = 5$. É possível atingir esse limite sem despender muitos recursos (basta escolher uma matriz que gere um código linear MDS), de forma que é natural adotar este requerimento para a camada linear. No entanto, este critério não é muito restritivo; existem inúmeras possibilidades para θ que atendem a este critério. Uma restrição que simplifica a implementação é o uso de matrizes circulantes, que permitem a redução do tamanho de código e de tabelas pré-computadas em implementações restritas em software, ou a quantidade de hardware sintetizado em implementações de hardware buscando a minimização de área, uma vez que apenas 4 coeficientes são suficientes para definir a matriz inteira. Por razões de eficiência, é preciso escolher coeficientes 'pequenos' para a matriz (ou seja, polinômios de grau limitado e com o mínimo possível de termos). Dados estes requerimentos, foi escolhida a matriz circulante $(\mathtt{02}, \mathtt{01}, \mathtt{01}, \mathtt{03})$, que permite a implementação da camada linear de Rijndael, caso tabelas pré-computadas não sejam empregadas, usando apenas 4 adições (XOR) e 2 multiplicações por x em GF(2^8) por coluna da matriz.

Já a transformação π pode ser escolhida sem preocupações com a eficiência, uma vez que em software é implementada apenas por aritmética de índices de arrays, e em hardware por roteamento de sinais. O único requerimento por parte da estratégia de trilha larga é que a transformação escolhida

possua difusão ótima, e uma das escolhas mais simples e óbvias é a rotação de cada linha (ou coluna) da matriz pelo seu índice; quer dizer, a i-ésima linha (ou coluna) é rotacionada por i posições. Essa foi a transformação escolhida para Rijndael, e por razões de eficiência, foram utilizadas rotações de linhas ao invés de colunas. Observa-se que a escolha de rotações por linha ou por coluna determina se a aplicação da camada linear deve ser feita por multiplicação à esquerda ou à direita pela matriz que implementa θ; uma escolha errada resultaria em difusão nula.

2.6.4 Resistência contra criptanálise

Devido à estrutura $\gamma\pi\theta$ de Rijndael (Figura 5), com $\mathcal{B}(\theta) = 5$, temos pelo Teorema 3 que o número de S-boxes ativas a cada 4 rodadas da cifra é $\mathcal{B}(\theta)^2 = 5^2 = 25$. Uma vez que a S-box possui peso diferencial 6 e peso linear 3, o peso mínimo de qualquer trilha diferencial através de 4 rodadas é de $6 \times 25 = 150$, e o peso mínimo de qualquer trilha linear através de 4 rodadas é de $3 \times 25 = 75$. Assim, apenas 4 rodadas de Rijndael seriam imunes a ataques diferenciais e lineares para chaves de 128 bits. Já 8 rodadas de Rijndael resultam no dobro dos pesos indicados, o que é suficiente para evitar ataques diferenciais e lineares contra Rijndael com chaves de 192 e 256 bits.

No entanto, a escolha do número de rodadas deve levar em conta outros ataques mais fortes sobre a cifra, como o ataque quadrado (DAEMEN; KNUDSEN; RIJMEN, 1997) e o ataque Gilbert-Minier (GILBERT; MINIER, 2000). Levando ataques alternativos em conta, e adicionando uma margem de segurança compatível com a vida útil esperada do algoritmo, os projetistas decidiram por 10 rodadas para a versão com chaves de 128 bits, 12 rodadas para a versão com chaves de 192 bits, e 14 rodadas para a versão com chaves de 256 bits.

2.7 Sumário

Este capítulo começou com uma descrição do DES (Seção 2.1), uma cifra de bastante importância histórica, e sobre a qual dois dos mais importantes ataques criptanalíticos foram originalmente montados: a criptanálise diferencial (Seção 2.2) e a criptanálise linear (Seção 2.3). Ataques diferenciais consideram pares de mensagens com pequenas diferenças entre si, e o efeito das operações da cifra sobre estas diferenças, enquanto ataques lineares tentam construir aproximações lineares para as operações não-lineares da cifra. Estes dois ataques foram tratados em um formalismo (Seção 2.4) que realça a semelhança entre ambos e sugere uma maneira unificada de projetar cifras resistentes ao mesmos, que se concretiza na estratégia de trilha larga (Seção 2.5). Brevemente, esta estratégia emprega componentes não-lineares (S-boxes) que garantem baixa probabilidade de propagação local de diferenças e correlações lineares, ao mesmo tempo que usa componentes lineares que garantem alta difusão global das diferenças e correlações lineares, por meio da maximização (dentro dos recursos disponíveis para implementação) do número de S-boxes ativas. Construções são dadas para estes componentes de maneira a obter boa segurança e eficiência. Por fim, os conceitos discutidos são ilustrados de forma concreta através de uma discussão das decisões de projeto por trás da cifra Rijndael (AES) (Seção 2.6).

Este capítulo tratou primariamente da segurança da estratégia de trilha larga, porém é preciso substanciar as afirmações de eficiência desta estratégia, o que exige cuidado na implementação, com estratégias que variam de plataforma para plataforma e que podem não ser imediatamente óbvias. Estas estratégias são discutidas no Capítulo 3.

3 IMPLEMENTAÇÃO DE PRIMITIVAS BASEADAS NA ESTRATÉGIA DE TRILHA LARGA

A implementação de primitivas projetadas de acordo com a estratégia de trilha larga (Capítulo 2) de maneira eficiente não é imediatamente óbvia, particularmente em plataformas com maior disponibilidade de recursos. A escolha de operações simples, disponíveis em qualquer plataforma, e a grande quantidade de paralelismo na função de rodada, são os trunfos da estratégia de trilha larga. É possível assim obter uma grande gama de implementações com características diversas, desde implementações compactas até de alto desempenho, desde que sejam tomados os cuidados aqui listados.

O capítulo é dividido em duas seções, com a Seção 3.1 tratando da implementação em software e a Seção 3.2 tratando da implementação em hardware. A primeira se subdivide em três subseções, tratando distintamente de plataformas embarcadas como microcontroladores (Seção 3.1.1) e de processadores de alto desempenho, como os empregados em PCs desktop e servidores (Seção 3.1.2), além da discussão de uma técnica alternativa de implementação, a técnica de *bitslicing*, que é investigada neste trabalho como um dos caminhos para obtenção de maior desempenho na implementação da estratégia de trilha larga. A Seção 3.3 faz um sumário do capítulo.

3.1 Software

A importância da implementação eficiente de primitivas criptográficas em software é indiscutível; nem todo sistema pode se dar ao luxo de incluir aceleradores criptográficos dedicados em hardware, especialmente sistemas embarcados e de menor porte em geral, e mesmo sistemas de maior porte, como servidores, tentam evitar hardware dedicado por questões de custo. Considerando a gama de plataformas de software disponíveis, com diferenças de ordens de magnitude nos recursos disponíveis, o projeto de uma cifra deve tomar muito cuidado para não se direcionar a um setor específico do espectro de desempenho, em detrimento dos demais. As primitivas projetadas pela estratégia de trilha larga tratam esta questão pelo uso de operações simples, como XORs e acessos à memória, que não favorecem nenhuma plataforma específica (8/16/32/64 bits), além da presença de bastante paralelismo na função de rodada, essencial para implementações de alto desempenho. Ademais, em plataformas com maior disponibilidade de memória, é possível construir tabelas pré-computadas que aceleram a computação, com tamanhos (e ganhos) variáveis de acordo com o espaço disponível, conforme discutido na Seção 3.1.2. Além disso, grande parte da função de rodada destas primitivas é amigável à implementação em bitslicing, como será visto na Seção 3.1.3.

3.1.1 Microcontroladores

As principais características dos processadores de pequeno porte, presentes nos núcleos de microcontroladores e smart cards, são:

1. Conjuntos de instruções de 8 ou, mais raramente, 16 bits (recentemente modelos de 32 bits estão se popularizando);

2. Baixas frequências de clock e falta de suporte a qualquer tipo de parale-

lismo;

3. Memória de código e dados limitada.

Aplicações típicas que empregam estes processadores, como smart cards, não exigem alto throughput de dados; porém, limitações sobre o tempo de resposta e consumo de energia são mais comuns. Ademais, em sistemas onde as operações criptográficas são apenas auxiliares (por exemplo, como parte de protocolos de comunicação) ao propósito do sistema, espera-se que não tomem uma porção muito grande do tempo do sistema. Assim, existe uma necessidade de implementações de alto desempenho das primitivas baseadas na estratégia de trilha larga para estes processadores.

Infelizmente, a obtenção de máximo desempenho exige uma certa quantidade de tabelas pré-computadas, e é conflitante com o item (3) da lista acima. Mesmo a implementação mais simples exigirá 256 bytes de ROM para a S-box, o que já pode ser problemático para as plataformas mais simples, mas isto não pode ser contornado.

A menos do passo SubBytes (γ), que exige a tabela da S-box, os demais passos de um algoritmo baseado na estratégia de trilha larga podem ser implementados diretamente. A implementação de ShiftRows (π) e AddRoundKey (σ) é imediata, e a única dificuldade encontra-se no passo MixColumns (θ), que requer multiplicações por constantes em GF(2^8). Uma maneira simples e relativamente eficiente de implementar esta operação é implementando uma rotina, chamada `xtimes`, para multiplicação pelo elemento $\mathtt{02} = x$ em GF(2^8), e uma cadeia de exponenciação para decompor uma multiplicação em operações de adição (que, em GF(2^8), correspondem à operação XOR) e chamadas a `xtimes`. Por exemplo, a multiplicação de um elemento b por $15 = x^4 + x^2 + 1$

seria realizada da seguinte forma (já com algumas otimizações):

$$\begin{aligned} b(x^4 + x^2 + 1) &= x^2(bx^2 + b) + b \\ &= x^2(\texttt{xtimes}(\texttt{xtimes}(b)) \oplus b) \oplus b \\ &= \texttt{xtimes}(\texttt{xtimes}(\texttt{xtimes}(\texttt{xtimes}(b)) \oplus b)) \oplus b. \end{aligned}$$

Por sua vez, a implementação da rotina `xtimes` depende da representação escolhida para GF(2^8), que é função do polinômio irredutível escolhido para definir o corpo. Como exemplo, seja o polinômio usado na definição da cifra Rijndael, $f(x) = x^8 + x^4 + x^3 + x + 1$. As operações são feitas módulos este polinômio, o que significa que no corpo escolhido, $x^8 + x^4 + x^3 + x + 1 = 0$ ou $x^8 = x^4 + x^3 + x + 1$ (lembrando que em GF(2^n), a operação de subtração coincide com a adição). Seja um elemento genérico de GF(2^8), $b = \sum_{i=0}^{7} b_i x^i$. Temos

$$\begin{aligned} bx &= (b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0)x \\ &= b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \\ &= b_6x^7 + b_5x^6 + b_4x^5 + (b_3 + b_7)x^4 + (b_2 + b_7)x^3 + b_1x^2 + (b_0 + b_7)x + b_7. \end{aligned}$$

Esta expressão pode ser implementada de duas maneiras. Independente da maneira escolhida, inicia-se por um deslocamento à esquerda de b, denotado por $b \ll 1$, produzindo o byte $b_6b_5b_4b_3b_2b_1b_00$. Em seguida, a implementação mais simples é através de um desvio condicional sobre b_7, que aplica uma máscara adequada (neste caso, `00011011`) caso $b_7 = 1$. Caso queira-se evitar o desvio condicional, como defesa contra ataques *side channel* ou por razões de desempenho em processadores com pipelines longas, usa-se o seguinte procedimento:

1. Gera-se um byte com todos os bits iguais a b_7 (isto pode ser feito usando uma operação de deslocamento aritmético à direita por 7);

2. Gera-se uma máscara a partir da aplicação da máscara do polinômio ao byte gerado, usando a operação AND;

3. Aplica-se a máscara gerada a $b \ll 1$ através da operação XOR.

Muitos processadores mais simples não implementam desvios por uma quantia arbitrária, o que exigiria no passo (1) a aplicação de 7 instruções de deslocamento, tornando esta opção menos atrativa nestes processadores sob o ponto de vista de desempenho; sua implementação ainda pode ser necessária caso haja um requerimento de proteção contra ataques *side channel*.

3.1.2 PCs e Servidores

As principais características dos processadores empregados em PCs e servidores são:

- Conjunto de instruções de 32 e, mais recentemente, 64 bits;
- Arquiteturas superescalares, capazes de executar múltiplas instruções em paralelo (desde que as instruções não dependam uma da outra, ou dependam de certos recursos compartilhados);
- Implementação do caminho de dados em pipelines cada vez mais profundas, que penalizam severamente as dependências de dados e instruções de desvio condicional (hardware para previsão de desvio aliado a especulação permite eliminar o custo de muitos tipos de desvios, porém outros são inerentemente imprevisíveis);
- Suporte a instruções vetoriais (SIMD), que executam uma dada instrução sobre um vetor de valores simultaneamente;

- Memória de código e dados essencialmente ilimitada, porém disposta numa hierarquia de memória (através do uso de caches), de maneira que conjuntos menores de dados podem ser acessados mais rapidamente;
- Principalmente em servidores, e mais recentemente em PCs, presença de múltiplos processadores ou processadores multi-núcleo.

A característica mais importante para a implementação de primitivas baseadas na estratégia de trilha larga é a possibilidade de empregar tabelas pré-computadas relativamente grandes (da ordem de kilobytes para acesso rápido através do cache L1, megabytes para acesso um pouco mais lento através do cache L2, e possivelmente até gigabytes para acesso lento através da memória principal). No melhor caso, estas tabelas combinam os passos SubBytes, ShiftRows e MixColumns numa única operação, bastando implementar à parte AddRoundKey para completar uma rodada; o número de operações por rodada é um acesso à memória e um XOR por byte da matriz de estado (por exemplo, 16 bytes no caso da cifra Rijndael). A memória ocupada por este esquema, admitindo S-boxes 8×8 e uma matriz de estado de dimensão $m \times n$, é $256mn$ bytes (por exemplo, 4 KB no caso da cifra Rijndael). Para valores práticos de m e n, estas tabelas cabem completamente no cache L1 de qualquer processador moderno. Este esquema é descrito a seguir, usando como exemplo concreto a cifra Rijndael; a modificação do exemplo para outras primitivas com parâmetros diferentes é trivial.

Sejam as matrizes de estado A, B, C, D, correspondendo respectivamente à entrada de uma rodada, a saída do passo SubBytes, a saída do passo ShiftRows e a saída do passo MixColumns. Os elementos destas matrizes são denotados por $a_{i,j}, b_{i,j}, c_{i,j}, d_{i,j}$. Seja $S[x]$ a operação de aplicar a S-box de SubBytes ao elemento x, e k_i o valor do deslocamento para a i-ésima coluna

de ShiftRows. Temos então

$$b_{i,j} = S[a_{i,j}],$$

$$c_{i,j} = b_{i,j+k_i},$$

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} \mathtt{02} & \mathtt{03} & \mathtt{01} & \mathtt{01} \\ \mathtt{01} & \mathtt{02} & \mathtt{03} & \mathtt{01} \\ \mathtt{01} & \mathtt{01} & \mathtt{02} & \mathtt{03} \\ \mathtt{03} & \mathtt{01} & \mathtt{01} & \mathtt{02} \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}.$$

Combinando os diferentes passos, obtém-se

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} \mathtt{02} & \mathtt{03} & \mathtt{01} & \mathtt{01} \\ \mathtt{01} & \mathtt{02} & \mathtt{03} & \mathtt{01} \\ \mathtt{01} & \mathtt{01} & \mathtt{02} & \mathtt{03} \\ \mathtt{03} & \mathtt{01} & \mathtt{01} & \mathtt{02} \end{bmatrix} \begin{bmatrix} S[a_{0,j+k_0}] \\ S[a_{1,j+k_1}] \\ S[a_{2,j+k_2}] \\ S[a_{3,j+k_3}] \end{bmatrix}.$$

Observa-se que a adição de índices deve ser feita módulo 4. A multiplicação de matriz na equação pode ser decomposta da seguinte maneira:

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} \mathtt{02} \\ \mathtt{01} \\ \mathtt{01} \\ \mathtt{03} \end{bmatrix} S[a_{0,j+k_0}] \oplus \begin{bmatrix} \mathtt{03} \\ \mathtt{02} \\ \mathtt{01} \\ \mathtt{01} \end{bmatrix} S[a_{1,j+k_1}] \oplus$$

$$\begin{bmatrix} \mathtt{01} \\ \mathtt{03} \\ \mathtt{02} \\ \mathtt{01} \end{bmatrix} S[a_{2,j+k_2}] \oplus \begin{bmatrix} \mathtt{01} \\ \mathtt{01} \\ \mathtt{03} \\ \mathtt{02} \end{bmatrix} S[a_{3,j+k_3}].$$

Definem-se agora as tabelas T_0, T_1, T_2, T_3 como

$$T_0[a] = \begin{bmatrix} \mathtt{02} \\ \mathtt{01} \\ \mathtt{01} \\ \mathtt{03} \end{bmatrix} S[a], \quad T_1[a] = \begin{bmatrix} \mathtt{03} \\ \mathtt{02} \\ \mathtt{01} \\ \mathtt{01} \end{bmatrix} S[a],$$

$$T_2[a] = \begin{bmatrix} \mathtt{01} \\ \mathtt{03} \\ \mathtt{02} \\ \mathtt{01} \end{bmatrix} S[a], \quad T_3[a] = \begin{bmatrix} \mathtt{01} \\ \mathtt{01} \\ \mathtt{03} \\ \mathtt{02} \end{bmatrix} S[a],$$

Por fim, pode-se expressar as operações combinadas através das tabelas como

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = T_0[a_{0,j+k_0}] \oplus T_1[a_{1,j+k_1}] \oplus T_2[a_{2,j+k_2}] \oplus T_3[a_{3,j+k_3}].$$

A implementação desta formulação exige 4 acessos (de 32 bits) à memória e 3 XORs de 32 bits, mais um XOR de 32 bits para implementação de AddRoundKey, totalizando uma rodada completa da cifra. Não são contadas operações para adição de índices, porque essas operações podem ser substituídas por uma expressão constante pelo compilador caso a técnica de *loop unrolling* seja utilizada. Uma observação é que T_0, T_1, T_2, T_3 são versões rotacionadas da mesma tabela; é possível empregar uma única tabela, reduzindo o uso de memória para $1/4$ do original, em troca de uma rotação de bits para cada acesso à memória.

Uma desvantagem dessa técnica é que o paralelismo em nível de instrução dos processadores não é aproveitado; de fato, o desempenho do algoritmo é limitado pela latência e número de acessos simultâneos possíveis ao cache do processador. Outra desvantagem é a possibilidade de ataques *side channel* por conta de pequenas variações na latência do cache em função do padrão de acessos às tabelas (BERNSTEIN, 2005). Embora os passos ShiftRows, MixColumns e AddRoundKey possam ser implementados de maneira relativamente eficiente sem o uso de tabelas, a implementação de SubBytes para uma S-box arbitrária exige o uso de tabelas para obter um desempenho aceitável. Parte da proposta deste trabalho é a construção de S-boxes com algum tipo de estrutura, que permita a implementação sem o uso de tabelas; uma das possibilidades é a técnica de S-boxes invariantes a rotação, descrita no Capítulo 4. Mesmo assim, a implementação de S-boxes desta maneira só terá desempenho competitivo se implementado em hardware, ou em software em conjunção com a técnica de *bitslicing*, descrita na Seção 3.1.3. Ademais, a técnica de bitslicing se beneficia de eventuais conjuntos de instruções vetoriais presentes no processador.

Em geral, a exploração de paralelismo em sistemas multi-processados não pode ser feita dentro de uma única execução da cifra, num modo de operação serializado como CBC (Cipher Block Chaining); embora haja uma certa quantidade de paralelismo entre as operações da cifra, o tempo de sincronização de variáveis entre diferentes processadores tomaria uma porção muito significativa do tempo de execução. Se o uso de sistemas multi-processados for um requerimento de aplicação, recomenda-se a adoção de modos de operação inerentemente paralelizáveis, como o modo counter.

3.1.3 Bitslicing

Em 1997, Eli Biham publicou um artigo descrevendo um novo método de implementação do DES em software (BIHAM, 1997), hoje conhecido pelo termo *bitslicing*. Biham observou, em um processador de 64 bits, uma melhora de desempenho de 3 vezes em relação a uma implementação convencional otimizada para aquela plataforma. Devido à descoberta de representações mais eficientes para as S-boxes do DES, reduzindo quase pela metade o número de operações para implementar as S-boxes em relação ao artigo original, esta diferença de desempenho torna-se ainda mais pronunciada. Deve-se observar, no entanto, que a estrutura do DES é particularmente amigável a implementação em hardware, em detrimento de seu desempenho em software, particularmente em processadores de 32 bits ou mais; projetos mais eficientes em software não obterão ganhos tão dramáticos com o uso de bitslicing.

A técnica mostrou-se útil para outras aplicações além do DES, e inclusive certas cifras (como Serpent (ANDERSON; BIHAM; KNUDSEN, 1998), um dos finalistas do AES) foram projetadas para serem implementadas com esta técnica. Devido à similaridade com técnicas de implementação em hardware, avanços na implementação em hardware podem ser aproveitados na implementação em bitslicing e vice-versa. Um dos objetivos desta dissertação é a compatibilização da estratégia de trilha larga com a técnica de implementação em bitslicing.

Um detalhe sobre esta técnica é que diversas instâncias da primitiva sendo implementada são calculadas em paralelo, de maneira que é necessário empregar um modo de operação paralelizável, como o modo counter (MENEZES; OORSCHOT; VANSTONE, 1997), para usufruir dos ganhos oferecidos pela técnica. Isso não necessariamente é verdade para primitivas projetadas direta-

mente para uso com bitslicing (empregando a técnica internamente à cifra), a exemplo da cifra Serpent, que pode ser utilizada com modos de operação serializados sem perda de desempenho.

A idéia básica da técnica de bitslicing é tratar um processador de n bits como n processadores independentes de 1 bit cada, algo como o paradigma SIMD levado ao extremo. Cada processador de 1 bit implementa uma cópia completa da primitiva desejada, de maneira completamente independente dos demais[1]. As operações admissíveis são principalmente operações lógicas como AND, OR, XOR, NOT, etc.; outras operações, como operações aritméticas e de deslocamento, não podem ser aproveitadas por violarem a compartimentalização entre os processadores de 1 bit (por exemplo, na operação de adição, podem ser gerados vai-uns durante o cálculo de um dado bit do resultado, e este vai-um afeta os bits seguintes do cálculo). Essencialmente, uma implementação em bitslicing equivale a tomar o circuito lógico que implementa a operação desejada em hardware, e instanciar n cópias do mesmo circuito, todas executando simultaneamente em paralelo (como se estivessem sujeitas aos mesmos pulsos de clock em hardware).

Vale observar que a representação de dados nesta técnica é distinta da representação tradicional; por exemplo, no caso do DES, que possui um estado interno de 64 bits, seria razoável armazenar este estado numa variável de 64 bits, ou duas variáveis de 32 bits. Admitindo que deseje-se aplicar a primitiva n vezes em paralelo, tipicamente será declarado um array de n elementos da(s) variável(is) do estado interno. Em contraste, uma representação com bitslicing é uma espécie de transposição da representação convencional; seriam empregadas 64 variáveis, cada uma de n bits, onde n é o tamanho de palavra do processador, e o k-ésimo bit da i-ésima variável representa o

[1] Obviamente, as operações ocorrerão nos mesmos registradores da CPU, mas cada bit de um registrador não interage com os demais.

i-ésimo bit do estado interno da k-ésima aplicação paralela do DES. Uma vantagem dessa representação transposta é que operações como permutações de bits (como as permutações inicial e final do DES, e a permutação de bits após a aplicação de S-boxes, na função F do DES) equivalem apenas a copiar uma variável para outra, ou mesmo apenas 'renomear' a variável, o que não gera operação nenhuma para a CPU, e inclusive pode ser feito, ao menos em teoria, de maneira automática pelo compilador. Da mesma forma, a operação de expansão de bits da função F do DES, se implementada com cuidado, não exige a geração de nenhuma operação de CPU. O escalonamento de chaves do DES emprega apenas permutações e rotações de bits (e por sua vez, rotações são um caso especial de permutações), de maneira que é possível realizar todo o escalonamento de chaves sem custo algum.

Das demais operações do DES, a aplicação de subchaves e o XOR das duas metades do estado interno (na estrutura de Feistel), por serem ambos baseados na operação XOR, podem ser implementados trivialmente em bitslicing usando essa operação. A única operação restante no DES é a aplicação de S-boxes; infelizmente, esta operação não tem uma representação óbvia. Em (BIHAM, 1997), Biham exibe um método simples para gerar um circuito lógico equivalente às S-boxes do DES, com um custo médio de 100 portas lógicas por S-box. Técnicas mais elaboradas foram aplicadas por diversos pesquisadores para reduzir esse custo; o recorde atual pertence a Matthew Kwan (KWAN, 2000), com um custo médio de 56 portas lógicas empregando operações lógicas 'típicas' (apenas AND, OR, NOT e XOR), ou 51 portas lógicas se as operações NXOR e AND-NOT estiverem disponíveis também. Empregando estas expressões mais eficientes para as S-boxes, é possível reduzir em 38% o custo total do algoritmo em relação à implementação original de (BIHAM, 1997), ou 33% se for incluído o custo de conversão da representação

convencional das mensagens/textos cifrados para a representação compatível com bitslicing.

Em geral, S-boxes são as operações mais complexas de serem representadas e implementadas em bitslicing. O tamanho das S-boxes está diretamente relacionado ao tamanho do circuito necessário para implementá-las; por exemplo, as S-boxes 4×4 do Serpent requerem na média 17 portas lógicas, enquanto as melhores expressões conhecidas para as S-boxes 6×4 do DES requerem na média 51 portas lógicas, como já mencionado. A estratégia de trilha larga tipicamente emprega S-boxes 8×8 como um bom compromisso entre desempenho e segurança, e S-boxes deste tamanho, a menos que sejam projetadas especificamente para otimizar sua expressão em hardware em bitslicing, possuirão uma expressão em bitslicing tão ineficiente que a vantagem da implementação completa da cifra em bitslicing, em relação a uma implementação convencional, será eliminada. Expressões eficientes são certamente possíveis, a exemplo da função de hash WHIRLPOOL (ver Seção 5.1), que expressa sua S-box 8×8 em termos de 5 S-boxes 4×4 e algumas operações lógicas extras – admitindo a mesma média de 17 operações do Serpent para S-boxes 4×4, é possível implementar a S-box 8×8 de WHIRLPOOL com menos de 100 portas lógicas. Uma abordagem promissora para o projeto das S-boxes 4×4 é o uso de S-boxes invariantes por rotação (Capítulo 4); embora o custo (em termos do número de portas lógicas) não seja reduzido, o paralelismo entre os diferentes bits da saída da S-box permite ou reduzir a profundidade do circuito em implementações em hardware, ou explorar o paralelismo a nível de instrução do processador em implementações em bitslicing. Em particular, a S-box 4×4 quase invariante por rotação empregada na cifra FUTURE (Capítulo 6) pode ser implementada usando um circuito com profundidade 3, composto respectivamente por 4, 8 e 6 portas lógicas em paralelo

para cada nível do circuito, num total de 18 portas lógicas.

3.2 Hardware

Primitivas baseadas na estratégia de trilha larga possuem implementação simples em hardware, em parte devido ao uso de operações em GF(2^n), que possuem um mapeamento direto para circuitos lógicos em hardware, ocupando menor área e gerando menores atrasos de propagação que operações utilizadas em outras cifras, como aritmética de inteiros e rotações dependentes de dados.

As principais operações empregadas na estratégia de trilha larga são:

- **Adição em GF(2^n)** (p.ex. AddRoundKey e MixColumns): corresponde à operação XOR sobre n bits, de implementação bastante simples;
- **Rotação por um número constante de bits** (p.ex. ShiftRows): por ser uma permutação de bits, exige apenas roteamento dos sinais, sem gerar nenhuma operação lógica;
- **Multiplicação por constantes em GF(2^n)** (p.ex. MixColumns): pode ser implementado por uma combinação pré-determinada de adições e deslocamentos à esquerda por um número constante de bits (deslocamentos são similares a rotações e podem ser implementados da mesma maneira – a única diferença é que os bits mais significativos do resultado são descartados).
- **Aplicação de S-boxes** (p.ex. SubBytes): geralmente, a operação mais complexa, que deve ser instanciada uma vez por byte do estado interno da cifra; pode ser implementada diretamente por meio de tabelas, o que

tipicamente consumiria a maior parte da área do chip, ou aproveitando a existência de algum tipo de estrutura na definição da S-box.

Além da dificuldade na implementação das S-boxes, outra preocupação do projetista é o compartilhamento de circuitos entre a operação de cifração e decifração. Observa-se que as operações do algoritmo de decifração são inversas das operações do algoritmo de cifração, e devem ser executadas em ordem inversa. Com relação ao segundo ponto, deve-se mencionar que devido à comutatividade de algumas operações da função de rodada da estratégia de trilha larga, é possível escrever um algoritmo equivalente de decifração em que dispensa-se a inversão de ordem das operações (embora ainda deva-se aplicar as operações inversas). Esta flexibilidade possui um pequeno custo extra, porém: é necessário aplicar a operação InvMixColumns sobre as subchaves. O leitor interessado pode consultar (DAEMEN; RIJMEN, 2001a) para mais detalhes sobre o algoritmo equivalente de decifração no caso da cifra Rijndael, que pode ser facilmente modificado para uso com outras primitivas baseadas na estratégia de trilha larga. Se o projetista julgar que o ganho de empregar a mesma ordem para os algoritmos de cifração e decifração é menor que o custo de uma aplicação extra de InvMixColumns sobre a subchave, ele pode proceder com esta abordagem.

O requerimento de instanciar versões invertidas das operações para decifração (InvSubBytes, InvShiftRows e InvMixColumns) pode parecer oneroso, porém o custo destas instanciações duplicadas é dominado por InvSubBytes, e em muitos casos a estrutura presente na S-box permite reduzir o custo desta operação. Um exemplo é a S-box da cifra Rijndael, que é uma composição de duas operações, inversão em $GF(2^8)$ e uma transformação afim, denotadas

por g e f a seguir. Matematicamente,

$$S[a] = f(g(a)).$$

Temos então que a inversa desta operação é

$$S^{-1}[a] = g^{-1}(f^{-1}(a)).$$

No entanto, g é uma função auto-inversa; assim, $g^{-1} = g$ e

$$S^{-1}[a] = g(f^{-1}(a)).$$

Desta forma, a implementação de SubBytes e InvSubBytes pode compartilhar a implementação da função g, exigindo apenas instanciações separadas de f e f^{-1}, que possuem um custo de implementação muito menor (algumas operações XOR).

A implementação das S-boxes de maneira otimizada é um dos maiores desafios, pois como já citado, uma implementação genérica, utilizando tabelas, dominaria a área do chip. É vantajoso que o projetista da primitiva escolha uma S-box com algum tipo de estrutura, que permita uma implementação mais compacta que a implementação genérica. Serão dados dois exemplos para ilustrar este princípio: a S-box da cifra Rijndael e da função de hash WHIRLPOOL.

Lembrando que a S-box da cifra Rijndael é a composição de inversão em GF(2^8) e uma transformação afim, e que o custo de implementação da transformação afim é baixo, busca-se uma maneira de otimizar a implementação da inversão em GF(2^8). Esta operação já foi bastante estudada na literatura, e diversas técnicas de implementação são conhecidas, representando diferentes compromissos entre área e frequência de clock. Uma das abordagens possíveis é listada a seguir.

Seja $GF(2^4)^2$ o corpo de polinômios de grau 1 com coeficientes em $GF(2^4)$, cuja aritmética é realizada módulo um polinômio da forma $x^2 + x + A$, com A um elemento de $GF(2^4)$ que torne este polinômio irredutível. Todo elemento de $GF(2^8)$ pode ser mapeado para um elemento de $GF(2^4)^2$ através de uma transformação linear. Um elemento $bx + c$ de $GF(2^4)^2$ e seu elemento inverso $px + q$ satisfazem

$$\begin{aligned} 1 &= (bx + c)(px + q) \pmod{x^2 + x + A} \\ &= (cp \oplus bq \oplus bp)x + (cq \oplus bpA). \end{aligned}$$

Igualando os coeficientes de x e 1, obtém-se o sistema linear

$$\begin{cases} 0 &= cp \oplus bq \oplus bp \\ 1 &= cq \oplus bpA \end{cases}$$

Resolvendo o sistema para p, q, obtém-se

$$\begin{cases} p &= b(Ab^2 \oplus bc \oplus c^2)^{-1} \\ q &= (c \oplus b)(Ab^2 \oplus bc \oplus c^2)^{-1} \end{cases}$$

Utilizando esta abordagem, o problema de calcular inversos em $GF(2^8)$ foi reduzido ao problema de calcular inversos em $GF(2^4)$ e um pouco de aritmética em $GF(2^4)$; a aritmética, como visto no começo da seção, é de fácil implementação, e o cálculo de inversos em $GF(2^4)$ pode ser feito através de uma tabela de 16 entradas, em contraste à tabela de 256 entradas necessária à implementação direta do cálculo de inversos em $GF(2^8)$.

O segundo exemplo de estrutura em S-boxes pode ser encontrado na função de hash WHIRLPOOL (Seção 5.1). Esta S-box emprega uma estrutura recursiva que compõe uma S-box 8×8 a partir de cinco S-boxes 4×4 (uma S-box R e duas instanciações das S-boxes E, E^{-1}), tomando como entrada um valor de 8 bits decomposto em duas metades (i_L, i_R) de 4 bits cada, da

seguinte maneira:

$$a = E(i_L),$$

$$b = E^{-1}(i_R),$$

$$c = R(a \oplus b),$$

$$d = E(a \oplus c),$$

$$e = E(b \oplus c).$$

A saída da S-box é o par (d, e). O processo é ilustrado graficamente na figura 7. As S-boxes 4×4 podem ser implementadas através de 3 tabelas de 16 entradas cada, e representam uma economia de 80% de área em relação a uma implementação direta com uma tabela de 256 entradas.

3.3 Sumário

Primitivas baseadas na estratégia de trilha larga empregam operações que não favorecem uma dada plataforma em relação a outra, obtendo alto desempenho em plataformas desde as mais restritas até as mais poderosas. No entanto, certas técnicas de implementação devem ser observadas para usufruir deste potencial. Inicialmente, são discutidas técnicas para implementação em plataformas embarcadas de software como microcontroladores e smart cards (Seção 3.1.1), caracterizadas por instruções de 8 bits e memória escassa, onde apenas uma implementação direta do algoritmo é viável. Em PCs e servidores (Seção 3.1.2), que dispõem de processadores de 32 ou 64 bits com alto grau de paralelismo e ampla disponibilidade de memória, são discutidas técnicas baseadas em pré-computação de tabelas, que substituem muitas das operações aritméticas e lógicas por acessos à memória, que são particularmente rápidos devido à presença de quantidades suficientes de memória

cache nos processadores modernos. Uma técnica alternativa de implementação nestas plataformas é discutida, a técnica de bitslicing (Seção 3.1.3), que guarda semelhanças com a implementação em hardware; com algumas modificações na estratégia de trilha larga, particularmente no projeto de S-boxes, é possível utilizar a técnica de bitslicing para atingir novos patamares de desempenho. Finalmente, é discutida a implementação em hardware, que se beneficia das operações simples utilizadas na estratégia de trilha larga, principalmente a aritmética em $GF(2^8)$, que possui um mapeamento direto com as operações lógicas típicas em hardware. Discute-se o fato que as S-boxes apresentam a maior dificuldade de implementação em hardware, e como o projeto de S-boxes estruturadas pode sanar este problema. Implementações em hardware podem se beneficiar mais ainda da técnica de S-boxes invariantes por rotação, assunto do Capítulo 4.

4 S-BOXES INVARIANTES POR ROTAÇÃO

Como mencionado na Seção 3.1.3, a maior dificuldade na implementação de uma primitiva criptográfica usando a técnica de bitslicing é a obtenção de expressões eficientes para as S-boxes. Da mesma forma, implementações em hardware podem se beneficiar de expressões compactas para as S-boxes. Como alternativa, o projetista de uma primitiva pode empregar uma estrutura na S-box que torne-a mais amigável para implementação em bitslicing ou em hardware, a exemplo da S-box da função de hash WHIRLPOOL (Seção 5.1). Porém, mesmo a S-box do Whirlpool é uma composição de S-boxes menores (4×4), cuja implementação é mais compacta, mas não direta, especialmente em bitslicing. O método deste capítulo permite a geração de S-boxes de fácil implementação, com circuitos lógicos de grande paralelismo e pequena profundidade.

Intuitivamente, numa S-box invariante por rotação, valores de entrada idênticos módulo rotação de bits produzem valores de saída também idênticos módulo as mesmas rotações de bits. Em particular, isto implica que a expressão

da S-box (de dimensões $n \times n$) é da forma

$$o_1 = f(i_1, i_2, \ldots, i_n),$$
$$o_2 = f(i_2, \ldots, i_n, i_1),$$
$$\ldots$$
$$o_n = f(i_n, i_1, i_2, \ldots, i_{n-1}),$$

onde f é uma função booleana. Enfatiza-se o fato que a mesma função é usada para todos os bits de saída, variando apenas a ordem dos argumentos de entrada. O problema de obter expressões eficientes para funções booleanas já foi amplamente estudado na literatura; por exemplo, pode-se empregar a técnica de mapas de Karnaugh para obter uma expressão eficiente para f.

Para exemplificar o conceito de invariância por rotação, seja $f(i_1, i_2) = \overline{i_2}$. Tomando $o_1 = f(i_1, i_2)$ e $o_2 = f(i_2, i_1)$, esta função produz a S-box 2×2 dada por $\mathrm{S}[00] = 11, \mathrm{S}[01] = 01, \mathrm{S}[10] = 10, \mathrm{S}[11] = 00$. Pode-se verificar por inspeção que esta função atende à propriedade de invariância por rotação.

Procedemos a uma definição formal de invariância por rotação. Primeiramente, um esclarecimento sobre a notação empregada no restante do capítulo: os símbolos v, x, z serão usados respectivamente para indicar vetores em $\mathrm{GF}(2)^n$, elementos do corpo $\mathrm{GF}(2^n)$ e inteiros módulos 2^n.

Definição 23. *Seja $r(v)$ a rotação à esquerda de v por um bit. Uma S-box $S : GF(2)^n \to GF(2)^n$ é invariante por rotação se e somente se $r(S[v]) = S[r(v)]$.*

O número de S-boxes invariantes por rotação é muito pequeno frente ao conjunto de todas as S-boxes: por serem definidas completamente pela função f acima, existem portanto no máximo 2^{2^n} S-boxes invariantes por rotação. No entanto, nem todas as escolhas de f produzem S-boxes invariantes por rotação; de fato, um argumento combinatórico permite mostrar que o número

Tabela 1: Número de S-boxes invariantes por rotação para $n = 2, \ldots, 6$.

n	$\#S$
2	4
3	36
4	1536
5	22500000
6	263303591362560

de S-boxes $n \times n$ invariantes por rotação é

$$\prod_{d|n} (dg(d))!^d,$$

onde $x!^y = x(x-y)(x-2y)(x-3y)\cdots$ é a função multifatorial, e a função aritmética $g(x)$ é dada por

$$g(x) = \frac{1}{x} \sum_{d|x} \mu(x/d) 2^d,$$

onde μ é a função de Möbius (HARDY; WRIGHT, 1979). Uma demonstração desta afirmação será dada num artigo a ser publicado futuramente. A Tabela 1 lista o número de S-boxes $n \times n$ invariantes por rotação para alguns valores de n.

Teorema 4. *A composição de S-boxes invariantes por rotação é invariante por rotação.*

Demonstração. Sejam $S[v]$ e $T[v]$ S-boxes invariantes por rotação. Então $\mathsf{r}(S[T[v]]) = S[\mathsf{r}(T[v])] = S[T[\mathsf{r}(v)]]$. □

O restante do capítulo consistirá de descrições algébricas de S-boxes invariantes por rotação, e sua relação com outros tipos de S-boxes. Para tanto, é necessário definir a idéia de descrição algébrica.

Definição 24. *Um mapa $s(x)$ sobre o corpo $GF(2^n)$ é uma descrição algébrica*

da S-box $S[v]$ se e somente se existe um mapa invertível

$$\ell : GF(2^n) \rightarrow GF(2)^n : x \mapsto v = \ell(x)$$

tal que para todo $x \in GF(2^n)$, $S[\ell(x)] = \ell(s(x))$.

O mapa ℓ é dito um *mapa rotulador.* Concentraremos os resultados deste capítulo num tipo específico de mapa, aquele formado pela interpretação dos bits do vetor $\ell(x)$ como coordenadas de x em alguma base de GF(2^n). Para esta classe de mapas, a operação de adição em GF(2^n) corresponde à operação XOR.

Através da escolha de diferentes mapas rotuladores, é possível construir múltiplas descrições algébricas para uma mesma S-box, ou alternativamente, uma mesma descrição algébrica para múltiplas S-boxes. Isto é feito em (BARKAN; BIHAM, 2002; YOUSSEF; TAVARES, 2002) para construir descrições algébricas equivalentes para a cifra Rijndael.

4.1 Invariância por rotação em bases normais

Definição 25. *Uma base normal de GF(2^n) é da forma*

$$\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{n-1}}\}$$

para um elemento β do corpo.

Observa-se que, ao menos em GF(2^n), sempre existe uma base normal. A notação ℓ_n indicará um mapa rotulador que mapeia elementos de GF(2^n) para suas coordenadas em uma base normal.

Lema 3. *A operação de elevar ao quadrado é linear em GF(2^n).*

Demonstração. Sejam x, y elementos de GF(2^n). Então

$$(x+y)^2 = x^2 + 2xy + y^2 = x^2 + y^2,$$

uma vez que $2 \equiv 0 \pmod 2$. (Observação: esta propriedade se estende trivialmente para a soma de mais de dois elementos, pela reescrita da expressão como uma soma de duas subexpressões e aplicação recursiva da propriedade.) □

Lema 4. *Numa base normal de GF(2^n), a operação de elevar ao quadrado corresponde a uma rotação do vetor de coordenadas: $\ell_n(x^2) = r(\ell_n(x))$.*

Demonstração. Seja $x = x_0\beta + x_1\beta^2 + x_2\beta^{2^2} + \ldots + x_{n-1}\beta^{2^{n-1}}$, com $x_0, x_1, x_2, \ldots, x_{n-1} \in \{0, 1\}$. Então

$$\begin{aligned} x^2 &= (x_0\beta + x_1\beta^2 + x_2\beta^{2^2} + \ldots + x_{n-1}\beta^{2^{n-1}})^2 \\ &= x_0\beta^2 + x_1\beta^{2^2} + x_2\beta^{2^3} + \ldots + x_{n-1}\beta^{2^n} \\ &= x_{n-1}\beta + x_0\beta^2 + x_1\beta^{2^2} + x_2\beta^{2^3} + \ldots + x_{n-2}\beta^{2^{n-1}}, \end{aligned}$$

onde foi utilizado o Lema 3 para linearidade da operação de elevar ao quadrado e o fato que $\beta^{2^n} \equiv \beta$ em GF(2^n). □

O teorema a seguir fornece um teste simples para a invariância por rotação de uma S-box descrita em uma base normal.

Teorema 5. *Uma S-box $S[v]$ é invariante por rotação se e somente se sua descrição $s(x)$ em uma base normal (a partir do mapa rotulador ℓ_n) possui a propriedade $s(x)^2 = s(x^2)$.*

Demonstração. Pela Definição 23, $r(S[\ell_n(x)]) = S[r(\ell_n(x))]$. Aplicando o Lema

4, temos

$$\mathrm{r}(S[\ell_n(x)]) = \mathrm{r}(\ell_n(s(x))) = \ell_n(s(x)^2)$$

$$S[\mathrm{r}(\ell_n(x))] = S[\ell_n(x^2)] = \ell_n(s(x^2)).$$

Como ℓ_n é invertível, $\ell_n(s(x)^2) = \ell_n(s(x^2))$ implica $s(x)^2 = s(x^2)$. □

Corolário 1. *Somas e produtos de S-boxes invariantes por rotação (definidos pela soma e produto em GF(2^n) de entradas e saídas correspondentes) são invariantes por rotação.*

Demonstração. Sejam $s(x), s'(x)$ S-boxes invariantes por rotação. Temos

$$(s(x) + s'(x))^2 = s(x)^2 + s'(x)^2 = s(x^2) + s'(x^2)$$

pela invariância por rotação das S-boxes e pelo Lema 3, e

$$(s(x)s'(x))^2 = s(x)^2 s'(x)^2 = s(x^2)s'(x^2).$$

□

Estes resultados são combinados para demonstrar a propriedade a seguir.

Teorema 6. *Uma S-box $S[v]$ é invariante por rotação se e somente se puder ser descrita algebricamente por um polinômio com coeficientes binários.*

Demonstração. Primeiramente será provada a asserção direta. Seja

$$s(x) = \sum_{i=0}^{2^n-1} c_i x^i$$

para $c_i \in \{0, 1\}$. Pelo Lema 3 e o fato que $c_i = c_i^2, \forall c_i$, temos

$$s(x)^2 = \left(\sum_{i=0}^{2^n-1} c_i x^i\right)^2 = \sum_{i=0}^{2^n-1} c_i^2 x^{2i} = s(x^2),$$

o que verifica a condição do Teorema 5, de modo que $S[v]$ é invariante por rotação.

Para provar a asserção conversa, observa-se que qualquer S-box $S[v]$ pode ser descrita por um polinômio sobre GF(2^n) (em geral, com coeficientes em GF(2^n) e não apenas restritos a coeficientes binários). Matematicamente,

$$s(x) = \sum_{i=0}^{2^n-1} d_i x^i$$

para $d_i \in$ GF(2^n). Concretamente, esta descrição pode ser obtida pela aplicação da fórmula de interpolação de Lagrange, por exemplo. Pelo Teorema 5, $S[v]$ é invariante por rotação caso $s(x)^2 = s(x^2)$. Temos

$$s(x)^2 = \left(\sum_{i=0}^{2^n-1} d_i x^i\right)^2 = \sum_{i=0}^{2^n-1} d_i^2 x^{2i}$$
$$s(x^2) = \sum_{i=0}^{2^n-1} d_i (x^2)^i = \sum_{i=0}^{2n-1} d_i x^{2i}.$$

A condição do Teorema 5 é satisfeita caso $d_i = d_i^2$, o que só vale para os elementos $\{0, 1\}$ de GF(2^n), ou seja, se $s(x)$ é um polinômio com coeficientes binários. □

Em seguida, consideraremos outra classe de S-boxes, baseados num *mapa de exponenciação*, ou seja, uma descrição algébrica da forma $s(z) = b^z$, para alguma raiz primitiva b de GF(2^n). Uma vez que esta descrição depende de um inteiro e não mais de um elemento de GF(2^n), é necessário introduzir um segundo mapa rotulador ℓ^*, que mapeie inteiros para vetores em GF$(2)^n$.

Definição 26. *Um mapa $s(z) : \mathbb{Z}/2^n\mathbb{Z} = \{0, 1, \ldots, 2^n - 1\} \to GF(2^n)$ é uma descrição algébrica da S-box $S[v]$ se e somente se existem mapas invertíveis*

$$\ell : GF(2^n) \to GF(2)^n : x \mapsto v = \ell(x)$$

$$\ell^* : \mathbb{Z}/2^n\mathbb{Z} \to GF(2)^n : z \mapsto v = \ell^*(z)$$

tais que para todo $z \in \mathbb{Z}/2^n\mathbb{Z}$, $S[\ell^*(z)] = \ell(s(z))$.

Um problema com esta definição é que para todo $b \in \mathrm{GF}(2^n)$ (exceto $b = 0$), vale que $b^0 = b^{2^n-1} = 1$; além disto, nenhuma potência de b satisfaz $b^i = 0$. Assim, é necessário mapear um dos dois elementos $\ell^*(0), \ell^*(2^n - 1)$ para $\ell(0)$; a escolha pelo mapeamento de $\ell^*(2^n-1)$ para $\ell(0)$ simplifica as demonstrações a seguir e será adotada.

O mapa ℓ^* adotado será o mapa canônico ℓ_c^* que converte um inteiro em sua representação binária. Isto é, se

$$z = \sum_{i=0}^{n-1} c_i 2^i,$$

então

$$\ell_c^*(z) = [c_{n-1}, c_{n-2}, \ldots, c_0].$$

Lema 5. *Para todo* $z \in \mathbb{Z}/2^n\mathbb{Z}$,

$$r(\ell_c^*(z)) = \ell_c^*(2z \bmod 2^n - 1).$$

Demonstração. Seja

$$z = 2^{n-1}c_{n-1} + 2^{n-2}c_{n-2} + \ldots + c_0.$$

Então

$$2z = 2^n c_{n-1} + 2^{n-1}c_{n-2} + \ldots + 2c_0 \equiv 2^{n-1}c_{n-2} + \ldots + 2c_0 + c_{n-1} \pmod{2^n - 1},$$

uma vez que $2^n \equiv 1 \pmod{2^n - 1}$. □

Temos então o seguinte resultado sobre S-boxes baseadas em mapas de exponenciação.

Teorema 7. *S-boxes baseadas em mapas exponenciais são invariantes por rotação, usando os mapas rotuladores* ℓ_n *e* ℓ_c^*.

Demonstração. Pela Definição 26, $S[\ell_c^*(z)] = \ell_n(s(z))$. Aplicando os Lemas 4 e 5, temos

$$\mathrm{r}(S[\ell_c^*(z)]) = \mathrm{r}(\ell_n(s(z))) = \ell_n(s(z)^2)$$

$$S[\mathrm{r}(\ell_c^*(z))] = S[\ell_c^*(2z \bmod 2^n - 1)] = \ell_n(s(2z \bmod 2^n - 1)).$$

Seja $s(z) = b^z$ para algum $b \in \mathrm{GF}(2^n)$ e $z \in \mathbb{Z}/2^n\mathbb{Z}$. Temos

$$s(2z \bmod 2^n - 1) = b^{2z \bmod 2^n - 1} = b^{2z} = (b^z)^2 = s(z)^2,$$

uma vez que $b^{2^n-1} = 1$. Isto estabelece que a S-box é invariante por rotação para $z < 2^n - 1$; resta o caso $z = 2^n - 1$, que pode ser verificado diretamente. Como $\ell_c^*(2^n - 1)$ é o vetor composto apenas por 1s, temos que $\mathrm{r}(\ell_c^*(2^n - 1)) = \ell_c^*(2^n - 1)$. Com relação ao outro mapa, lembrando que foi definido $s(2^n - 1) = 0$, temos

$$\mathrm{r}(\ell_n(s(2^n - 1))) = \mathrm{r}(\ell_n(0)) = \mathrm{r}(0) = 0 = \ell_n(0) = \ell_n(s(2^n - 1)).$$

□

Observa-se que é fácil estender o teorema para somas e produtos de S-boxes baseadas em mapas de exponenciação.

Finalmente, é possível combinar os Teoremas 6 e 7 e demonstrar o seguinte corolário.

Corolário 2. *Para S-boxes descritas em base normal, uma S-box baseada em mapas de exponenciação também pode ser escrita como um polinômio com coeficientes binários.*

4.2 Invariância por rotação em outras bases

Lema 6. *Sejam B_1, B_2 duas bases diferentes para GF(2)^n, e ℓ_1, ℓ_2 os mapas rotuladores correspondentes. Então existe uma matriz Q tal que*

$$\ell_1(x) = Q\ell_2(x), \forall x.$$

Demonstração. O lema é consequência do teorema de álgebra linear que afirma que as coordenadas de um vetor v, numa base B_2, estão relacionadas com as coordenadas de v na base B_1 por uma transformação linear (LANG, 1987). □

Definição 27. *Duas S-boxes $S[v]$ e $T[v]$ são linearmente equivalentes se existirem matrizes A e B tais que $S[v] = AT[Bv]$ para todos os vetores v.*

Teorema 8. *Toda S-box que pode ser descrita algebricamente por um polinômio com coeficientes binários (numa base arbitrária) é invariante por rotação.*

Demonstração. Para o caso de bases normais, este resultado é apenas uma reafirmação do Teorema 6.

Mais geralmente, seja $s_1(x)$ um polinômio com coeficientes binários que descreve uma S-box $S[v]$ usando um mapa rotulador ℓ_1 empregando uma base não necessariamente normal. Então

$$s_1(x) = \ell_1^{-1}(S[\ell_1(x)]).$$

Seja ℓ_2 o mapa rotulador para uma base normal, e seja Q tal que $\ell_1(x) = Q\ell_2(x)$. Define-se a S-box $T[v] = Q^{-1}S[Qv]$, que é linearmente equivalente a $S[v]$. Então

$$s_1(x) = \ell_2^{-1}(Q^{-1}S[Q\ell_2(x)]) = \ell_2^{-1}(T[\ell_2(x)])$$

é uma descrição algébrica da S-box $T[v]$ numa base normal. Como $s_1(x)$ é um

polinômio com coeficientes binários, a S-box deve ser invariante por rotação.

□

Por exemplo, a S-box da cifra Rijndael é baseada no mapa $x^{-1} = x^{255-1}$ sobre GF(2^8) seguida de uma transformação linear, e portanto é linearmente equivalente a uma S-box invariante por rotação. Este fato já foi observado anteriormente em (FULLER; MILLAN, 2003).

4.3 Sumário

Neste capítulo, é identificada uma classe particular de S-boxes, as S-boxes invariantes por rotação. A simetria presente nestas S-boxes permite implementação eficiente em hardware e bitslicing, como discutido no início do capítulo. Quando descritas algebricamente sobre uma base normal de GF(2^8) (Seção 4.1), é possível desenvolver uma rica teoria destas S-boxes, culminando com o resultado que as S-boxes invariantes por rotação podem ser descritas algebricamente por polinômios com coeficientes binários (Teorema 6) e mapas de exponenciação (Teorema 7), e que todas as S-boxes com estas descrições são invariantes por rotação. S-boxes com estas descrições algébricas, porém em bases polinomiais, são usadas por exemplo na cifra Rijndael e como parte da definição da S-box da função de hash WHIRLPOOL. A Seção 4.2 estende os resultados sobre bases normais para bases arbitrárias, em particular generalizando os Teoremas 6 e 7 para bases arbitrárias (Teorema 8).

5 A FUNÇÃO DE HASH MAELSTROM-0

Um dos objetivos desta dissertação é a proposta de novas primitivas criptográficas baseadas na estratégia de trilha larga, oferecendo alternativas às primitivas já existentes, que incorporam as vantagens de eficiência e resistência a criptanálise fornecidas pela estratégia de trilha larga. Este capítulo se ocupa de uma destas propostas, a função de hash MAELSTROM-0, proposta como alternativa à família SHA-2 de funções de hash (NIST, 2002), cuja segurança é questionada em face a ataques recentes à função SHA-1 (WANG; YIN; YU, 2005), predecessora da família SHA-2 e que possui uma estrutura semelhante.

MAELSTROM-0 é baseada num projeto anterior de Paulo S.L.M. Barreto e Vincent Rijmen, WHIRLPOOL (BARRETO; RIJMEN, 2000c). Boa parte da estrutura de WHIRLPOOL foi reaproveitada no projeto de MAELSTROM-0, incorporando modificações visando principalmente melhora de desempenho em relação a WHIRLPOOL. Assim, a abordagem adotada neste capítulo é a descrição da função WHIRLPOOL (Seção 5.1), seguido da descrição das modificações realizadas no projeto de MAELSTROM-0 (Seção 5.2).

5.1 A função de hash WHIRLPOOL

WHIRLPOOL (BARRETO; RIJMEN, 2000c) é uma função de hash criptograficamente segura (não-invertível e resistente a colisões), projetada de acordo

com a estratégia de trilha larga, que produz resumos de 512 bits de tamanho. WHIRLPOOL sofreu duas revisões desde sua introdução; a primeira revisão introduziu uma nova S-box mais eficiente para implementação em hardware/bitslicing e ligeiramente mais segura, enquanto a segunda revisão introduziu uma nova matriz de difusão, com número de ramificação ótimo e implementação ligeiramente mais eficiente em hardware e plataformas embarcadas. Ao leitor interessado no histórico de WHIRLPOOL, recomenda-se a leitura de (BARRETO; RIJMEN, 2000c); o presente texto trata exclusivamente da revisão mais recente do algoritmo.

5.1.1 Descrição de WHIRLPOOL

Em alto nível, Whirlpool consiste na iteração (através do esquema de encadeamento de Merkle-Damgård) de uma função de compressão, baseada na construção de Miyaguchi-Preneel empregando uma cifra de bloco dedicada W, com tamanho de bloco e de chave de 512 bits.

Definição 28. *Seja uma função de compressão f com tamanho de bloco de b bits. Seja uma mensagem m de k bits de comprimento, com k múltiplo de b (caso não seja, aplica-se alguma forma de preenchimento à mensagem para satisfazer esta condição). Divide-se a mensagem em blocos $m_1, \ldots, m_n$ de b bits cada, com $n = k/b$. O esquema de encadeamento de Merkle-Damgård calcula o resumo $h(x) = H_n$ de m a partir de*

$$H_i = f(m_i, H_{i-1}),$$

com o vetor de inicialização H_0 definido a critério do projetista.

Definição 29. *Seja uma cifra de bloco $E_k(m)$, que toma como entrada uma chave k e uma mensagem m, e sejam m_i, H_i como na Definição 28. A construção de Miyaguchi-Preneel atualiza o estado interno H_i da função de hash com*

o bloco de mensagem m_i de acordo com

$$H_i = E_{H_{i-1}}(m_i) \oplus H_{i-1} \oplus m_i.$$

O processo é ilustrado graficamente na Figura 6.

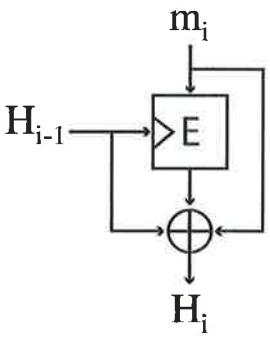

Figura 6: Construção de Miyaguchi-Preneel.

O estado interno de WHIRLPOOL é considerado como uma matriz 8×8 sobre GF(2^8). A conversão de um bloco de dados de 512 bits (interpretado como um vetor de 64 elementos de GF(2^8)) para este formato matricial é feito por uma função μ, que realiza o mapeamento canônico por linhas, ou seja,

$$\mu(a) = b \Leftrightarrow b_{ij} = a_{8i+j},\ 0 \leq i, j \leq 7.$$

A cifra de bloco W, utilizada na construção da função de compressão de WHIRLPOOL, é baseada na estratégia de trilha larga e é dividida nos seguintes passos:

1. camada não-linear γ, que consiste na aplicação de uma S-box 8×8 (descrita em detalhes na Seção 5.1.2) em paralelo a cada elemento da matriz;

2. permutação cíclica π, que consiste na rotação cíclica das colunas da matriz pelo índice de cada coluna;

3. camada de difusão linear θ, que consiste na aplicação do código MDS $[16, 8, 9]$ definido pela matriz circulante $\text{circ}(\texttt{01}, \texttt{01}, \texttt{04}, \texttt{01}, \texttt{08}, \texttt{05}, \texttt{02}, \texttt{09})$, que possui número de ramificação máximo (9) para códigos desta dimensão;

4. aplicação de subchaves $\sigma[k]$, que consiste no XOR do estado da cifra com a matriz de subchave k da rodada atual, gerada pelo algoritmo de escalonamento de chaves descrito a seguir.

A função de rodada da cifra, $\rho[k]$, parametrizada pela subchave de rodada k, é dada por

$$\rho[k] = \sigma[k] \circ \theta \circ \pi \circ \gamma.$$

Por fim, define-se a cifra completa $W[K]$, parametrizada pela chave K, como

$$W[K] = \left(\bigcirc_{1}^{R} \rho[K^R]\right) \circ \sigma[K^0],$$

onde $K^0, \ldots, K^R$ são as subchaves de cada rodada, e R é o número de rodadas, fixado em 10 para uso com WHIRLPOOL. Observa-se que, ao contrário da cifra Rijndael, a última rodada inclui uma aplicação de θ; embora esta aplicação não adicione nada à segurança da cifra, mantém a homogeneidade das funções de rodada, o que simplifica a implementação, e em particular remove a necessidade de uma tabela pré-computada distinta para a última rodada em implementações que utilizam tabelas.

O escalonamento de chaves é definido a partir da função de rodada da cifra da seguinte maneira:

$$K^r = \rho[c^r](K^{r-1}),$$

definindo-se $K^0 = K$ e as constantes de rodada c^r como matrizes 8×8 sobre

GF(2^8) dadas por

$$c_{ij}^r = \begin{cases} S[8(r-1)+j] & \text{se } i = 0, \\ 0 & \text{caso contrário.} \end{cases}$$

5.1.2 A S-box de WHIRLPOOL

Os critérios de projeto para a S-box original de WHIRLPOOL (gerada pseudo-aleatoriamente) são:

- Uniformidade diferencial $\delta < 8 \times 2^{-8}$;
- Viés linear $\lambda < 16 \times 2^{-6}$;
- Grau álgebrico máximo $\nu = 7$;
- Inexistência de pontos fixos (i.e. u tal que $S[u] = u$)[1].

A falta de estrutura na S-box obtida, devido à natureza do processo de geração, dificulta a implementação da mesma em hardware e em bitslicing. Com isso, uma revisão de WHIRLPOOL foi proposta, com uma S-box estruturada que ainda atende aos critérios de projeto acima. Esta estrutura é ilustrada na Figura 7. Matematicamente, a S-box é descrita como um mapeamento S : GF(2^4)^2 $\rightarrow$ GF(2^4)^2, $S(u, v) = (u', v')$, onde

$$u' = E(E(u) \oplus R(E(u) \oplus E^{-1}(v)))$$
$$v' = E^{-1}(E^{-1}(v) \oplus R(E(u) \oplus E^{-1}(v))).$$

[1]Embora esta restrição não produza melhora demonstrável na segurança, foi inserida mesmo assim para reduzir o espaço de busca.

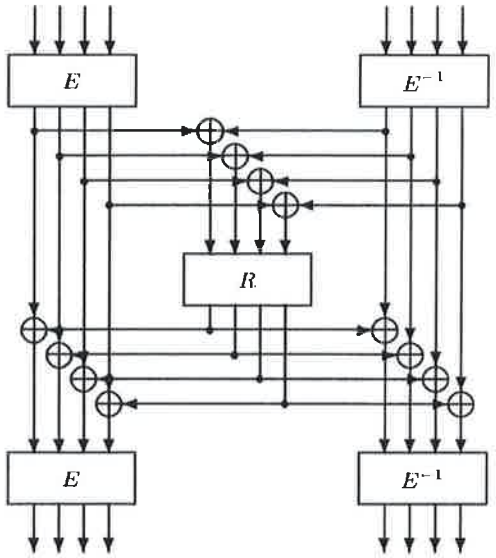

Figura 7: Estrutura da S-box de WHIRLPOOL.

A S-box auxiliar E é uma S-box 4×4 baseada num mapa de exponenciação[2], a saber:

$$E : \mathrm{GF}(2^4) \to \mathrm{GF}(2^4) : E(u) = \begin{cases} \mathtt{B}^u & \text{se } u \neq \mathtt{F}, \\ \mathtt{0} & \text{caso contrário,} \end{cases}$$

onde os valores são dados em notação hexadecimal (base 16), e u é interpretado como o inteiro com a representação binária correspondente. A base B foi escolhida para garantir que E e sua inversa E^{-1} não possuam pontos fixos e nem ciclos de ordem 2 (i.e. $E(E(u))$). Nota-se que esta S-box possui parâmetros δ, λ, ν ótimos.

A S-box auxiliar R foi gerada de maneira pseudo-aleatória, verificando que R possua parâmetros δ, λ, ν ótimos, e testando se a estrutura completa da S-box S satisfaz os critérios de projeto listados no início da seção. Na verdade, foi obtida uma S-box com um parâmetro $\lambda = 14 \times 2^{-6}$, ligeiramente melhorado em relação ao valor listado nos critérios de projeto.

[2]Embora seja baseada num mapa de exponenciação, é utilizada uma base polinomial e não normal, de maneira que os resultados do Capítulo 4 não se aplicam diretamente.

Tabela 2: As S-boxes auxiliares de WHIRLPOOL, em notação hexadecimal.

S-box	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E	1	B	9	C	D	6	F	3	E	8	7	4	A	2	5	0
E^{-1}	F	0	D	7	B	E	5	A	9	2	C	1	3	4	8	6
R	7	C	B	D	E	4	9	F	6	3	8	A	2	5	1	0

As S-boxes auxiliares são listadas em notação hexadecimal na Tabela 2.

Em (BARRETO; RIJMEN, 2000c), expressões são fornecidas para E, E^{-1}, R com 18, 18 e 17 portas lógicas, respectivamente, permitindo a implementação S-box completa com apenas 101 portas lógicas. Em comparação, uma implementação eficiente da S-box do AES exige em torno de 150 portas lógicas (WOLKERSTORFER; OSWALD; LAMBERGER, 2002).

5.2 A função de hash MAELSTROM-0

MAELSTROM-0 (GAZZONI FILHO; BARRETO; RIJMEN, 2006) é uma evolução de WHIRLPOOL (BARRETO; RIJMEN, 2000c), projetado para obter melhor desempenho e incorporar novos desenvolvimentos criptográficos desde o projeto de WHIRLPOOL, em particular medidas de proteção contra ataques de colisão multi-bloco, a base dos ataques recentes contra as principais funções de hash em uso corrente, como MD4, MD5 e SHA-1 (WANG; YU, 2005; WANG; YIN; YU, 2005). As modificações feitas em MAELSTROM-0 em relação a WHIRLPOOL são detalhadas na seção seguinte.

5.2.1 Descrição de MAELSTROM-0

MAELSTROM-0, assim como WHIRLPOOL, produz resumos com 512 bits de tamanho; no entanto, o processamento é realizado sobre blocos de 1024 bits da mensagem, diferentemente do WHIRLPOOL, que processa blocos de 512

bits por vez. A vantagem de dobrar o tamanho de bloco é que para uma dada mensagem, o número de aplicações da função de compressão é reduzido pela metade. Se fosse mantida a estrutura de WHIRLPOOL, em particular a construção de Miyaguchi-Preneel, seria necessário dobrar o tamanho de bloco da cifra de bloco empregada na função de compressão, o que aproximadamente dobraria o custo de implementação da cifra de bloco e anularia o ganho de desempenho presumido pelo novo tamanho de bloco. Com isso, a função de compressão foi modificada para usar a construção de Davies-Meyer ao invés de Miyaguchi-Preneel, exceto para o caso de mensagens 'curtas', como explicado em seguida.

Definição 30. *Seja uma cifra de bloco $E_k(m)$, que toma como entrada uma chave k e uma mensagem m, e sejam m_i, H_i como na Definição 28. A construção de Davies-Meyer atualiza o estado interno H_i da função de hash com o bloco de mensagem m_i de acordo com*

$$H_i = E_{m_i}(H_{i-1}) \oplus H_{i-1}.$$

O processo é ilustrado graficamente na Figura 8.

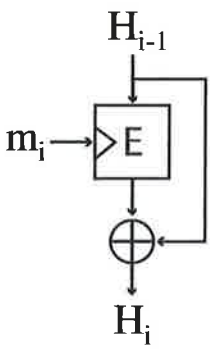

Figura 8: Construção de Davies-Meyer.

Com a construção de Davies-Meyer, é possível manter o tamanho de bloco da cifra em 512 bits, e aumentar o tamanho de chave para 1024 bits; porém,

como o escalonamento de chaves é uma operação mais leve que a função principal da cifra, o desempenho da cifra não sofre muito com esta modificação (certamente não é reduzido pela metade), e o desempenho geral da função de hash é melhorado, uma vez que o número de aplicações da cifra cai pela metade. A nova cifra projetada para uso com MAELSTROM-0, $\mathcal{M}$, é idêntica à cifra dedicada W de WHIRLPOOL, a menos do escalonamento de chaves, descrito na Seção 5.2.2.

5.2.1.1 Distinção entre mensagens curtas e longas

Em MAELSTROM-0, é feita uma distinção entre mensagens curtas e longas. Mensagens curtas são aquelas cujo comprimento é menor que a metade de um bloco, ou seja, $1024/2 = 512$ bits. (O último meio bloco da mensagem sempre armazena a representação binária do comprimento da mensagem, independente de a mensagem ser curta ou longa). Mensagens curtas não estão sujeitas ao esquema de encadeamento descrito a seguir; ao invés disso, define-se a saída de MAELSTROM-0 pela aplicação de $\mathcal{M}$ ao único bloco da mensagem usando a construção de Miyaguchi-Preneel, ou seja,

$$f(m, \mathrm{IV}) = \mathcal{M}_m(\mathrm{IV}) \oplus \mathrm{IV} \oplus m.$$

Mensagens longas são processadas usando a construção de Davies-Meyer e o esquema de encadeamento descrito a seguir.

5.2.1.2 O esquema de encadeamento

Outra modificação importante é a substituição do esquema de encadeamento de Merkle-Damgård por um membro da família 3C/3C+ (GAURAVARAM et al., 2006), com o objetivo de proteger a cifra contra ataques de colisão multi-bloco e ataques de extensão de tamanho da mensagem.

Definição 31. *Sejam m, n como na Definição 28. A família de modos de hash 3C é um esquema de encadeamento que generaliza a construção de Merkle-Damgård pelo uso de múltiplas cadeias $u_i, s_i, t_i, \ldots$. As cadeias extras são transformadas num bloco extra de mensagem $m_n = g(s_n, t_n, \ldots)$ através de alguma função g, que é processado para atualizar a primeira cadeia u_i pela última vez e produzir o estado final da função de compressão $u_{n+1} = f(m_{n+1}, u_n)$. Todas as variantes da família 3C utilizam uma segunda cadeia s_i, enquanto que a subfamília 3C+ emprega uma terceira cadeia t_i.*

Sejam f, m_i como na Definição 28. Na família 3C, a primeira cadeia u_i coincide com a cadeia H_i utilizada na construção de Merkle-Damgård, a saber:

$$u_i = f(m_i, u_{i-1}),$$

e u_0 é definido com o vetor de inicialização IV. A segunda cadeia s_i combina os elementos da primeira cadeia u_i através da operação XOR, ou seja,

$$s_i = s_{i-1} \oplus u_i,$$

onde define-se $s_0 = 0$. Para a terceira cadeia t_i, MAELSTROM-0 utiliza uma construção baseada num registrador de deslocamento com realimentação linear (LFSR). Matematicamente,

$$t_i = u_i \oplus \zeta(t_{i-1}),$$

com $t_0 =$ IV, e onde ζ é a função que mapeia seu argumento (uma string de 512 bits) para o corpo GF(2^{512}), multiplica o resultado pelo elemento primitivo x^8 e mapeia o produto de volta para uma string de bits.

Concretamente, ζ pode ser implementado como um deslocamento à esquerda por um byte e algumas operações extras sobre bytes ou half-words de 16 bits. Primeiramente, observamos que a representação de GF(2^{512}) es-

colhida para MAELSTROM-0 utiliza o polinômio primitivo $x^{512} + x^8 + x^5 + x^2 + 1$. Considere agora a forma geral do argumento v de $\zeta(v)$,

$$v = \sum_{j=0}^{511} v_j x^j = \sum_{j=0}^{63} w_j x^{8(63-j)},$$

onde na segunda formulação $w_j(x) = \sum_{k=0}^{7} v_{511-k} x^{7-k}$ é o agrupamento dos termos de $x^{8(63-j)}$ a $x^{8(63-j)+7}$. Uma representação alternativa para v é a sequência de bytes $(b_0, \ldots, b_{63})$, onde $b_j = w_j(2)$ é a representação binária de w_j. Nesta representação, a aplicação de $\zeta(v) = x^8 v$ produz a sequência $(b_1, \ldots, b_{63} \oplus c_1[b_0], c_0[b_0])$, onde

$$c[b_0] = (c_1[b_0] \ll 8) \oplus c_0[b_0] = (b_0 \ll 8) \oplus (b_0 \ll 5) \oplus (b_0 \ll 2) \oplus b_0.$$

Esta expressão é derivada do polinômio correspondente à representação escolhida para GF(2^{512}), e pode ser implementada diretamente, ou pré-computada e armazenada em duas tabelas c_1 e c_0, cada uma contendo 256 entradas de um byte.

Por fim, a escolha da função g na Definição 31 em MAELSTROM-0, assim como no esquema 3C+ básico, é a escolha mais simples, a saber, a concatenação das duas cadeias s_n e t_n:

$$g(s_n, t_n) = s_n \parallel t_n.$$

5.2.1.3 O vetor de inicialização

WHIRLPOOL define seu vetor de inicialização como $00\ldots00$, independente se o resumo produzido será truncado ou tomado módulo algum inteiro n, como pode ocorrer em algoritmos de assinatura digital. Isto abre uma brecha para um ataque de Vaudenay (VAUDENAY, 1996), em que uma colisão $H(m) \equiv H(m') \pmod{q}$ é produzida variando as mensagens m, m' até que $H(m) - H(m')$ seja primo, e toma-se $q = H(m) - H(m')$. Para evitar este ataque, MAELSTROM-0

define o IV da seguinte maneira: se o resumo obtido será reduzido módulo q (em particular, o caso $q = 2^k$ denota truncamento), então IV $= q \bmod 2^{512}$. Assim, q deve ser definido antes do cálculo do resumo, o que invalida o ataque de Vaudenay.

5.2.2 O escalonamento de chaves de $\mathcal{M}$

Uma vez que a função principal de $\mathcal{M}$ é idêntica à da cifra W de WHIRLPOOL, $\mathcal{M}$ também emprega $R + 1$ subchaves $K^0, \ldots, K^R$ (onde R é o número de rodadas, definido como 10 para uso com MAELSTROM-0[3].) de 512 bits cada. No entanto, como $\mathcal{M}$ possui tamanho de chave de 1024 bits, um novo escalonamento de chaves é necessário.

O primeiro passo é mapear a chave K para um vetor coluna $\mathcal{K}_{-1} = (\kappa_{-2}, \kappa_{-1}) \in \mathrm{GF}(2^{512}) \times \mathrm{GF}(2^{512})$ pelas expressões

$$\kappa_{-2} = \sum_{j=0}^{511} v_j x^{511-j}$$

$$\kappa_{-1} = \sum_{j=0}^{511} v_{j+512} x^{511-j}.$$

A cada passo $s \geq 0$, o vetor coluna $\mathcal{K}_{s-1} = (\kappa_{2s-2}, \kappa_{2s-1})$ é mapeado para o vetor seguinte $\mathcal{K}_s = (\kappa_{2s}, \kappa_{2s+1})$ através da transformação de evolução de chave

$$\mathcal{K}_s = \mathcal{E}\mathcal{K}_{s-1} + C_s,$$

onde a matriz $\mathcal{E}$ de dimensão 2×2 sobre $\mathrm{GF}(2^{512})$ é dada por

$$\mathcal{E} = \begin{bmatrix} 1 & 1 \\ x^8 & x^8 + 1 \end{bmatrix}$$

e o vetor-coluna $C_s = (c_{2s}, c_{2s+1})$ é obtido das constantes de rodada c^r definidas

[3]O valor de R é o mesmo utilizado na cifra W para uso com WHIRLPOOL; esta escolha é baseada na mesma análise de (BARRETO; RIJMEN, 2000c).

por (lembrando que S é a S-box da cifra)

$$c_{ij}^r = \begin{cases} \mathrm{S}[16r + j] & \text{se } i = 3, \\ \mathrm{S}[16r + 8 + j] & \text{se } i = 7, \\ 0 & \text{caso contrário.} \end{cases}$$

Nota-se que as escolhas dos elementos de S para uso com as constantes de rodada é arbitrário.

A multiplicação por x^8 em GF(2^{512}), no contexto da multiplicação pela matriz $\mathcal{E}$, é a mesma transformação ζ utilizada no encadeamento, e as mesmas tabelas pré-computadas (quando usadas) podem ser reaproveitadas. O custo total de uma aplicação da transformação de evolução de chave é um cálculo de ζ e dois XORs de 512 bits cada.

A extração das subchaves é feita através de $K^r = \psi(\kappa_r)$, onde ψ é a função de extração de chave definida da seguinte maneira:

- Primeiramente, κ_r é mapeado para uma matriz κ^r de dimensão 8×8 sobre GF(2^8);
- Em seguida, aplica-se a S-box aos elementos das linhas correspondentes a linhas não-nulas na matriz de constantes de rodada c^r (ou seja, a quarta e oitava linhas da matriz), obtendo-se uma matriz modificada $\tilde{\kappa}^r$;
- Por fim, aplica-se a camada de difusão linear θ da cifra a $\tilde{\kappa}^r$, porém apenas às mesmas linhas descritas no item anterior, enquanto que as demais linhas são copiadas sem modificação de κ^r; a matriz resultante é a subchave K^r.

Observa-se que este escalonamento de chaves é mais eficiente que o de WHIRLPOOL e possui não-linearidade superior ao escalonamento de chaves

da cifra Rijndael (GAZZONI FILHO; BARRETO; RIJMEN, 2006).

5.2.3 Implementação

A implementação de MAELSTROM-0, e mais especificamente de $\mathcal{M}$, deve ser feita de acordo com as recomendações do Capítulo 3 para obtenção de máxima eficiência.

Embora a estratégia de trilha larga não favoreça plataformas específicas, devido à disposição do estado de $\mathcal{M}$ em uma matriz 8×8 com elementos de 8 bits, a implementação é mais natural em plataformas de 8 bits ou de 64 bits. Plataformas de 16 ou 32 bits devem compor múltiplas operações para simular a aritmética de 64 bits, ao invés de restringirem-se a operações de 8 bits. Já plataformas de 128 bits ou mais (em geral, utilizando conjuntos de instruções vetoriais como SSE2 na plataforma Intel ou Altivec na plataforma PowerPC) tem a opção de empregar apenas 64 dos 128 bits disponíveis no registrador, ou de utilizar todos os 128 bits. A decisão deve ser analisada caso a caso, de acordo com diversos fatores, como o suporte do conjunto de instruções para operações em partições de 64 bits dos registradores, a quantidade de registradores disponíveis, etc.

Em termos de tabelas pré-computadas, distingue-se dois casos. A implementação em processadores embutidos, como microcontroladores e smart cards, segue a discussão da Seção 3.1.1, e exigirá apenas o armazenamento da S-box, que ocupa 256 bytes de memória. Já em sistemas como PCs e servidores, com mais memória disponível, segue a discussão da Seção 3.1.2, com o uso de 8 tabelas pré-computadas de 256 entradas de 64 bits (8 bytes) cada, totalizando 16 KB de memória. Como estas 8 tabelas são apenas versões rotacionadas uma da outra, pode ser desejável não armazenar todas as tabelas, de acordo com a disponibilidade de memória cache na CPU em questão,

e realizar as rotações durante a execução do algoritmo. Conforme discutido na Seção 5.2.1.2, a implementação eficiente da transformação ζ exige uma tabela pré-computada de 256 entradas de 16 bits (2 bytes) cada, adicionando mais 512 bytes aos requerimentos do algoritmo. Por fim, são necessários mais 176 bytes para armazenamento dos elementos da S-box empregados no escalonamento de chaves. Ao total, uma implementação utilizando o máximo de pré-computação exigirá menos de 17 KB de tabelas pré-computadas.

As seções seguintes relatam experiências de implementação de MAELSTROM-0 em duas plataformas de software.

5.2.3.1 Plataforma Intel

A plataforma Intel foi selecionada por sua dominância no mercado de computadores pessoais, workstations e servidores de pequeno porte, além de presença expressiva nos servidores de médio e grande porte.

Existem versões da plataforma Intel de 32 e 64 bits; ademais, ambas as versões contam com extensões vetoriais de 64 bits (MMX) e 128 bits (SSE). Desta forma, é interessante investigar qual destas variantes, ou mesmo uma combinação delas, possibilita uma implementação mais eficiente.

A primeira implementação de MAELSTROM-0 (GAZZONI FILHO; BARRETO; RIJMEN, 2006) optou pela versão escalar de 32 bits da arquitetura, e obteve um desempenho de 30 ciclos/byte em processadores Intel Pentium M[4]. Em processadores Intel Core 2 Duo, o valor é ligeiramente reduzido, para 28 ciclos/byte. Esta versão é uma implementação direta da cifra, empregando todas as tabelas pré-computadas discutidas na Seção 5.2.3, e particionando as operações de 64 bits em 2 operações de 32 bits. Tentativas de otimizar esta

[4]Os valores fornecidos nesta seção referem-se apenas ao 'núcleo' de MAELSTROM-0, ou seja, à cifra $\mathcal{M}$, incluindo o escalonamento de chaves, e as operações de encadeamento de blocos.

implementação além do valor obtido foram infrutíferas.

Devido ao desempenho inferior ao esperado, foram buscadas outras alternativas. Uma segunda tentativa empregou a versão escalar de 64 bits da arquitetura, que reduziu o valor para 20 ciclos/byte em processadores Intel Core 2 Duo. A principal diferença nesta versão é a substituição de pares de operações de 32 bits por uma única operação de 64 bits, o que reduz a contagem de instruções do código quase que pela metade. Poderia-se esperar que, de maneira correspondente, o tempo de execução do código fosse reduzido quase pela metade, porém não foi o que ocorreu. Teoriza-se que o aumento no tamanho de código, provocado pelo uso de instruções de 64 bits, tenha criado um gargalo no processo de decodificação de instruções pelo processador, o que limita o paralelismo na execução do código e não permite a obtenção da melhoria esperada.

Por fim, seguindo a estratégia de (NAKAJIMA; MATSUI, 2002), foi escrita uma implementação usando a extensão vetorial MMX de 64 bits. Esta extensão possui diversas vantagens, podendo-se citar sua presença em processadores tanto de 32 como de 64 bits, incluindo processadores tão antigos como o Intel Pentium MMX, e o uso de registradores próprios (distintos dos registradores escalares), permitindo a carga completa do estado da cifra $\mathcal{M}$ em registradores. Este último fator é muito importante; quando o estado da cifra não pode ser completamente carregado, caso deseje-se acessar um elemento não disponível atualmente nos registradores, é preciso armazenar um dos registradores ocupados na memória para liberá-lo, e carregar o valor desejado. Estas são operações 'desnecessárias' (no sentido de não realizarem operações úteis) que aumentam a pressão sobre o subsistema de carga e armazenamento do processador, e que forçam padrões específicos de acesso à memória para garantir que o máximo de computação entre uma carga e des-

Algoritmo	MAELSTROM-0	WHIRLPOOL	SHA-1	SHA-256	SHA-512
Desempenho (ciclos/byte por núcleo)	12	30	11	22	18

Tabela 3: Desempenho de funções de hash em processadores Intel Core 2 Duo.

carga, minimizando o número destas operações desnecessárias. Já no caso que o estado da cifra pode ser completamente carregado em registradores, é possível utilizar padrões de acesso mais vantajosos, que reduzem ainda mais a pressão sobre o subsistema de carga e armazenamento do processador.

Uma possível desvantagem é a impossibilidade de endereçar memória utilizando o conteúdo dos registradores MMX, como necessário no acesso às tabelas pré-computadas empregadas no algoritmo. Com isso, é necessário armazenar os dados na memória e carregá-los novamente nos registradores escalares, o que abre espaço para diversas ineficiências em potencial. Apesar disto, estas ineficiências não parecem ter se manifestado, e esta implementação apresentou o melhor desempenho de todas as implementações testadas, atingindo 12 ciclos/byte em processadores Intel Core 2 Duo. Para efeito de comparação, a Tabela 3 lista valores para outras funções de hash de destaque, conforme implementadas na biblioteca Crypto++ (DAI, 2007). Em particular, MAELSTROM-0 é 2,5 vezes mais rápido que WHIRLPOOL, é mais rápido que SHA-512 e competitivo até mesmo com SHA-1.

Uma possibilidade não investigada é o uso da extensão vetorial SSE de 128 bits. Pode-se argumentar que o uso desta extensão vetorial não deverá produzir uma melhora de desempenho, em pequena parte parte por aumentar o tamanho das instruções, produzindo um gargalo na decodificação de instruções como já mencionado, mas principalmente por usar registradores e acessos à memória de 128 bits. Por um lado, isto reduz a quantidade de operações

aritméticas em algumas das operações do algoritmo (como a adição de subchaves), mas não muitas. Por outro lado, isto aumenta desnecessariamente a pressão sobre o subsistema de carga e armazenamento do processador, que já é o provável gargalo no tempo de execução do algoritmo, e dobra o tamanho das tabelas pré-computadas exigidas, uma vez que apenas 64 dos 128 bits de cada elemento da tabela conterão dados, enquanto os restantes devem conter zeros. Com isso, as tabelas excederão o armazenamento disponível no cache L1 de todos os processadores Intel em existência, aumentando os tempos de acesso à memória. No entanto, uma possibilidade a ser investigada é o uso simultâneo das duas extensões vetoriais, MMX e SSE. Por exemplo, é possível realizar o escalonamento de chaves em SSE, em paralelo com a função de rodada em MMX. O mix de instruções empregadas no escalonamento de chaves poderia usufruir melhor da aritmética e acesso à memória em 128 bits, possibilitando uma melhora no desempenho de MAELSTROM-0, no melhor caso, para 9–10 ciclos/byte em processadores Intel Core 2 Duo.

5.2.3.2 Plataforma Cell

A plataforma Cell (KAHLE et al., 2005), presente no console de videogame Playstation 3 e em alguns servidores, foi selecionada por representar um novo paradigma de projeto de processadores que deverá ser explorado por projetistas no futuro. A maioria dos projetos atuais busca maximizar o desempenho de cada núcleo de execução dentro do orçamento de área/potência permitido, utilizando tecnologias como escalonamento dinâmico, execução superescalar e fora de ordem, memórias cache, etc. Em contraste, o processador Cell utiliza núcleos simples, com conjunto de instruções RISC, escalonamento estático, execução superescalar limitada e em ordem, e no lugar de caches, gerenciamento manual de memória pelo programador ou compilador. Estes núcleos

simples rodam em alta frequência de clock e ocupam uma fração do espaço de um núcleo de outros processadores contemporâneos, permitindo sintetizar uma quantidade maior destes núcleos simples na mesma área e envelope de potência. A título de exemplo, a primeira implementação de um microprocessador baseado na arquitetura Cell, contendo 9 núcleos, utiliza-se de apenas 234 milhões de transistores e ocupa uma área de 221 mm^2 num processo de fabricação de 90 nm, sendo lançado comercialmente na frequência de 3.2 GHz, embora seja capaz de atingir frequências maiores. Em comparação, um processador Intel Pentium 4, com um único núcleo, fabricado também num processo de 90 nm e atingindo frequências de clock semelhantes, emprega 125 milhões de transistores e ocupa uma área de 112 mm^2.

Obviamente, o uso de núcleos simples, e novos paradigmas como gerenciamento manual de memória, exige maior esforço por parte do programador e do compilador para obtenção de alto desempenho, além de exigir o uso de algoritmos paralelizáveis para aproveitar todos os núcleos disponíveis no processador[5]. Os projetistas da arquitetura, reconhecendo estas dificuldades, projetaram um conjunto de instruções vetorial de 128 bits focado na simplicidade e regularidade, e com diversas características de destaque, como abundância de registradores (128 ao total) e inclusão de muitas instruções 'redundantes', porém essenciais à programação de alto desempenho (por exemplo, rotações de bits e diversas instruções lógicas como NOR, NAND, etc.) Aliado a propriedades da implementação, como regras simples de escalonamento de instruções e tempo fixo de acesso à memória local, a programação em linguagem de montagem é mais produtiva que em outras arquiteturas, e permite atingir desempenho próximo do máximo teórico com facilidade.

[5]Embora funções de hash exijam processamento serial, é possível que sejam empregadas em paralelo com outras tarefas computacionais, criptográficas ou não, que ocupem os demais núcleos do processador. Também é possível considerar o caso de servidores processando múltiplas requisições simultâneas e independentes.

Por ser uma arquitetura de 128 bits, a primeira decisão a ser tomada é com relação ao uso total dos registradores de 128 bits, dispondo o estado da cifra em 4 registradores, ou o uso de apenas 64 dos 128 bits disponíveis nos registradores, dispondo o estado da cifra em 8 registradores. Para esta arquitetura, nenhum fator favorece a segunda opção, exceto possivelmente a necessidade de utilizar tabelas pré-computadas aproximadamente quatro vezes maiores[6]. Com 256 KB de armazenamentos local disponível para cada núcleo, este aumento no tamanho das tabelas ainda se mostra viável. Já o uso de palavras de 128 bits permite a economia de operações em boa parte do escalonamento de chaves, na adição de subchaves da função de rodada, na função de compressão (construção de Davies-Meyer) e encadeamento (3CM). Portanto, optou-se pelo uso total dos registradores de 128 bits.

Tomada esta decisão, o restante da implementação é imediato. Vale notar que é possível escalonar muitas das operações em paralelo, para aproveitar a capacidade de execução superescalar de 2 vias do processador, mas na implementação realizada, ainda existe um potencial para escalonar cerca de 20% das operações em paralelo com outras operações, o que poderia reduzir o tempo de execução por um valor correspondente. No entanto, é difícil enxergar uma maneira de realizar tal escalonamento sem mudar significativamente a estratégia de implementação. Mesmo assim, o desempenho de MAELSTROM-0 nesta plataforma é respeitável, atingindo um valor de 16 ciclos/byte. A Tabela 4 compara o valor obtido com o desempenho de alguns membros da família SHA, conforme implementados pelo próprio fabricante do chip (CHEN et al., 2007), e como no caso da plataforma Intel, observa-se um desempenho de MAELSTROM-0 superior a funções de porte semelhante

[6]A necessidade de se empregar tabelas quatro vezes maiores vem do fato que cada entrada da tabela usa 128 ao invés de 64 bits, contribuindo com um fator de 2 vezes, e que são necessárias duas cópias de cada tabela, uma com os dados na metade mais significativa do registrador, e outra com os dados na metade menos significativa do registrador.

Algoritmo	MAELSTROM-0	MD5	SHA-1	SHA-256
Desempenho (ciclos/byte/núcleo)	16	10	12	30

Tabela 4: Desempenho de funções de hash em processadores Cell.

(SHA-256, neste caso), e competindo até mesmo com MD5 e SHA-1.

5.3 Sumário

Iniciou-se o capítulo com a discussão da crise das funções de hash, provocada pelas recentes quebras (parciais ou totais) das principais funções de hash em uso corrente. Com isso, a necessidade de uma nova função de hash segura torna-se evidente; esta necessidade foi identificada inclusive pelo governo americano, que tomou a iniciativa de propor um concurso, nos moldes do AES, para criação de um novo padrão de funções de hash para substituir a família SHA. Um dos algoritmos ainda intactos é WHIRLPOOL (Seção 5.1), principalmente em virtude de ser projetado de acordo com a estratégia de trilha larga, porém seu desempenho deixa a desejar. Identificando esta lacuna de funções de hash seguras e de alto desempenho, é proposta uma nova função de hash, MAELSTROM-0 (Seção 5.2), baseada em WHIRLPOOL, mas substancialmente modificada para obtenção de melhor desempenho e proteção contra ataques de colisão multi-bloco (a classe de ataques responsáveis pela crise das funções de hash). As seguintes modificações merecem destaque:

- o tamanho de bloco de MAELSTROM-0 foi dobrado em relação a WHIRLPOOL;
- a função de compressão emprega a construção de Davies-Meyer no lugar da construção de Miyaguchi-Preneel;
- a cifra de bloco utilizada na função de compressão é baseada na cifra

W de WHIRLPOOL, porém com chaves de 1024 bits e um novo escalonamento de chaves mais eficiente que o original;

- um novo esquema de encadeamento (baseado na família 3C/3C+) foi adotado;
- a escolha de IVs leva em conta a truncação (ou, em geral, redução módulo q) do resumo calculado;
- foi adotado um modo de processamento de mensagens distinto para mensagens 'curtas'.

Com as modificações propostas, MAELSTROM-0 é uma função de hash mais competitiva que WHIRLPOOL e um sério competidor para o novo padrão de funções de hash. Esta conclusão é suportada pelos resultados de implementação em plataformas Intel e Cell, que mostram resultados superiores a outras funções de porte semelhante, como da família SHA-2, e competitivos até com funções mais simples como SHA-1 e MD5.

6 A CIFRA DE BLOCO FUTURE

FUTURE é uma cifra de bloco destinada a aplicações legadas, devido ao tamanho de bloco de 64 bits, vulnerável a certos ataques que são atualmente ineficazes contra blocos de 128 bits e além, como os usados na cifra Rijndael. Apesar disso, ainda existe espaço para o uso destas cifras legadas, por exemplo como alternativa ao DES em aplicações onde a modificação do tamanho de bloco é inviável, mas um nível mais alto de segurança é desejado. Muitas destas aplicações estão associadas a dispositivos de baixo custo, que devem ser implementadas em hardware com área bastante reduzida, ou em microcontroladores e smart cards com pouca memória, o que enfatiza ainda mais a importância da eficiência de uma cifra desta classe.

6.1 Descrição de FUTURE

FUTURE é uma cifra de bloco com tamanho de bloco de 64 bits e tamanho de chave de 128 bits, projetada de acordo com a estratégia de trilha larga, com duas características que merecem destaque especial: sua estrutura involucional e escalonamento de chaves cíclico. Por ser uma involução, assim como o DES, os algoritmos de cifração e decifração são idênticos (a menos do escalonamento de chaves, que deve ser aplicado em ordem reversa); esta estrutura já foi estudada em outras cifras baseadas na estratégia de trilha larga, como KHAZAD (BARRETO; RIJMEN, 2000b), ANUBIS (BARRETO; RIJMEN, 2000a) e

mais recentemente CURUPIRA (BARRETO; SIMPLÍCIO, 2007). Uma vantagem da estrutura involucional é que não é necessário instanciar duas versões do algoritmo, ocupando mais área em hardware ou memória de código em software, e outra é que torna o algoritmo menos suscetível a ataques que dependem de assimetrias entre os algoritmos de cifração e decifração, como o ataque bumerangue (WAGNER, 1999). Já o escalonamento de chaves cíclico de FUTURE não requer a computação sequencial de subchaves, podendo ser calculadas em qualquer ordem.

Dentre as propostas semelhantes a FUTURE, podemos listar as cifras PRESENT (BOGDANOV et al., 2007) e NOEKEON (DAEMEN et al., 2000). No entanto, será mostrado que FUTURE é mais leve que PRESENT, e com relação a NOEKEON, embora seja uma cifra bastante leve, não possui tamanho de bloco de 64 bits e portanto é incompatível com as aplicações que FUTURE contempla.

A estrutura de FUTURE é similar à de outras primitivas baseadas na estratégia de trilha larga, porém opera sobre nybbles ao invés de bytes: o estado da cifra é representado como uma matriz 4×4 sobre GF(2^4). As operações que compõem a cifra são, como de costume, uma camada não-linear σ, uma camada de permutação π, uma camada linear de difusão μ, e a aplicação de subchaves α. A função de rodada assim formada é aplicada 15 vezes, além de uma aplicação de subchaves extra no início da cifra. A menos de α, todos os componentes possuem particularidades específicas a FUTURE discutidas nas seções seguintes.

6.1.1 A S-box de FUTURE

A camada não-linear de FUTURE consiste na aplicação de uma S-box 4×4 aos 16 nybbles do estado da cifra em paralelo.

FUTURE representa a primeira aplicação do conceito de S-boxes invariantes por rotação, discutidas no Capítulo 4. No entanto, como não existe nenhuma S-box involutiva invariante por rotação com parâmetros ótimos ($\delta = 1/4, \lambda = 1/2, \nu = 3$)[1], foi empregada uma S-box quase invariante por rotação. Especificamente, seja

$$\phi(a,b,c,d) = (b \wedge ((a \wedge c) \oplus d)) \oplus \overline{(c \vee d)}.$$

Para um elemento de GF(2^4) da forma $ax^3 + bx^2 + cx + d$, o efeito da aplicação da S-box é $S[ax^3 + bx^2 + cx + d] = ex^3 + fx^2 + gx + h$, onde

$$e = \phi(a,b,c,d),$$

$$h = \phi(b,c,d,a),$$

$$g = \phi(c,d,a,b),$$

$$f = \phi(d,a,b,c).$$

Nota-se a permutação dos bits de saída f e h, de maneira que S não é estritamente invariante por rotação, porém observa-se uma relação clara, particularmente no que tange à implementação, com o conceito de invariância por rotação. Diz-se portanto que S é *quase* invariante por rotação. Uma implementação direta da função exigiria 20 portas lógicas, porém é possível reescrever a expressão para utilizar apenas 18 portas lógicas num circuito com profundidade lógica de 3 estágios, a saber:

1. $t_0 \leftarrow a \wedge c$, $t_1 \leftarrow b \wedge d$, $t_2 \leftarrow \overline{c \vee d}$, $t_3 \leftarrow \overline{b \vee c}$, $t_4 \leftarrow \overline{a \vee b}$, $t_5 \leftarrow \overline{a \vee d}$;

2. $t_6 \leftarrow t_0 \wedge b$, $t_7 \leftarrow t_1 \wedge a$, $t_8 \leftarrow t_0 \wedge d$, $t_9 \leftarrow t_1 \wedge c$, $t_{10} \leftarrow t_1 \oplus t_2$, $t_{11} \leftarrow t_0 \oplus t_3$, $t_{12} \leftarrow t_1 \oplus t_4$, $t_{13} \leftarrow t_0 \oplus t_5$;

3. $e \leftarrow t_6 \oplus t_{10}$, $f \leftarrow t_7 \oplus t_{11}$, $g \leftarrow t_8 \oplus t_{12}$, $h \leftarrow t_9 \oplus t_{13}$.

[1]Observa-se que isso não impede que S-boxes com parâmetros sub-ótimos sejam empregadas na construção de S-boxes estruturadas 8×8, como em WHIRLPOOL e MAELSTROM-0.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	F	6	3	2	9	A	1	B	C	4	5	7	8	E	D	0

Tabela 5: A S-box de FUTURE.

A contribuição deste estágio para uma implementação em hardware ou bitslicing é de $16 \times 18 = 288$ portas lógicas. A representação tabular desta S-box é dada na Tabela 5.

6.1.2 A camada de permutação

FUTURE emprega uma camada de permutação baseada na seguinte involução, ilustrada graficamente na Figura 9:

$$\pi(a) = b \Leftrightarrow b_{i,j} = a_{i,i \oplus j}.$$

Por ser uma permutação, não é gerada lógica em hardware (apenas roteamento de sinais), e nem instruções em bitslicing.

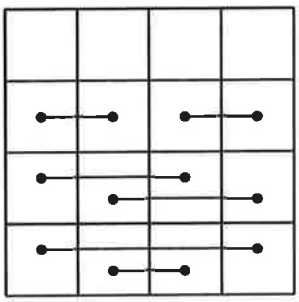

Figura 9: Camada de permutação de FUTURE.

6.1.3 A camada linear de difusão

A difusão em FUTURE é realizada por multiplicação à esquerda do estado da cifra pela matriz

$$M = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

Esta matriz gera um código linear de distância 4, e portanto o número de ramificação desta transformação é também 4. Uma vantagem desta matriz é que possui coeficientes restritos a GF(2), não exigindo multiplicação por elementos de $GF(2^4)$; fatorando expressões comuns, esta camada pode ser implementada em hardware ou bitslicing com apenas 96 portas lógicas XOR.

6.1.4 O escalonamento de chaves

Devido a problemas encontrados na versão atual do escalonamento de chaves de FUTURE, não foi possível incluir uma versão corrigida dentro do prazo de publicação deste trabalho.

6.1.5 Implementação

A implementação de FUTURE em software é bastante compacta, podendo ser realizada diretamente, sem o uso de tabelas pré-computadas. Em outras cifras baseadas na estratégia de trilha larga, a finalidade principal do uso de tabelas pré-computadas é a eliminação da aritmética em $GF(2^n)$ implícita na camada linear da cifra. Como a matriz de difusão de FUTURE emprega apenas elementos de GF(2), não há necessidade de tabelas pré-computadas.

Ainda é preciso armazenar uma cópia da S-box, porém isto requer apenas

16 bytes de armazenamento (é possível reduzir este valor para 8 bytes, compactando dois nybbles em um único byte, à custa de aumento no tamanho de código para lidar com esta representação). Plataformas que disponham de 256 bytes de memória podem armazenar uma versão da S-box para todos os pares de nybbles possíveis, totalizando $16^2 = 256$ elementos, o que permite aplicar a S-box a dois elementos simultaneamente, usando apenas um acesso à memória.

O estado da cifra pode ser armazenado utilizando um byte por nybble, ou compactando dois nybbles em um único byte. A segunda opção permite reduzir o número de operações na aplicação de subchaves, o número de cargas e armazenamentos (admitindo plataformas limitadas, com uma quantidade insuficiente de registradores para conter todo o estado da cifra), e pode ser utilizada diretamente em conjunto com a versão de 256 bytes da S-box. Em contrapartida, a aplicação da camada de permutação e de difusão exige deslocamentos e máscaras de bits para alinhamento dos nybbles dentro de um byte. Já a primeira opção possui as propriedades opostas. Devido às vantagens e desvantagens balanceadas entre cada representação, a decisão deve ser tomada caso a caso, em função da arquitetura empregada, recursos disponíveis e outros fatores.

Em processadores com tamanhos de palavra de 32 bits ou mais, a implementação direta de FUTURE não será tão eficiente. Porém, uma possibilidade mais interessante a ser investigada nessas plataformas é a implementação em bitslicing. A estrutura de FUTURE é particularmente apropriada para implementação em bitslicing, devido a seus componentes leves como a S-box quase invariante por rotação (18 portas lógicas por aplicação, ou 288 por rodada) e a camada de difusão (96 portas lógicas por rodada). Somando as 64 portas lógicas para aplicação de subchaves, uma rodada de FUTURE pode ser

implementada com $288 + 96 + 64 = 448$ portas lógicas. Uma vez que a cifra é composta de 15 rodadas, e mais uma aplicação de subchaves extra no início, obtém-se um total de $448 \times 15 + 64 = 6784$ portas lógicas para a cifra completa.

Em bitslicing, FUTURE expõe muitas possibilidades de paralelismo ao programador:

- paralelismo total (de 64 vias) na S-box: entre cada bit da S-box, devido à quase invariância por rotação, e entre as aplicações independentes da S-box aos 16 elementos do estado da cifra;
- paralelismo dentro da multiplicação da matriz de difusão, e entre os bits de cada elemento;
- paralelismo total (de 64 vias) na aplicação de subchaves.

Ao total, é possível implementar cada rodada de FUTURE com uma profundidade lógica de 3 estágios para a S-box, 2 estágios para a camada de difusão linear e 1 estágio para a aplicação de subchaves. Tamanho paralelismo garante que FUTURE será capaz de aproveitar todos os recursos de qualquer processador disponível atualmente, e mesmo de projetos futuros.

Assim, decidiu-se investigar a implementação de FUTURE em bitslicing nas plataformas Intel e Cell. A justificativa para escolha destas plataformas é dada nas Seções 5.2.3.1 e 5.2.3.2. Para fornecer uma referência ao desempenho obtido, também foi implementada a cifra PRESENT (BOGDANOV et al., 2007), que é brevemente exposta na seção seguinte.

6.1.5.1 A cifra PRESENT

PRESENT é uma cifra de bloco de 64 bits orientada a hardware, com chaves de 80 ou 128 bits. A estrutura de PRESENT é bastante simples, sendo

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Tabela 6: A S-box de PRESENT.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P[x]$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P[x]$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P[x]$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
x	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P[x]$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Tabela 7: A permutação de bits de PRESENT.

composta de três camadas:

- Aplicação de subchaves;
- Aplicação de uma S-box 4×4 (listada na Tabela 6) a todos os nybbles do estado da cifra;
- Permutação de bits (listada na Tabela 7).

A Figura 10 ilustra uma rodada de PRESENT. A cifra completa é formada pela composição de 31 rodadas desta forma, e uma aplicação de subchaves final.

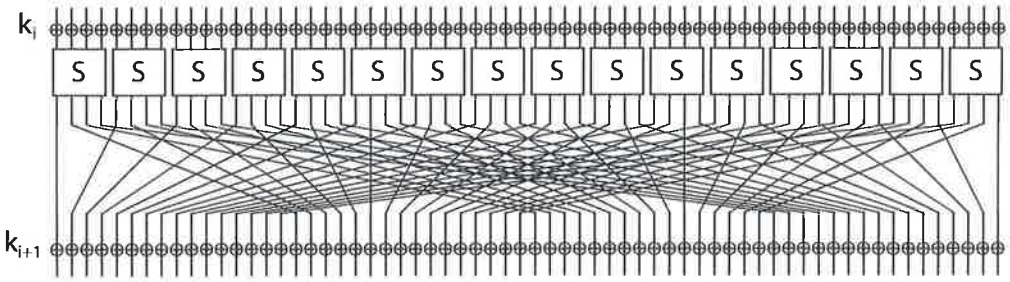

Figura 10: A função de rodada de PRESENT.

O escalonamento de chaves de PRESENT é feito pela rotação do registrador de chaves à esquerda por 61 posições, seguida por uma aplicação da S-box ao nybble mais significativo do registrador (para chaves de 80 bits) ou aplicação da S-box aos dois nybbles mais significativos do registrador (para chaves de 128 bits), e a adição de um contador de rodada. Extrai-se a chave de rodada tomando os 64 bits menos significativos do contador de rodada.

Deve-se observar que PRESENT, por ser orientada a hardware, não leva em conta a eficiência de implementação em software. Em particular, a etapa de permutação de bits representa um obstáculo para implementação eficiente em software, a menos que tabelas pré-computadas sejam empregadas. Contempla-se uma implementação relativamente eficiente de PRESENT em processadores de 32 ou 64 bits, pelo o uso de 8 tabelas, cada uma com 256 entradas de 64 bits cada, correspondendo a todos pares de nybbles possíveis na entrada de cada par de S-boxes, e às permutações de bits correspondentes para cada caso. No entanto, a implementação de PRESENT em bitslicing é bastante simples, desde que se obtenha uma expressão para a S-box de PRESENT. Utilizando uma versão modificada de um programa de busca de Brian Gladman (GLADMAN, 1998), obteve-se a seguinte expressão booleana $(e, f, g, h) = S(a, b, c, d)$, com custo de 15 portas lógicas:

$$t_1 = a \vee b;$$

$$t_2 = b \oplus c;$$

$$t_3 = c \wedge t_2;$$

$$t_4 = d \oplus t_3;$$

$$e = a \oplus t_4;$$

$$t_6 = a \vee t_4;$$

$$t_7 = t_2 \wedge t_6;$$

$$t_8 = t_1 \oplus t_7;$$

$$f = e \oplus t_8;$$

$$t_{10} = \overline{t_4};$$

$$t_{11} = c \oplus t_8;$$

$$t_{12} = t_2 \vee t_{10};$$

$$h = t_{11} \oplus t_{12};$$

$$t_{14} = t_{16} \wedge h;$$

$$g = t_{10} \oplus t_{14}.$$

Com isto, a implementação de PRESENT exige $16{\times}15 = 240$ portas lógicas por rodada para as S-boxes, e mais 64 portas lógicas para a aplicação de subchaves, totalizando 304 portas lógicas para a função de rodada. Como PRESENT consiste de 31 rodadas e mais uma aplicação de subchaves ao final, obtém-se um total de $31 \times 304 + 64 = 9488$ portas lógicas, um valor 40% superior ao de FUTURE.

6.1.5.2 Plataforma Intel

A implementação de cifras em bitslicing na plataforma Intel é mais vantajosa se feita empregando a extensão vetorial SSE, devido ao uso de aritmética de 128 bits. Vale observar que o desempenho sofre um pouco pela oferta limitada de operações lógicas em SSE; dadas variáveis a, b, são fornecidas instruções apenas para o cálculo de $a \vee b$, $a \wedge b$, $a \wedge \overline{b}$ e $a \oplus b$; nem mesmo uma operação de negação é fornecida, sendo necessário gerar uma constante com todos os bits setados, aumentando ainda mais a pressão sobre o escasso banco de registradores da arquitetura, e calcular $11\ldots11 \oplus a$ ou $11\ldots11 \wedge \overline{a}$.

Outro problema da arquitetura é o uso de instruções com apenas dois argumentos, da forma `dest = dest OP fonte` para uma operação `OP`. Com isso, quando `dest` não pode ser sobrescrito, é preciso incluir uma instrução somente para fazer uma cópia deste registrador, um desperdício desnecessário de recursos.

A implementação de PRESENT nesta plataforma é direta e não necessitou de nenhuma estratégia especial para a função de rodada. O desempenho obtido é de 7,3 ciclos/byte/núcleo em processadores Intel Core 2 Duo. Já a implementação do escalonamento de chaves apresentou um problema: o código da função de rodada e escalonamento de chaves de PRESENT é grande demais para realizar 31 cópias, correspondentes às 31 rodadas, e ainda ser armazenado no cache L1 de instruções de processadores atualmente disponíveis. Assim, é preciso inserir a função de rodada num laço. No entanto, isto impede que a aritmética de índices do array de subchaves, necessária devido à rotação de bits presente no escalonamento de chaves, seja realizada em tempo de compilação, pois é diferente para cada rodada. Ao invés de calcular os endereços manualmente para cada acesso ao array, a solução implementada foi o uso de duas cópias (contíguas na memória) do array de chaves, atualizadas simultaneamente quando há uma escrita no array. Como a quantidade de escritas é limitada (apenas aplicação de uma S-box e de uma constante de rodada para cada atualização), o desempenho não sofre com esta duplicação, ao mesmo tempo que permite a leitura do array sem cálculo de endereço em tempo de execução. Incluindo o escalonamento de chaves, o desempenho de PRESENT é de 8,7 ciclos/byte/núcleo em processadores Intel Core 2 Duo.

A implementação de FUTURE nesta plataforma também é direta, porém não tão eficiente quanto desejável. Pode-se listar duas razões principais para

isto. A primeira é que a implementação das S-boxes é relativamente ineficiente, pelas razões expostas acima (oferta limitada de operações lógicas, instruções de apenas dois argumentos, pequena quantidade de registradores). A segunda, também devida à pequena quantidade de registradores, é a grande quantidade de cargas e armazenamentos; cada passo (aplicação de S-boxes, camada linear e aplicação de subchaves) é precedido por uma carga e seguido por um armazenamento. Assim, obteve-se um desempenho de apenas 6,7 ciclos/byte/núcleo nesta plataforma, ainda assim ligeiramente mais rápido que PRESENT.

6.1.5.3 Plataforma Cell

A plataforma Cell é particularmente eficiente para implementações de cifras em bitslicing; os obstáculos verificados na plataforma Intel não existem em Cell. Em particular, a oferta de operações lógicas é bastante farta, contendo as operações $a \vee b$, $a \wedge b$ e $a \oplus b$ e variações com um dos argumentos negados, ou com o resultado negado (no caso de $a \oplus b$, estas variações coincidem). Uma maneira alternativa de analisar a situação é que negações são gratuitas em Cell. Isto abre caminho para pesquisa de S-boxes com representações mais compactas sob esta hipótese de negações gratuitas.

A implementação de PRESENT em Cell é direta. O grande banco de registradores permite o armazenamento de todo o estado da cifra em registradores, exigindo acessos à memória apenas para a carga das subchaves a serem aplicadas. O desempenho obtido foi de 8,9 ciclos/byte/núcleo. A inclusão do escalonamento de chaves não gerou o problema observado na implementação em plataforma Intel, devido ao amplo espaço disponível no armazenamento local de cada núcleo (256 KB), que permitiu a implementação do código sem nenhum laço. Incluindo o escalonamento de chaves, o desempenho é de 9,5

ciclos/byte/núcleo.

A implementação de FUTURE em Cell também é direta, e obtém um desempenho de 6,6 ciclos/byte/núcleo, 35% superior ao de PRESENT sem o escalonamento de chaves.

É instrutivo observar como as diferenças no conjunto de instrução compensam a simplicidade dos núcleos da plataforma Cell. Embora o processador Intel Core 2 Duo tenha capacidade de executar até 3 operações lógicas SSE em paralelo por ciclo, enquanto a implementação atual de Cell está restrita a uma única operação lógica por ciclo, os desempenhos obtidos nas implementações de PRESENT e FUTURE em bitslicing são comparáveis, devido às diferenças arquiteturais já discutidas.

6.2 Sumário

Este capítulo descreveu FUTURE, uma cifra de bloco de nível legado (tamanho de bloco de 64 bits e tamanho de chaves de 128 bits) projetada de acordo com a estratégia de trilha larga, de implementação bastante eficiente, particularmente em plataformas embarcadas (hardware ou software). Diversas propriedades de FUTURE contribuem para sua eficiência; podemos citar sua estrutura involucional, seu escalonamento de chaves cíclico, uma S-box quase invariante por rotação e uma camada de difusão implementável utilizando apenas portas lógicas XOR, sem aritmética em GF(2^4). Com isso, FUTURE é uma cifra mais leve que propostas compatíveis como PRESENT (BOGDANOV et al., 2007), afirmação substanciada pelos resultados de implementações de ambas as cifras em bitslicing em plataformas Intel e Cell.

7 CONCLUSÃO

Este trabalho partiu de uma base reconhecidamente sólida no projeto de cifras de bloco, a estratégia de trilha larga (RIJMEN, 1997; DAEMEN, 1995; DAEMEN; RIJMEN, 2001a; DAEMEN; RIJMEN, 2001b), e buscou maneiras de projetar cifras baseadas nesta estratégia que apresentem maior eficiência. Nesse sentido, damos destaque à descoberta das S-boxes invariantes por rotação (Capítulo 4), que permite a obtenção de S-boxes com boas propriedades contra criptanálise, e ao mesmo tempo permitindo implementação eficiente em hardware, e em software caso a técnica de bitslicing (BIHAM, 1997) seja usada. Uma vez que S-boxes mostravam-se o principal obstáculo à implementação em bitslicing de cifras baseadas na estratégia de trilha larga, a contribuição do trabalho é especialmente importante para o avanço deste tópico de pesquisa, considerado como uma das abordagens mais promissoras para obtenção de melhor desempenho nas cifras projetadas de acordo com a estratégia de trilha larga.

Destaca-se também o projeto de uma nova função de hash, MAELSTROM-0, e uma cifra de bloco de nível legado, FUTURE. MAELSTROM-0 apresenta-se como sucessor de WHIRLPOOL (BARRETO; RIJMEN, 2000c), obtendo desempenho 2–3 vezes superior que seu antecessor, significativamente superior a outras funções de porte semelhante (como a família SHA-2 (NIST, 2002)) e comparável a funções mais leves como MD5 e SHA-1, ao mesmo tempo que se mostra mais segura que estas últimas, que já se encontram en-

fraquecidas por ataques como os de (WANG; YU, 2005; WANG; YIN; YU, 2005). Já FUTURE emprega técnicas inovadoras, incluindo uma S-box quase invariante por rotação, para obter alto desempenho independente de plataforma, superando até mesmo propostas otimizadas para plataformas específicas como PRESENT (BOGDANOV et al., 2007).

Ao longo do desenvolvimento deste trabalho, foram vislumbradas outras linhas de pesquisa relacionadas que, infelizmente, não puderam ser contempladas neste trabalho. Estes tópicos ficam como sugestão para trabalhos futuros.

- Foram consideradas S-boxes invariantes por rotação de tamanho 4×4 apenas. Seria desejável aplicar a técnica para construção de S-boxes 8×8, de aplicação mais ampla que as S-boxes 4×4. Também é possível aplicar S-boxes 4×4 invariantes por rotação como blocos de construção de S-boxes maiores, no estilo da S-box de WHIRLPOOL e MAELSTROM-0. Um resultado particularmente favorável seria a obtenção de S-boxes 8×8, cuja implementação em hardware ou bitslicing exija uma quantidade menor de portas lógicas que a S-box de WHIRLPOOL e MAELSTROM-0.

- De posse de S-boxes 8×8 com implementação eficiente em hardware e bitslicing, cria-se a oportunidade para o projeto de uma nova cifra contemplando as mesmas aplicações que o AES (tamanho de bloco de 128 bits e chaves de 128–256 bits), porém utilizando estas novas S-boxes, e possivelmente incorporando outras construções mais eficientes em hardware e bitslicing, a exemplo da matriz de difusão de FUTURE.

- Considerando o potencial das implementações em bitslicing, é interessante pesquisar como aplicar esta técnica em funções de hash, por exemplo pelo projeto de esquemas de encadeamento que permitam o

processamento de blocos distintos da mensagem em paralelo.

REFERÊNCIAS

ANDERSON, R.; BIHAM, E.; KNUDSEN, L. *Serpent: A Proposal for the Advanced Encryption Standard*. 1998. http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf.

BARKAN, E.; BIHAM, E. In how many ways can you write Rijndael? In: *Advances in Cryptology – Asiacrypt 2002*. Queenstown: Springer-Verlag, 2002. (Lecture Notes in Computer Science, v. 2501), p. 160–175.

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

____. The Khazad legacy-level block cipher. In: *First open NESSIE Workshop*. Leuven: NESSIE Consortium, 2000.

____. The Whirlpool hashing function. In: *First Open NESSIE Workshop*. Leuven: NESSIE Consortium, 2000.

BARRETO, P. S. L. M.; SIMPLÍCIO, J. M. A. CURUPIRA, a block cipher for constrained platforms. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – SBRC'2007*. Belém: Sociedade Brasileira de Computação – SBC, 2007.

BERNSTEIN, D. J. *Cache-timing attacks on AES*. 2005. http://cr.yp.to/papers.html#cachetiming.

BIHAM, E. A fast new DES implementation in software. In: *Fast Software Encryption 4*. Haifa: Springer-Verlag, 1997. (Lecture Notes in Computer Science, v. 1267), p. 260–271.

BIHAM, E.; SHAMIR, A. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, v. 4, n. 1, p. 3–72, 1991.

____. Differential cryptanalysis of the full 16-round DES. In: *Advances in Cryptology – Proceedings of CRYPTO '92*. Santa Barbara: Springer-Verlag, 1992. (Lecture Notes in Computer Science, v. 740), p. 487–496.

BOGDANOV, A. et al. PRESENT: An ultra-lightweight block cipher. In: *Cryptographic Hardware and Embedded Systems Workshop – CHES'2007*. Vienna: Springer-Verlag, 2007. (Lecture Notes in Computer Science).

CHEN, T. et al. Cell Broadband Engine and its first implementation – a performance view. *IBM Journal of Research and Development*, v. 51, n. 5, p. 559–572, 2007.

COPPERSMITH, D. The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, v. 38, n. 3, p. 243–250, 1994.

DAEMEN, J. *Cipher and hash function design strategies based on linear and differential cryptanalysis*. Tese (Doutorado) — Katholieke Universiteit Leuven, Março 1995.

DAEMEN, J.; KNUDSEN, L. R.; RIJMEN, V. The block cipher Square. In: *Fast Software Encryption 4*. Haifa: Springer-Verlag, 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: NESSIE Consortium, 2000.

DAEMEN, J.; RIJMEN, V. *The Design of Rijndael: AES – The Advanced Encryption Standard*. Berlin: Springer-Verlag, 2001.

____. The wide trail design strategy. In: *Proceedings of the 8th IMA Conference on Cryptography and Coding*. Cirencester: The Institute of Mathematics and its Applications – IMA, 2001. p. 222–238.

DAI, W. *Crypto++ 5.5 Benchmarks*. 2007. http://www.cryptopp.com/benchmarks.html.

DINIZ, P. S. R.; SILVA, E. A. B.; NETTO, S. L. *Digital Signal Processing: System Analysis and Design*. Cambridge: Cambridge University Press, 2002.

FULLER, J.; MILLAN, W. Linear redundancy in S-boxes. In: *Fast Software Encryption 2003*. Lund: Springer-Verlag, 2003. (Lecture Notes in Computer Science, v. 2887), p. 74–86.

GAURAVARAM, P. et al. Constructing secure hash functions by enhancing Merkle-Damgård construction. In: *Australasian Conference on Information Security and Privacy – ACISP'2006*. Melbourne: Springer-Verlag, 2006. (Lecture Notes in Computer Science, v. 4058).

GAZZONI FILHO, D. L.; BARRETO, P. S. L. M.; RIJMEN, V. The Maelstrom-0 hash function. In: *VI Simpósio Brasileiro em Segurança da Informação e Sistemas Computacionais*. Santos: Sociedade Brasileira de Computação – SBC, 2006.

GILBERT, H.; MINIER, M. A collision attack on 7 rounds of Rijndael. In: *Proceedings of the 3rd AES candidate conference*. New York: NIST, 2000. p. 230–241.

GLADMAN, B. *Serpent S Boxes*. 1998. http://fp.gladman.plus.com/cryptography_technology/serpent/index.htm.

HARDY, G. H.; WRIGHT, E. M. *An Introduction to the Theory of Numbers*. Oxford: Oxford University Press, 1979.

KAHLE, J. A. et al. Introduction to the Cell multiprocessor. *IBM Journal of Research and Development*, v. 49, n. 4/5, p. 589–604, 2005.

KWAN, M. *Reducing the Gate Count of Bitslice DES*. 2000. Cryptology ePrint Archive, Report 2000/051. http://eprint.iacr.org/.

LANG, S. *Linear Algebra*. New York: Springer, 1987.

MATSUI, M. The first experimental cryptanalysis of the Data Encryption Standard. In: *Advances in Cryptology – Proceedings of CRYPTO '94*. Santa Barbara: Springer-Verlag, 1994. (Lecture Notes in Computer Science, v. 839), p. 1–11.

____. Linear cryptanalysis method for DES cipher. In: *Advances in Cryptology – Proceedings of Eurocrypt '93*. Lofthus: Springer-Verlag, 1994. (Lecture Notes in Computer Science, v. 765), p. 386–397.

MENEZES, A. J.; OORSCHOT, P. C. van; VANSTONE, S. A. *Handbook of Applied Cryptography*. Boca Raton: CRC Press, 1997.

NAKAJIMA, J.; MATSUI, M. Performance analysis and parallel implementation of dedicated hash functions. In: *Advances in Cryptology – Eurocrypt'2002*. Amsterdam: Springer, 2002. (Lecture Notes in Computer Science, v. 2332), p. 165–180.

NIST. *Federal Information Processing Standard (FIPS 46-3) – Data Encryption Standard (DES)*. Gaithersburg, Outubro 1999.

____. *Federal Information Processing Standard (FIPS 197) – Advanced Encryption Standard (AES)*. Gaithersburg, Novembro 2001.

____. *Federal Information Processing Standard (FIPS 180-2) – Secure Hash Standard (SHS)*. Gaithersburg, Agosto 2002.

RIJMEN, V. *Cryptanalysis and design of iterated block ciphers*. Tese (Doutorado) — Katholieke Universiteit Leuven, Outubro 1997.

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

VAUDENAY, S. Hidden collisions on DSS. In: *Advances in Cryptology – Proceedings of CRYPTO'96*. Santa Barbara: Springer-Verlag, 1996. (Lecture Notes in Computer Science, v. 1109), p. 83–88.

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

WANG, X.; YIN, Y. L.; YU, H. Finding collisions in the full SHA-1. In: *Advances in Cryptology – Proceedings of CRYPTO'2005*. Santa Barbara: Springer-Verlag, 2005. (Lecture Notes in Computer Science, v. 3621), p. 17–36.

WANG, X.; YU, H. How to break MD5 and other hash functions. In: *Advances in Cryptology – Proceedings of Eurocrypt'2005*. Aarhus: Springer-Verlag, 2005. (Lecture Notes in Computer Science, v. 3494), p. 19–35.

WOLKERSTORFER, J.; OSWALD, E.; LAMBERGER, M. An ASIC implementation of the AES S-boxes. In: *Topics in Cryptology - CT-RSA 2002*. San Jose: Springer, 2002. (Lecture Notes in Computer Science, v. 2271), p. 67–78.

YOUSSEF, A. M.; TAVARES, S. E. *On Some Algebraic Structures in the AES Round Function*. 2002. Cryptology ePrint Archive, Report 2002/144. http://eprint.iacr.org/.

APÊNDICE A – ARTIGOS PRODUZIDOS

Como subprodutos da pesquisa dessa dissertação, os seguintes artigos foram produzidos:

- "Rotation Symmetry in Algebraically Generated Substitution Tables", Vincent Rijmen, Paulo S. L. M. Barreto e Décio Luiz Gazzoni Filho, contendo o material do Capítulo 4. Este artigo foi submetido ao periódico Information Processing Letters, e aceito para publicação;

- "The MAELSTROM-0 Hash Function", Décio Luiz Gazzoni Filho, Paulo S. L. M. Barreto e Vincent Rijmen, contendo o material do Capítulo 5. Este artigo foi submetido ao VI Simpósio Brasileiro em Segurança da Informação e Sistemas Computacionais (SBSeg 2006), aceito e agraciado com o prêmio de melhor artigo do simpósio;

- "FUTURE, a lightweight hardware-oriented block cipher", Décio Luiz Gazzoni Filho, Paulo S. L. M. Barreto e Paris Kitsos, contendo o material do Capítulo 6. Este artigo ainda encontra-se em processo de escrita.