Capítulo 3

Canais laterais em criptografia simétrica e de curvas elípticas: ataques e contramedidas

Lucas Z. Ladeira, Erick N. Nascimento, João Paulo F. Ventura,
Ricardo Dahab, Diego F. Aranha, Julio C. López Hernández

Resumo

Para prover segurança em um ambiente hostil, algoritmos criptográficos precisam resistir a uma infinidade de ataques buscando a obtenção de informação sigilosa, acesso não-autorizado, entre outros. Tais ataques ocorrem tanto nos algoritmos e seus problemas computacionais subjacentes, quanto nas implementações de um criptossistema. Uma classe de ataques sobre implementações de algoritmos são os chamados ataques de canal lateral, que fazem uso de informações vazadas durante a execução de uma primitiva criptográfica. Ataques dessa natureza utilizam variações no tempo de execução, no consumo de energia, emanações eletromagnéticas e outras características do dispositivo alvo. Contramedidas para esses ataques podem ser baseadas em modificações no software ou no hardware. Neste minicurso, discutem-se ataques e contramedidas em implementações em software de métodos criptográficos simétricos, e assimétricos baseados em curvas elípticas.

Abstract

In order to provide security in a hostile environment, cryptographic algorithms must resist many attacks aiming to capture confidential information, obtain non-authorized access, among others. These attacks can target either the algorithms and their underlying hardness assumptions, or the implementations of a cryptosystem. Side-channel attacks are a class of attacks directed at the implementations of cryptographic algorithms and make use of information leaked during the execution of a cryptographic primitive. Such attacks are based on variations in execution time, energy consumption, electromagnetic emanations and other features of the device. Countermeasures are based on software or hardware mechanisms, or both. In this short course, we discuss side-channel attacks and countermeasures against software implementations of symmetric primitives and curve-based public-key cryptography.

3.1. Introdução

A Internet das Coisas (IoT, do inglês *Internet of Things*) permite a interconexão de múltiplos dispositivos embarcados que manipulam dados de diferentes tipos e realizam tarefas diversas, algumas até críticas. Este ambiente promete trazer enormes benefícios para a vida em sociedade e, naturalmente, induz novos requisitos para sua implementação eficaz e robusta. Entre estes requisitos, segurança, tolerância a falhas e privacidade surgem como dimensões novas e fundamentais no projeto de sistemas embarcados.

Apesar da importância de se equipar a Internet das Coisas com mecanismos robustos de segurança, a mudança de paradigma trazida por essa nova tecnologia cria um problema desafiador para o projeto de mecanismos de segurança: enquanto os dispositivos precisam permanecer compactos e baratos, a quantidade massiva de dados coletados e transportados por esses dispositivos e sua natureza sensível certamente terão implicações significativas em privacidade. Simultaneamente, qualquer solução prática precisa levar em conta os recursos reduzidos e a proteção física limitada que são típicas de dispositivos da Internet das Coisas.

Este minicurso discute técnicas para implementação segura de algoritmos criptográficos, com aplicações para a proteção de dispositivos embarcados operando na Internet das Coisas. Pretende-se discutir duas classes de algoritmos, criptografia simétrica baseada em cifras de bloco e funções de resumo criptográfico e criptografia assimétrica baseada em curvas elípticas, abrangendo seu projeto e implementação eficiente e segura contra ataques de canal lateral. A motivação para o estudo de primitivas simétricas é o menor consumo de recursos que seus correspondentes assimétricos, resultando em maior eficiência e tempo de vida quando sua utilização é maximizada em protocolos de comunicação para a Internet das Coisas. Por outro lado, esquemas para criptografia assimétrica são essenciais para estabelecer chaves criptográficas para primitivas simétricas e, portanto, precisam ser parte integral para qualquer arquitetura aplicada de segurança. Nesse cenário, o estudo de esquemas assimétricos baseados em curvas elípticas são de interesse especial, devido ao seu potencial para implementação em dispositivos com recursos restritos.

Sobre o conteúdo este documento

Este minicurso é uma versão atualizada e aprofundada de um minicurso apresentado no SBSeg 2009 por dois dos autores deste texto. Maior ênfase é dada a experimentos reais e contramedidas aos ataques de canais laterais, bem como aos métodos criptográficos simétricos. Quanto aos assimétricos, a maior parte é devotada aos métodos baseados em curvas elípticas sobre corpos primos, e tão somente à aritmética de pontos da curva elíptica, sem nos atermos à aritmética de corpos finitos.

Um bom número de termos técnicos foram mantidos em inglês, em razão de não termos um vocabulário estável da área em português, ainda que vários termos tenham já boas traduções. Procuramos manter os termos em inglês em fonte itálico.

Agradecimentos

Os autores gostariam de agradecer: ao comitê de programa dos minicursos do SBSeg 2016 pela oportunidade de apresentarmos este trabalho; à Fapesp e Intel pelo suporte financeiro ao projeto "Secure execution of cryptographic algorithms"; à Intel pelo generoso apoio ao projeto de pesquisa "Software Implementation of Cryptographic Algorithms"; ao CNPq/Intel pelas bolsas concedidas no contexto deste último projeto.

3.2. Criptografia simétrica

Algoritmos criptográficos podem ser classificados em *simétricos* e *assimétricos*. Os algoritmos simétricos baseiam-se na existência de um segredo pré-compartilhado, chamado de *chave secreta*. Em um contexto de sigilo, chaves de encriptação e decriptação são idênticas ou podem ser calculadas eficientemente uma a partir da outra. O tarefa (difícil) de um adversário nesses esquemas normalmente resume-se a explorar o espaço de chaves, de tamanho gigantesco, em um ataque chamado de *busca exaustiva*. O algoritmo AES (*Advanced Encryption Standard*) [AES 2001] é talvez o mais difundido entre os algoritmos simétricos de encriptação em uso, enquanto SHA [SHA 2012] é uma família padronizada de funções de resumo criptográfico. Algoritmos assimétricos empregam um par de chaves, *pública* e *privada*, que são calculadas de forma a existir uma relação especial entre ambas: calcular a chave pública a partir da chave privada é eficiente, mas não se conhece algoritmo eficiente para resolver o problema no sentido inverso. Assim, a chave pública pode ser utilizada para encriptação ou verificação de assinaturas, sem ameaça à segurança do algoritmo criptográfico, e a chave privada para decriptação ou assinatura digital deve ser mantida em sigilo, sob posse exclusiva do seu detentor. Criptografia de curvas elípticas (ECC – *Elliptic Curve Cryptography*) é um exemplo de família de algoritmos assimétricos.

3.2.1. Encriptação

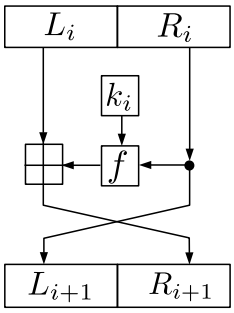
O requisito de segurança de *confidencialidade* pode ser provido por um par de funções de encriptação E e decriptação D, ambas parametrizadas por uma chave k. A encriptação de uma mensagem M sob a chave k produz um criptograma $C = E_k(M)$; assim, a mensagem original pode ser recuperada pela função de decriptação como $M = D_k(C)$, de forma que a propriedade de consistência seja mantida, isto é, $M = D_k(E_k(M))$ para quaisquer valores de M e k.

As funções E e D podem ser implementadas por um algoritmo simétrico de encriptação de duas formas distintas. *Cifras de bloco*, como o AES, especificam como encriptar uma sequência de *bits* com tamanho fixo, e precisam ser estendidos para mensagens de tamanho arbitrário por meio de um *modo de operação*. *Cifras de fluxo*, como ChaCha [Bernstein 2005], calculam uma cadeia de chaves a partir da chave k, que é então combinada com a mensagem original utilizando operações `XOR` (OU exclusivo). É desejável que criptogramas transmitidos em canais de comunicação sejam acompanhados de *autenticadores*, que permitem ao destinatário certificar-se de que o remetente é a origem legítima da mensagem e verificar sua integridade.

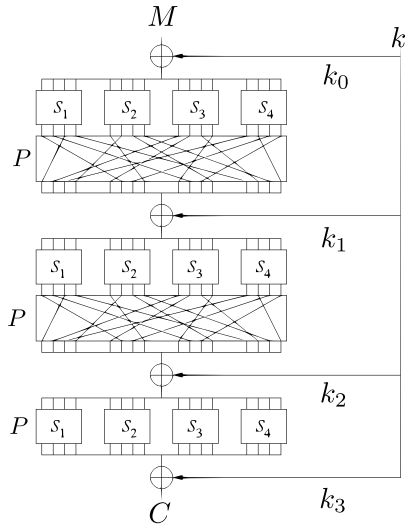
Há muitos paradigmas diferentes para se construir cifras simétricas. Em geral,

os algoritmos seguem a idéia proposta por Shannon [Shannon 1949] de um *produto iterado de cifras*, em que uma *função de rodada* composta por cifras menores e parametrizada por *chaves de rodada* é repetida um número fixo r de vezes. A função de rodada combina pequenas *cifras de substituição*, que substituem símbolos para confundir a relação entre o criptograma e a chave criptográfica, com uma *cifra de transposição*, que altera a ordem dos *bits* para espalhar a redundância do texto claro ao longo do criptograma [Shannon 1949]. O cálculo de chaves de rodada k_i a partir da chave k é também chamado de *escalonamento de chaves*.

Os paradigmas clássicos mais comuns para construção de cifras de bloco são *Redes de Feistel* e *Redes de Substituição-Permutação* (SPN – *Substitution-Permutation Network*). Redes de Feistel (ou cifras de Luby-Rackoff [Luby and Rackoff 1988]) utilizam uma função de rodada não-inversível f que atua sobre o bloco de texto claro $(L_0 \parallel R_0)$. A cada rodada, a função f calcula a próxima metade do estado interno pela regra $R_{i+1} = L_i \oplus f(R_i, k_i)$, enquanto a metade restante é uma simples cópia $L_{i+1} = R_i$, até que seja atingido o bloco de criptograma $(L_r \parallel R_r)$. A decriptação segue processo análogo, utilizando as regras inversas $R_i = L_{i+1}$ e $L_{i+1} = R_{i+1} \oplus f(L_{i+1}, k_i)$. SPNs alternam *caixas de substituição* S_i que operam sobre pedaços de ℓ *bits* do texto claro (que implementam cifras de substituição) com uma *camada de difusão linear* P que opera sobre todo o estado interno. Ao final de cada rodada, a chave de rodada k_i é adicionada ao estado interno por uma operação `XOR`. A decriptação é realizada pela aplicação de operações inversas também na ordem inversa. As Figuras 3.1a e 3.1b apresentam diagramas para Redes de Feistel e de Permutação-Substituição, respectivamente.



(a) Rodada de uma Rede de Feistel.



(b) Rede de Permutação-Substituição.

Figura 3.1: Construções para cifras de bloco.

Cifras de fluxo, por sua vez, possuem construções derivadas de geradores pseudo-aleatórios, onde a chave criptográfica faz o papel da semente, a qual é expandida numa sequência pseudo-aleatória de bits, formando uma chave muito mais longa. Estas devem ser construídas de modo a maximizar o período do gerador, para evitar repetições na chave expandida. Pode-se dizer que cifras de fluxo são um relaxamento do *one-time pad* (OTP),

onde cada símbolo do texto claro é combinado com um símbolo aleatório da chave, que deve possuir o mesmo tamanho da mensagem a ser encriptada. A chave pseudo-aleatória de uma cifra de fluxo abre mão da *segurança incondicional* do OTP [Shannon 1949], em favor de uma premissa de segurança computacional com ganho de eficiência.

Modos de operação

Modos de operação estendem o requisito de confidencialidade para mensagens de tamanho arbitrário, ou até mesmo requisitos de segurança adicionais, como *encriptação autenticada*, em um único passo. Estes modos dividem a mensagem M em n blocos de mesmo tamanho $\{M_1, \ldots, M_n\}$ e utilizam uma regra para produzir os blocos de criptograma $C = \{C_1, \ldots, C_n\}$. O último bloco precisa ser completado com *preenchimento* (ou *padding*) para poder ser processado corretamente. Diversos modos de operação já foram padronizados por agências de padronização como o NIST para utilização governamental ou na indústria, dentre eles CBC, CTR e GCM.

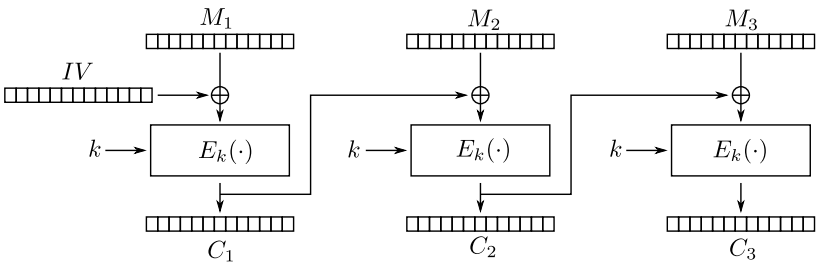
No modo de operação CBC (*Cipher Block Chaining*), o próximo bloco de criptograma é calculado a partir de um bloco de texto claro e o bloco de criptograma anterior, pela regra $C_i = E_k(M_i \oplus C_{i-1})$, para $2 \leq i \leq n$. O bloco C_0 é definido como um *vetor de inicialização IV* único e imprevisível, que não deve se repetir em encriptações distintas. Observe que o encadeamento da encriptação aleatoriza o bloco de texto claro antes da próxima encriptação, fazendo com que blocos idênticos produzam blocos distintos no criptograma. Como apenas a decriptação de blocos distintos pode ser feita de modo independente, paralelismo pode ser extraído apenas no processo de decriptação.

O modo de operação CTR (*Counter* simula uma cifra de fluxo a partir de uma cifra de bloco. Os blocos do criptograma são calculados pela operação `XOR` entre blocos de texto claro e encriptação de valores consecutivos do vetor de inicialização IV. Portanto, a regra é dada por $C_i = M_i \oplus E_k(IV + i)$, para $1 \leq i \leq n$. Como o processamento de cada bloco é independente, este modo de operação oferece paralelismo tanto na encriptação quanto decriptação, sendo tipicamente mais eficiente que o modo CBC. A Figura 3.2 ilustra os dois modos de operação.

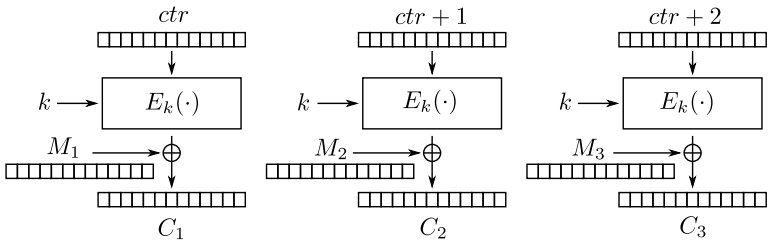
O modo de encriptação autenticada *GCM* (*Galois/Counter Mode*) [McGrew and Viega 2004] utiliza o modo CTR para encriptação e atualiza um autenticador calculado a partir dos blocos de criptograma C_i que permite detetar posteriormente qualquer manipulação do criptograma em trânsito, com alta probabilidade. Opcionalmente, o modo também autentica dados associados transmitidos às claras, funcionalidade útil para autenticar cabeçalhos de pacotes de rede ou outra informação pública cuja integridade deva ser preservada e possa ser verificada pelo destinatário.

3.2.2. Funções de *hash* ou resumo criptográfico

O objetivo de funções de *hash* ou resumo criptográfico é mapear uma cadeia de *bits* de tamanho arbitrário em um *resumo* com tamanho fixo em *bits*. Este resumo tem como papel identificar unicamente a mensagem de entrada, servindo como uma espécie de "impressão digital" da mesma. São muitas as aplicações em Criptografia e Segurança Computacional de funções criptográficas: armazenamento seguro de senhas, esquemas de assinatura di-



(a) Encriptação no modo CBC.



(b) Encriptação no modo CTR.

Figura 3.2: Modos de operação para cifras de bloco.

gital, verificação de integridade, derivação de chaves criptográficas, geração de números pseudo-aleatórios, projeto de cifras de fluxo e autenticadores, provas de trabalho de moedas criptográficas, entre outras. A Figura 3.3 apresenta essa funcionalidade abstrata de funções de resumo criptográfico.

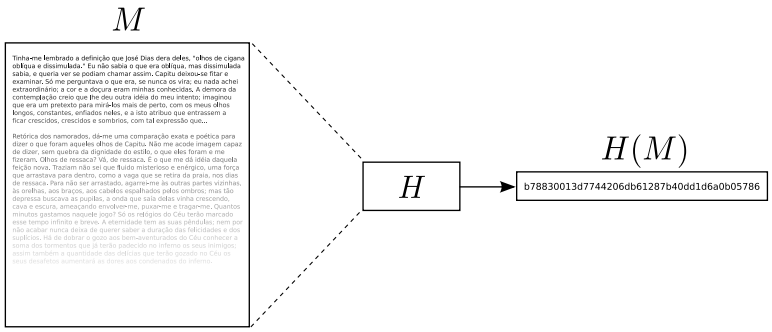


Figura 3.3: Função de resumo criptográfico mapeando M para o valor de resumo $H(M)$.

Em termos matemáticos, funções de resumo são da forma $H : \{0,1\}^* \to \{0,1\}^n$. Ou seja, mapeiam uma pré-imagem x de tamanho finito e arbitrário em um *valor de hash* $y = H(x)$ de tamanho fixo. A princípio, tratam-se de funções não-chaveadas, ou seja, não-parametrizadas por uma chave criptográfica, mas é claro que podem ser utilizadas também para calcular resumos de informação secreta. Para haver interesse criptográfico,

funções de resumo precisam satisfazer as três propriedades abaixo:

- **Resistência à pré-imagem**: Dada uma função de resumo criptográfico H e um resumo y, deve ser computacionalmente inviável encontrar x tal que $y = H(x)$. Em outras palavras, deve ser difícil "inverter" a função para um certo valor de resumo.
- **Resistência à segunda pré-imagem**: Dado um resumo y e uma mensagem x tais que $y = H(x)$, deve ser computacionalmente inviável encontrar uma mensagem $x' \neq x$ tal que $H(x') = H(x) = y$. Ou seja, deve ser difícil encontrar uma segunda mensagem mapeada para o mesmo valor de resumo.
- **Resistência a colisões**: Deve ser computacionalmente inviável encontrar mensagens x, x' tais que $H(x) = H(x')$; ou seja, deve ser difícil encontrar duas mensagens que colidem sob H para um mesmo valor de resumo.

Há muitas formas diferentes de se construir essas funções. Observe que, pelo tamanho e natureza dos conjuntos de domínio e imagem, *colisões* entre diferentes entradas sob uma função h necessariamente irão existir. O desafio é projetar uma função que torne computacionalmente inviável encontrar essas colisões, mesmo que seu número seja infinito. Classicamente, o paradigma Merkle-Damgård constrói uma função de resumo criptográfico resistente a colisão a partir de uma função de compressão h com entrada de tamanho fixo, também resistente a colisão [Merkle 1979, Damgård 1989]. Na Figura 3.4, a entrada X é dividida entre B blocos, com aplicação de preenchimento para formar o último bloco, e a saída da última função de compressão define o valor de resumo. Funções de resumo podem também ser construídas a partir de cifras de bloco [Preneel et al. 1993], ou problemas difíceis de Teoria dos Números, como a fatoração de inteiros.

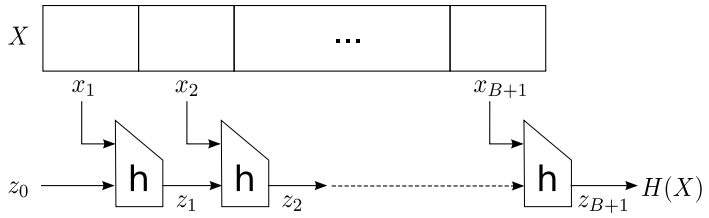


Figura 3.4: Construção Merkle-Damgård para funções de resumo criptográfico.

Um paradigma de construção que se tornou imensamente popular é a classe de *esponjas criptográficas*. Uma esponja é uma função com estado interno s de tamanho finito b, dividido em duas seções: a taxa de *bits* de tamanho r e a capacidade de tamanho c, tais que $b = c + r$. Completam a especificação uma função de permutação f de tamanho fixo, que transforma o estado interno, e uma regra de preenchimento. Durante a inicialização, uma função esponja atribui o valor 0 ao estado interno e completa a mensagem de entrada para que seu tamanho seja múltiplo de r, de forma que a entrada possa ser dividida em blocos de tamanho r. A cada iteração da função, um novo bloco da entrada é adicionado a parte do estado interno pela operação `XOR` e o estado interno s é substituído por $f(s)$. Em outras palavras, os blocos da entrada são absorvidos sucessivamente no estado interno. Na etapa final, toma-se r *bits* do estado interno por iteração até que a saída atinja

um tamanho pré-determinado, alternando com novas aplicações da função de permutação $f(s)$. A resistência a colisões ou ataques de pré-imagem depende do tamanho c, tipicamente escolhido como duas vezes o nível de segurança desejado. A Figura 3.5 apresenta a construção [Bertoni et al. 2008].

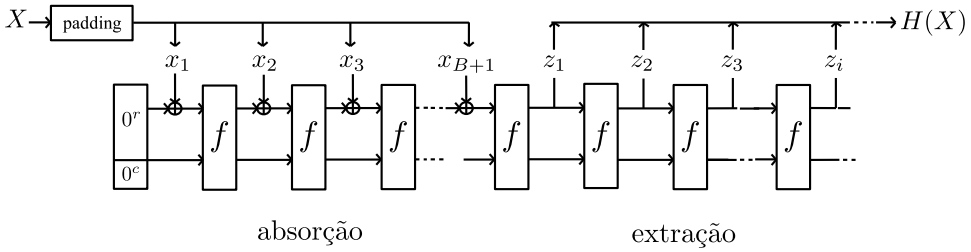
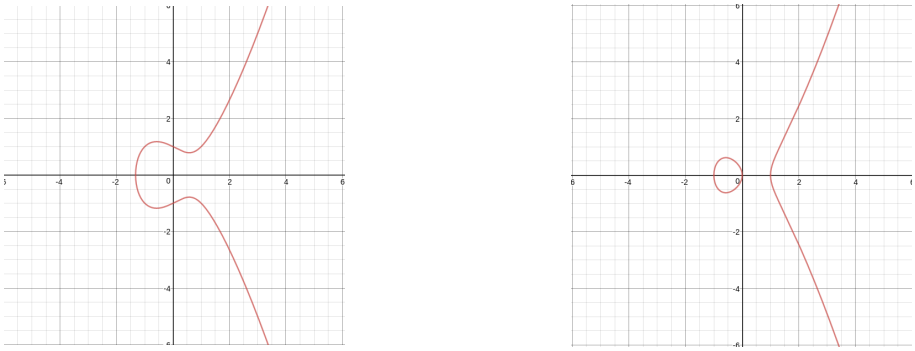


Figura 3.5: Esponja criptográfica, e suas fases de absorção e extração.

3.3. Criptografia de curvas elípticas (ECC)

Em 1985, Neal Koblitz [Koblitz 1987] e Victor Miller [Miller 1986], de forma independente, propuseram a utilização de curvas elípticas para projetar criptossistemas de chave-pública.

Curvas elípticas são definidas por equações cúbicas, como por exemplo $y^2 = x^3 - x$, onde os valores x, y pertencem a uma estrutura algébrica chamada corpo, dos quais os números racionais, reais e complexos são exemplos. A Figura 3.6 ilustra duas curvas elipticas sobre o conjunto $\mathbb{R}$ dos reais. Curvas elípticas não são importantes apenas para a Criptografia, mas também em outros ramos da Matemática como, por exemplo, na prova do Último Teorema de Fermat. [Hankerson et al. 2003]. No caso da Criptografia



(a) $y^2 = x^3 - x + 1$ (b) $y^2 = x^3 - x$

Figura 3.6: Exemplo de duas curvas elípticas

de Curvas Elípticas (ECC), o corpo sobre o qual curvas são definidas são os chamados corpos finitos. Tais corpos são conjuntos finitos de tamanho p^m, denotados $\mathbb{F}_{p^m}$, onde p um número primo e m um inteiro maior ou igual a 1, e munidos de duas operações $+, \times$. Os elementos de $\mathbb{F}_{p^m}$ são os polinômios de grau máximo $m-1$ e coeficientes em $Z_p = \{0, 1, 2, \ldots, p-1\}$. As operações $+, \times$ são as operações usuais de adição e

multiplicação de polinômios, com duas condições extras: (i) os coeficientes do polinômio $r(x)$ resultante de uma operação são reduzidos módulo p; e (ii) $r(x)$ é tomado módulo um polinômio irredutível $f(x)$ em Z_p.

Quando $m = 1$, dizemos $\mathbb{F}_p$ é um corpo primo e seus elementos são simplesmente os inteiros em $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$ com aritmética módulo p. A seguir apresentaremos equações de curvas elípticas para corpos primos $\mathbb{Z}_p$ e *corpos binários* $\mathbb{F}_{2^m}$.

Curvas sobre $\mathbb{F}_p$ e $\mathbb{F}_{2^m}$

Uma *curva elíptica* E sobre $\mathbb{F}_p$ é definida por uma equação da forma

$$y^2 = x^3 + ax + b, \tag{1}$$

onde os coeficientes $a, b \in \mathbb{F}_p$ satisfazem $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. Para corpos binários $\mathbb{F}_{2^m}$, uma curva elíptica (não-supersingular) é definida por uma equação da forma

$$y^2 + xy = x^3 + ax^2 + b, \tag{2}$$

onde os coeficientes $a, b \in \mathbb{F}_{2^m}$ e $b \neq 0$. Dizemos que um par (x, y), onde $x, y \in \mathbb{F}_q$ ($\mathbb{F}_p$ ou $\mathbb{F}_{2^m}$), é um *ponto* na curva se (x, y) satisfaz a equação 1 (ou 2). O conjunto de pontos da curva junto com um ponto especial ∞, chamado *ponto no infinito*, é denotado por $E(\mathbb{F}_q)$. Por exemplo, se E é a curva elíptica sobre $\mathbb{Z}_{29}$, definida pela equação

$$y^2 = x^3 + 4x + 20, \tag{3}$$

então o conjunto de 37 pontos de $E(\mathbb{F}_{29})$ é

$$\begin{array}{lllllll}
\{\infty, & (2,6), & (4,19), & (8,10), & (13,23), & (16,2), & (19,16), \\
(0,7), & (2,23), & (5,7), & (8,19), & (14,6), & (16,27), & (20,3), \\
(0,22), & (3,1), & (5,22), & (10,4), & (14,23), & (17,10), & (20,26), \\
(1,5), & (3,28), & (6,12), & (10,25), & (15,2), & (17,19), & (24,7), \\
(1,24), & (4,10), & (6,17), & (13,6), & (15,27), & (19,13), & (24,22), \\
(27,2), & (27,27)\}. & & & & &
\end{array}$$

Existe uma operação, conhecida como lei de "secantes e tangentes", que permite "somar" pontos elípticos em E. Com essa operação, o conjunto de pontos de $E(\mathbb{F}_q)$ forma um grupo abeliano cuja identidade é o ponto no infinito ∞. A fórmula para somar pontos requer umas poucas operações aritméticas no corpo finito subjacente. Para corpos primos $\mathbb{F}_p$, as fórmulas explícitas para somar dois pontos são dadas a seguir:

1. *Identidade:* $P + \infty = \infty + P = P$ para todo $P \in E(\mathbb{F}_p)$
2. *Negativos:* Se $P = (x, y) \in E(\mathbb{F}_p)$ então $(x, y) + (x, -y) = \infty$.
   O ponto $(x, -y)$ é denotado por $-P$ e chamado o *negativo* de P.
3. *Soma de pontos:* Seja $P = (x_1, y_1) \in E(\mathbb{F}_p)$ e $Q = (x_2, y_2) \in E(\mathbb{F}_p)$, onde $P \neq \pm Q$. Então $P + Q = (x_3, y_3)$, onde

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad \text{e} \quad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1.$$

4. *Duplicação de ponto:* Seja $P = (x_1, y_1) \in E(\mathbb{F}_p)$, onde $P \neq -P$. Então $2P = P + P = (x_3, y_3)$, onde

$$x_3 = (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1 \text{ e } y_3 = (\frac{3x_1^2 + a}{2y_1})(x_1 - x_3) - y_1.$$

A partir da operação de soma no grupo $E(\mathbb{F}_q)$, define-se o produto de um escalar $k \in \mathbb{Z}$ por um ponto $P \in E(\mathbb{F}_q)$ como o ponto $kP = P + P + \cdots + P$ (com k parcelas) se $k > 0$, e por extensão $0P = \infty$ e $-kP = k(-P) = -(kP)$. Esta operação é chamada de *multiplicação de um ponto por um escalar* e é a operação central dos esquemas criptográficos baseados em curvas elípticas. Quando estiver subentendido o corpo sobre o qual a curva $E(\mathbb{F}_{p^m})$ é definida, escreveremos simplesmente E para denotar a curva.

3.3.1. Curvas NIST e curvas modernas

As curvas NIST foram geradas pela NSA (National Security Agency) dos EUA e recomendadas para utilização pelo governo americano. São dez escolhas para corpos finitos, sendo cinco corpos primos ($\mathbb{F}_p$) e cinco corpos binários ($\mathbb{F}_{2^m}$) [Brown et al. 2001]. Os corpos finitos recomendados são relacionados a seguir, com os respectivos polinômios irredutíveis, onde o prefixo P denota corpos primos e o B denota corpos binários:

- P-192
  $\mathbb{F}_{192}$ $p = 2^{192} - 2^{64} - 1$;
- P-224
  $\mathbb{F}_{224}$ $p = 2^{224} - 2^{96} + 1$;
- P-256
  $\mathbb{F}_{256}$ $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$;
- P-384
  $\mathbb{F}_{384}$ $p = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$;
- P-521
  $\mathbb{F}_{521}$ $p = 2^{521} - 1$;
- B-163
  $\mathbb{F}_{2^{163}}$ $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$;
- B-233
  $\mathbb{F}_{2^{233}}$ $f(x) = x^{233} + x^{74} + 1$;
- B-283
  $\mathbb{F}_{2^{283}}$ $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$;
- B-409
  $\mathbb{F}_{2^{409}}$ $f(x) = x^{409} + x^{87} + 1$;
- B-571
  $\mathbb{F}_{2^{571}}$ $f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$.

Os corpos foram determinados com base no desempenho, facilitando a aritmética utilizada. Há também opções de curvas binárias genéricas (ou de Koblitz) para as curvas binárias. As curvas acima foram geradas de forma pseudo-aleatória, descartando-se curvas que não fossem resistentes aos ataques conhecidos. Nem todas as etapas do processo de geração das curvas são completamente transparentes, especialmente na escolha de sementes do gerador de bits pseudo-aleatórios. Essa lacuna tem produzido resistência na comunidade técnica, por haver a possibilidade de que as sementes tenham sido escolhidas de forma a produzir curvas com vulnerabilidades conhecidas pela NSA mas desconhecidas do público em geral.

Por esse motivo, outros modelos de curvas elípticas têm recebido maior atenção na literatura científica e na adoção pela indústria, por terem métodos de geração mais transparentes e implementações de alto desempenho e protegidas contra ataques de canal

lateral de tempo. Exemplos dessas são as curvas de Montgomery e Edward. Nessas curvas, o número de operações realizadas é *regular* e baseiam-se apenas no comprimento das chaves e não no valor dos *bits* da chave.

A curva de Montgomery é dada pela equação $E : y^2 = x^3 + Ax^2 + x$. O valor de A pode ser alterado de forma a melhorar o desempenho das multiplicações escalares. O corpo primo mais utilizado com essa curva é $\mathbb{F}_{2^{255}-19}$ com o valor de $A = 486662$ [Düll et al. 2015]. Essa combinação possibilita a implementação eficiente da multiplicação por escalar usando o método de *Montgomery Ladder*.

A curva de Edwards é encontrada na literatura na sua forma *twisted*, em razão do desempenho exibido na combinação com os primos de Mersenne $p = 2^{127} - 1$ e 2^{255-10} em [Costello and Longa 2015] e [Bernstein et al. 2012], respectivamente. A equação dessa curva é $-x^2 + y^2 = 1 + dx^2y^2$, onde d é um não-quadrado em $\mathbb{F}_p^2$.

Nessa curva, a soma de dois pontos segue a fórmula de adição de *Edwards*:

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + x_2y_1}{1 + xd_1x_2y_1y_2}, \frac{y_1y_2 + x_1x_2}{1 - dx_1x_2y_1y_2}\right)$$

Um fato interessante é que a duplicação de um ponto tem a mesma fórmula da adição, o que não acontece em geral.

3.3.2. Algoritmos básicos para multiplicação por escalar (ECSM)

Como já dissemos, a principal operação em curvas elípticas é a multiplicação por escalar kP. O escalar k é, de fato, a chave privada, ou parte dela, em vários esquemas criptográficos baseados em ECC. Dessa forma, é necessário explorar diferentes formas de implementar esse cálculo, não só do ponto de vista de desempenho como o de segurança, assunto central deste texto. Nesta seção serão apresentados os métodos mais simples para o cálculo de kP, resistentes ou não a ataques de canal lateral, e pequenas variações.

O primeiro é o método *Double-and-add* (também conhecido como *Double-and-add-not-always*) [Rivain 2011]. O Algoritmo 1 é chamado de *left-to-right, double-and-add* já que os bits de k são processados da esquerda para a direita. Esse algoritmo não é resistente a ataques de canal lateral. Uma simples inspeção do código revela que o comando na linha 5 terá tempos de execução diferentes dependendo do valor do bit k_i.

Algoritmo 1 Left-to-right double-and-add

Entrada: $P \in E(\mathbb{F}_p), k = (k_{t-1}, \ldots, k_1, k_0)_2 \in \mathbb{N}$
Saída: $kP \in E(\mathbb{F}_p)$
1: $N \leftarrow P$
2: $Q \leftarrow 0$
3: **for** i from $t-1$ **downto** 0 **do**
4: $\quad N \leftarrow 2N$
5: $\quad$ **if** $k_i = 1$ **then**
6: $\quad\quad Q \leftarrow Q + N$
7: $\quad$ **end if**
8: **end for**
9: **return** Q

Uma outra maneira de percorrer os *bits* de k é da direita para a esquerda, conhecida como *right-to-left* [Rivain 2011], resultando num algoritmo ligeiramente diferente, o Algoritmo 2. Pelas mesmas razões acima, esse algoritmo não resiste a ataques de canal lateral de tempo.

Algoritmo 2 Right-to-left double-and-add

Entrada: $P \in E(\mathbb{F}_p), k = (k_{t-1}, \ldots, k_1, k_0)_2 \in \mathbb{N}$
Saída: $kP \in E(\mathbb{F}_p)$
1: $N \leftarrow P$
2: $Q \leftarrow 0$
3: **for** i from 0 **to** $t-1$ **do**
4: **if** $k_i = 1$ **then**
5: $Q \leftarrow Q + N$
6: **end if**
7: $N \leftarrow 2N$
8: **end for**
9: **return** Q

Mais adiante trataremos de outros métodos de multiplicação por escalar que são resistentes a ataques de canal lateral, tornando constante o número de operações executadas em cada iteração e o tempo de execução independente do valor do escalar k.

3.3.3. Algoritmos para multiplicação de ponto fixo por escalar

Nos casos em que P é um ponto fixo, isto é, reutilizado para diferentes valores de k, é possível pré-processar parte da multiplicação escalar kP, obtendo um ganho de desempenho [Hankerson et al. 2003].

A ideia é pré-computar os valores $2^wP, 2^{2w}P, 2^{3w}P, \ldots, 2^{w(d-1)}P$, onde onde w é o tamanho da palavra do processador-alvo da implementação, usualmente 32 ou 64 bits, também chamada de janela ou *window* em inglês, e d é o número de tais palavras ocupadas pelo escalar k [Hankerson et al. 2003]; isto é, se $k = (k_{t-1}, \ldots, k_1, k_0)_2$ tem k bits, então $d = \lceil t/w \rceil$, e escrevemos $k = (K_{d-1}, \ldots, K_1, K_0)_{2^w}$ para denotar os blocos de w bits da chave k.

Utilizar janelas é uma prática comum para aumento da eficiência de implementações, já que operandos podem ser descritos e manipulados em blocos de bits do tamanho da janela, para o qual as instruções do processador já são especializadas. O Algoritmo 3, proposto por Brickell at al. utiliza essa técnica, conhecida por *Fixed-base Windowing Method* .

Algoritmo 3 *Fixed-base windowing method* para multiplicação escalar de um ponto

Entrada: Janela w, $P \in E(\mathbb{F}_p)$, $d = \lceil t/w \rceil$, $k = (K_{d-1}, \ldots, K_1, K_0)_{2^w} \in \mathbb{N}$
Saída: $kP \in E(\mathbb{F}_p)$

1: Compute $P_i = 2^i P, 0 \leq i \leq d-1$ (*pré-computação*)
2: $A \leftarrow \infty$, $B \leftarrow \infty$
3: **for** j from $2^w - 1$ **downto** 1 **do**
4: **for each** i for which $K_i = j$ **do**
5: $B \leftarrow B + P_i$
6: $A \leftarrow A + B$
7: **end for**
8: **end for**
9: **return** A

3.3.4. Algoritmos regulares para multiplicação por escalar

Para que algoritmos de multiplicação por escalar sejam resistentes a ataques de canal lateral é necessário que o fluxo de execução seja regular, isto é, que não haja variações de tempo em função do valor do escalar k, a chave privada.

O primeiro exemplo dessa regularidade é ilustrada no Algoritmo 4, chamado *Double-and-add-always* e proposto por Coron [Coron 1999], que executa uma adição e uma duplicação elípticas em todas a iterações. Uma implementação mais uniforme é a chamada *Atomic Double-and-Add* [Batina et al. 2014], que executa somente operações de soma de pontos elípticos.

Algoritmo 4 Double-and-add-always

Entrada: $P \in E(\mathbb{F}_p)$, $k = (k_{t-1}, \ldots, k_1, k_0)_2 \in \mathbb{N}$
Saída: $kP \in E(\mathbb{F}_p)$

1: $R_0 \leftarrow P$
2: **for** i from $t-2$ **downto** 0 **do**
3: $R_0 \leftarrow 2R_0$
4: $R_1 \leftarrow R_0 + P$
5: $b \leftarrow k_i$
6: $R_0 \leftarrow R_b$
7: **end for**
8: **return** R_0

Algoritmo 5 Atomic double-and-add

Entrada: $P \in E(\mathbb{F}_p)$, $k = (k_{t-1}, \ldots, k_1, k_0)_2 \in \mathbb{N}$
Saída: $kP \in E(\mathbb{F}_p)$
1: $R_0 \leftarrow P$
2: $R_1 \leftarrow P$
3: $j \leftarrow t-2$
4: $b \leftarrow 0$
5: **while** $j \geq 0$ **do**
6: $\quad R_0 \leftarrow R_0 + R_b$
7: $\quad b \leftarrow b \oplus k_j$
8: $\quad j \leftarrow j + k_j - 1$
9: **end while**
10: **return** R_0

Um outro algoritmo importante é o chamado *Montgomery Ladder* [Düll et al. 2015], que é mais eficiente que outros similares, e tem a vantagem adicional de poder ser otimizado para diferentes curvas. Primeiramente exibimos uma implementação genérica do método, no Algoritmo 6.

Algoritmo 6 *Montgomery Ladder*

Entrada: $P \in E(\mathbb{F}_p), k = (k_{t-1}, \ldots, k_1, k_0)_2 \in \mathbb{N}$
Saída: $kP \in E(\mathbb{F}_p)$
1: $R_0 \leftarrow P$
2: $R_1 \leftarrow 2P$
3: **for** $j \leftarrow t-2$ **to** 0 **do**
4: $\quad b \leftarrow k_j$
5: $\quad R_{1-b} \leftarrow R_0 + R_1$
6: $\quad R_b \leftarrow 2R_b$
7: **end for**
8: **return** R_0

Quando se usa o *Montgomery Ladder* com a curva de Montgomery com primo $p = 2^{255} - 19$, há um ganho de eficiência. Para o cálculo de kP, as entradas para o algoritmo são o escalar k com 256 bits, e a coordenada x_P do ponto P. A função *cswap*, ou *conditional swap*, faz a troca dos valores dos dois primeiros parâmetros caso o valor do terceiro parâmetro seja 1.

Algoritmo 7 *Montgomery ladder* particularizado para a curva de Montgomery

Entrada: Coordenada x_P de um ponto $P \in E(\mathbb{F}_p), k = (k_{255}, \ldots, k_1, k_0)_2 \in \mathbb{N}$.
Saída: (X_1, Z_1) tais que a coordenada x_{kP} de kP é igual a X_1/Z_1.

1: $X_1 \leftarrow 1$
2: $Z_1 \leftarrow 0$
3: $X_2 \leftarrow x_P$
4: $Z_2 \leftarrow 1$
5: $r \leftarrow 0$
6: **for** $i \leftarrow 254$ **to** 0 **do**
7: $\quad b \leftarrow k_i$
8: $\quad c \leftarrow b \oplus r$
9: $\quad r \leftarrow b$
10: $\quad (X_1, X_2) \leftarrow cswap(X_1, X_2, c)$
11: $\quad (Z_1, Z_2) \leftarrow cswap(Z_1, Z_2, c)$
12: $\quad (X_1, Z_1, X_2, Z_2) \leftarrow ladder(x_P, X_1, Z_1, X_2, Z_2)$
13: **end for**
14: **return** (X_1, Z_1)

A função *ladder*, detalhada no Algoritmo 8, calcula uma duplicação seguida por uma adição diferencial de pontos. Uma adição diferencial é uma adição de dois pontos conhecendo-se sua diferença. Não justificaremos esse fato aqui, nos limitando à sua descrição. A operação não utiliza valores pré-calculados pré-armazenados, dificultando ataques de tempo na latência da hierarquia de memória (cache). A quantia A na linha 10 é o coeficiente A da curva de Montgomery.

Algoritmo 8 Função *ladder* - duplicação e adição diferencial de pontos

Entrada: x, X_1, Z_1, X_2, Z_2
Saída: X_1, Z_1, X_2, Z_2

1: $T_1 \leftarrow X_2 + Z_2$
2: $X_2 \leftarrow X_2 - Z_2$
3: $Z_2 \leftarrow X_1 + Z_1$
4: $X_1 \leftarrow X_1 - Z_1$
5: $T_1 \leftarrow T_1 \cdot X_1$
6: $X_2 \leftarrow X_2 \cdot Z_2$
7: $Z_2 \leftarrow Z_2 \cdot Z_2$
8: $X_1 \leftarrow X_1 \cdot X_1$
9: $T_2 \leftarrow Z_2 - X_1$
10: $Z_1 \leftarrow T_2 \cdot (A+2)/4$
11: $Z_1 \leftarrow Z_1 + X_1$
12: $Z_1 \leftarrow T_2 \cdot Z_1$
13: $X_1 \leftarrow Z_2 \cdot X_1$
14: $Z_2 \leftarrow Z_2 - X_2$
15: $Z_2 \leftarrow Z_2 \cdot Z_2$
16: $Z_2 \leftarrow Z_2 \cdot x$
17: $X_2 \leftarrow T_1 + X_2$
18: $X_2 \leftarrow X_2 \cdot X_2$
19: **return** (X_1, Z_1, X_1, Z_2)

3.3.5. Protocolos de IoT e ECC

A necessidade por protocolos eficientes na Internet das Coisas é bem exemplificada no trabalho de Simplício et al. [Jr. et al. 2016], onde é descrita uma rede de sensores sem fio utilizando protocolos criptográficos baseados em ECC. Nós dessa rede têm baixa capacidade de recursos vitais como processamento, largura de banda, energia, memória, entre

outros. Além do mais, sendo nós sensores autônomos, há boa chance de que sejam fisicamente monitorados ou manipulados por adversários. Portanto, a comunicação entre os nós da rede requer não só implementações eficientes mas também robustas contra ataques por canais laterais, para que possam bem desempenhar suas funções em protocolos criptográficos para acordo de chaves, encriptação e decriptação, entre outras funções de segurança. Descreveremos a seguir alguns ataques por canais laterais.

3.4. Ataques de canal lateral

3.4.1. Ataques temporais

A premissa fundamental de ataques temporais é de que o tempo gasto na execução de uma instrução é influenciado por seus respectivos operandos [Hankerson et al. 2004]. Estudos mostraram [Brumley and Boneh 2003] a viabilidade desse ataque contra servidores executando protocolos como o SSL com RSA devido à latência da comunicação decorrente da rede local.

Como todos os ataques por canais laterais envolvem o monitoramento de uma grandeza física, um requisito para o sucesso de um ataque temporal é que as operações com a chave criptográfica sejam *lentas* o suficiente para serem medidas. Acreditava-se que esse tipo de ataque seria possível apenas em operações de rede ou rádio e jamais em processadores de dispositivos móveis ou em computadores justamente devido a essa limitação.

Porém, uma forma derivada desse ataque, denominada *branch prediction analysis* (análise de preditor de salto) [Aciiçmez et al. 2007], demonstrou ser possível atacar uma implementação do OpenSSL rodando em processadores convencionais (PowerPC, Intel, ARM, etc.). Executando processos maliciosos em sistemas operacionais (Windows, Linux, Android, iOS, BlackBerry, etc.), foi demonstrado ser possível afetar a execução do OpenSSL. Tornando as iterações da exponenciação modular mais lentas, passando de nanosegundos para microsegundos, foi possível detetar e inferir informações sobre a chave privada.

3.4.1.1. Unidade de predição de saltos

As instruções que compõem o código binário de um programa executável podem consumir diferentes quantidades de ciclos de *clock* de acordo com suas respectivas complexidades. Como no decorrer do fluxo de programas podem existir diversas dependências entre as instruções executadas, existe a possibilidade de que valores necessários para a execução de uma determinada instrução ainda não tenham sido calculados.

Quando a instrução depende de um salto condicional, essa situação é denominada *control hazard*. Para que o processador não permaneça ocioso até que o fluxo do programa seja definido, durante o período de decisão especula-se qual deverá ser a próxima instrução executada. Se a predição se mostrar correta (*hit*), o fluxo do programa prossegue sem degradação de desempenho; caso a predição se mostre incorreta (*miss prediction*), o *pipeline* deve ser esvaziado e a instrução correta executada. Observe que uma *miss prediction* acarreta em uma penalidade de ciclos de *clock* que é proporcional à quantidade de

estágios do *pipeline*.

Quando a CPU determina um salto como tomado (feito), ela deve buscar a instrução do endereço-alvo do salto na memória e entregá-la à unidade de execução. Para tornar o processo mais eficiente, a CPU mantém um registro dos saltos executados anteriormente no BTB (*Branch Target Buffer*). Observe que o tamanho do BTB é limitado; logo, alguns endereços armazenados precisam ser removidos para que novos endereços sejam armazenados. O preditor também possui uma parte denominada BHR (*Branch History Registers*) responsável por gravar a história dos registradores usados globalmente e localmente pelo programa [Jean-Pierre et al. 2006].

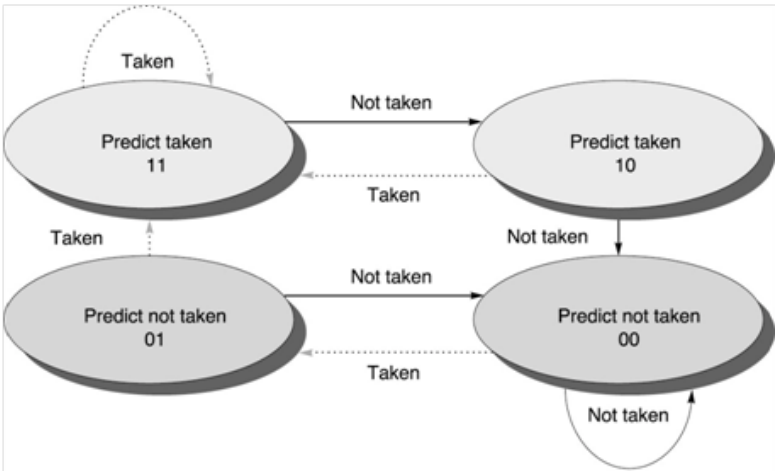


Figura 3.7: Autômato finito descreve o comportamento do preditor de saltos [Hennessy and Patterson 2002].

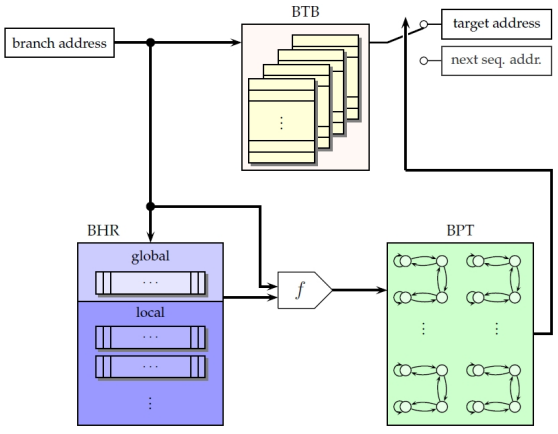


Figura 3.8: Unidade de predição de saltos [Jean-Pierre et al. 2006].

3.4.1.2. Medição direta de tempo

A máquina de estados que descreve as possíveis decisões da BTU possui um número finito de estados; logo, o algoritmo que a descreve é determinístico. O adversário pode

supor que a implementação do RSA utilizou S&M (*Square-and-Multiply exponentiation algorithm*) e MM (*Montgomery Multiplication algorithm* [Hankerson et al. 2004, Denis 2006]) e o BTU possui um autômato finito de apenas dois estados: salto tomado ou não tomado.

Seja d a chave privada, e vamos supor que o adversário conheça seus i primeiros bits e está tentando determinar d_i. Para qualquer mensagem m, o adversário pode simular as primeiras i iterações e obter um resultado intermediário que será a entrada da $(i+1)-$*ésima* iteração. Então, ele gera quatro conjuntos distintos tais que:

$$\begin{aligned}
M_1 &= \{m \mid d_i = 1 \rightarrow m \text{ causa } \textit{misspredict}\textit{ion} \text{ durante } MM\} \\
M_2 &= \{m \mid d_i = 1 \rightarrow m \text{ causa } \textit{hit} \text{ durante } MM \quad\} \\
M_3 &= \{m \mid d_i = 0 \rightarrow m \text{ causa } \textit{misspredict}\textit{ion} \text{ durante } MM\} \\
M_4 &= \{m \mid d_i = 0 \rightarrow m \text{ causa } \textit{hit} \text{ durante } MM \quad\}
\end{aligned}$$

O adversário calcula o tempo médio de execução na multiplicação de Montgomery em cada conjunto M_i. Sendo $d_i = t, t \in \{0,1\}$, então a diferença dos tempos médios de execução para o mesmo valor correto t serão muito mais significativas do que a obtida dos outros dois conjuntos, pois, para o valor incorreto, os valores de tempo de cada multiplicação terão um caráter aleatório. Portanto, se a diferença entre os tempos médios de M_1 e M_2 for muito mais significativa do que M_3 e M_4, então o palpite correto é $d_i = 1$, e $d_i = 0$ caso contrário.

Nesse ataque o adversário precisa saber de antemão o estado do BPU antes do algoritmo de decriptação ser iniciado. Uma possibilidade simples de implementação, porém menos eficiente, seria realizar a análise supondo cada um dos quatro estados iniciais. A segunda abordagem consiste em forçar o estado inicial do BPU de modo que nenhum endereço de salto esteja no BTB. Essa abordagem será fundamentalmente a mesma utilizada em todos os ataques de predição de salto listados a seguir.

3.4.1.3. Forçando BPU à mesma predição assincronamente

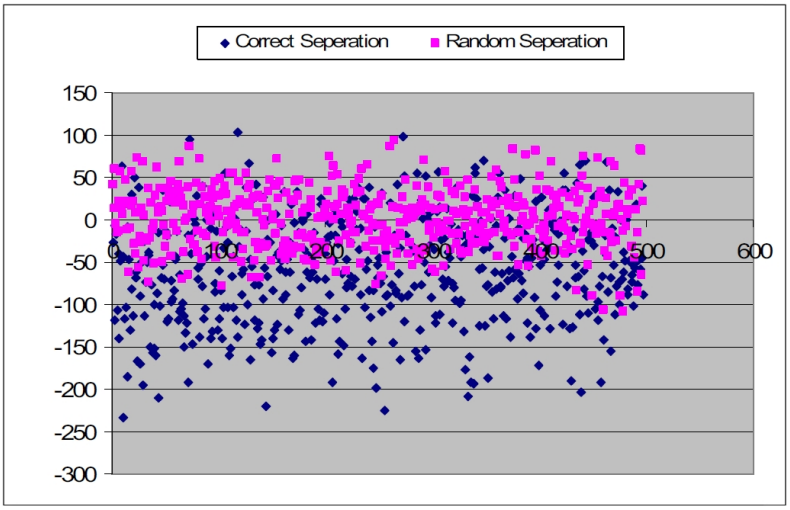
Unidades de processamento que permitem execução concorrente de processos (SMT ou *Simultaneous Multi-Threading* [Silberschatz et al. 2004]) permitem que um adversário execute um processo espião simultaneamente ao programa de encriptação. Dessa forma, o adversário pode fazer com que o valor previsto dos saltos do encriptador nunca estejam no BTB; consequentemente, sempre ocorrerá uma *misprediction* quando o resultado correto, segundo a previsão, seria que o salto fosse tomado. Comparado ao processo anterior, a análise diferencial seria similiar exceto pelo fato de que $d_i = 1$ em caso de *hit* e $d_i = 0$ em caso de *misprediction* durante o cálculo de $m^2 \bmod N$.

O processo espião remove do BTB o endereço-alvo de salto das seguintes maneiras:

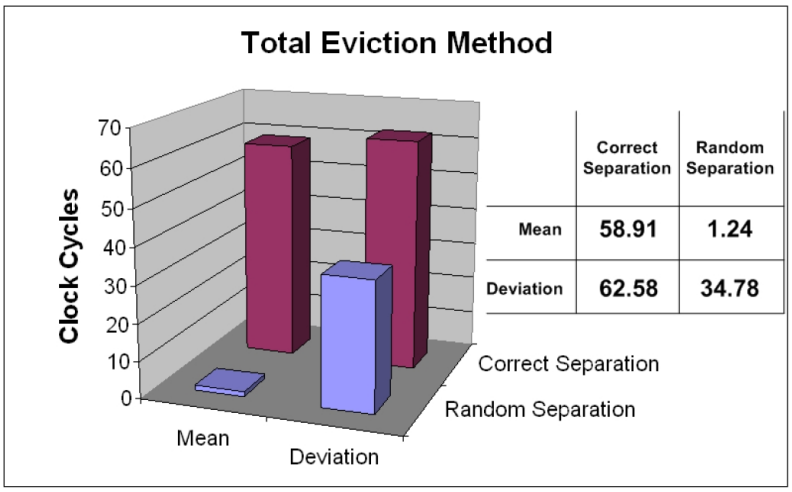
1. (*Total Eviction Method*: todas as entradas do BTB são removidas.

2. (*Partial Eviction Method*): um conjunto de entradas do BTB é removido.
3. (*Single Eviction Method*): apenas o endereço de interesse é removido da tabela.

Obviamente o primeiro método é o de implementação mais simples (assumindo que sejamos capazes de esvaziar todo o BTB entre duas iterações da exponenciação). O diferencial desse ataque é o adversário não ter que saber detalhes de implementação da BPU para ser capaz de criar o processo espião e determinar quais são os bits da chave secreta.



(a) Separações corretas e aleatórias



(b) Maior diferença das médias

Figura 3.9: Resultados práticos do *Total Eviction Method* [Jean-Pierre et al. 2006].

Esse ataque foi aplicado sobre uma implementação do RSA em OpenSSL versão 0.9.7, rodando sob uma workstation RedHat 3. Foram gerados 10 milhões de blocos de mensagens aleatórias e chaves aleatórias de 512 bits. As mensagens foram encriptadas e separadas segundo os critérios acima, sendo assumido como tomado o salto do próximo bit desconhecido.

Na Figura 3.9 (a), o eixo x corresponde aos bits do expoente de 2 até 511, sendo que cada coordenada x_i apresenta os valores das médias das separações corretas e a média das separações aleatórias, denotadas respectivamente por μ_{Y_i} e μ_{X_i}. Analisando todos os pares (μ_{Y_i}, μ_{X_i}), o adversário verifica qual deles teve a diferença mais significativa (Figura 3.9 (b)) e utiliza seus respectivos desvios padrões para determinar o desvio da diferença das médias

$$\mu_Z = \mu_Y - \mu_X = 58.91 - 1.24 = 57.67$$
$$\sigma_Z = \sqrt{\sigma_Y^2 + \sigma_X^2} = \sqrt{62.58^2 - (34.78)^2} = 71.60$$

Sempre que o adversário encontrar $Z > 0$, ele irá supor que seu palpite do valor do *bit* foi correta. O grau de certeza que o adversário pode ter nessas decisões pode ser medido pela probabilidade

$$Pr[Z > 0] = \phi(\frac{0 - \mu_Z}{\sigma_Z}) = \phi(-0.805) = 0.79.$$

Portanto, a probabilidade de suas decisões estarem corretas para essas medidas é de quase 80%, possibilitando ao adversário obter o restante da chave por força bruta.

3.4.2. Ataques de potência

Em um ataque no canal lateral de potência, o adversário analisa sutis variações no consumo de energia elétrica de um dispositivo cujo *hardware* implementa um algoritmo criptográfico (sensores RFID, *smartcards*, *SIM cards*, etc).

Operações com dados sensíveis geram alterações na corrente ou tensão da alimentação do dispositivo, permitindo extrair parcialmente (ou mesmo integralmente) a chave criptográfica e outras informações sensíveis. O primeiro ataque dessa natureza foi apresentado por [Kocher et al. 1999], também autor da célebre pesquisa precursora sobre *time attacks* [Kocher 1996].

3.4.2.1. Ataque de potência simples (SPA)

A tecnologia de semicondutores dominante em microprocessadores, memórias e dispositivos embarcados é a CMOS [Sedra and Smith 1997], sendo inversores lógicos sua unidade básica de construção. Como dispositivos utilizam fontes constantes de tensão, a potência consumida varia de acordo com o fluxo de sinais nos componentes, e esses de acordo com as operações realizadas. Se esse consumo de potência for monitorado com auxílio de um osciloscópio, poderemos estabelecer um rastro de consumo de potência (*power trace*) a cada ciclo do dispositivo.

3.4.2.2. Análise simples de potência sobre ECDSA

Uma das rotinas mais executadas em dispositivos que utilizam ECC são os algoritmos de assinatura digital de curvas elípticas (*ECDSA* ou *Elliptic Curve Digital Signature Algorithm*), tendo como operação central a multiplicação de um ponto por um escalar (Algoritmo 9).

Algoritmo 9 Binary NAF method for scalar multiplication

Entrada: $P \in E(\mathbb{F}_p), k \in \mathbb{N}$
Saída: $Q = kP \in E(\mathbb{F}_p)$
1: $(k_{t-1}, k_{t-2}, ..., k_1, k_0) \leftarrow NAF(k)$
2: $Q \leftarrow \infty$
3: **for** $j \leftarrow t-2$ **to** 0 **do**
4: $\quad Q \leftarrow 2Q$
5: $\quad$ **if** $k_i = 1$ **then**
6: $\quad\quad Q \leftarrow Q + P$
7: $\quad$ **end if**
8: $\quad$ **if** $k_i = -1$ **then**
9: $\quad\quad Q \leftarrow Q - P$
10: $\quad$ **end if**
11: **end for**
12: **return** Q

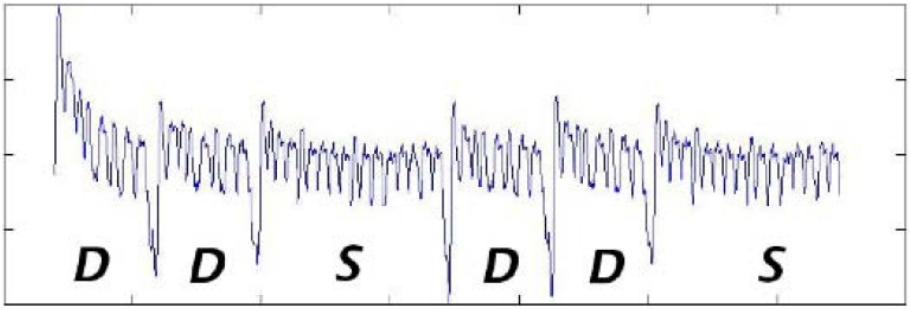


Figura 3.10: Consumo de potência durante cálculo de kP [Hankerson et al. 2004].

O que torna a forma não adjacente de k mais interessante do que sua representação binária é o fato da $NAF(k)$ possuir apenas $1/3$ de dígitos não nulos. Consequentemente uma quantidade muito menor de adições (linhas 6 e 9 do Algoritmo 9) são efetuadas.

Entretanto um adversário que soubesse que o dispositivo implementa um algoritmo *ECDSA* poderia monitorar o consumo de potência utilizando um osciloscópio, obtendo o gráfico mostrado na Figura 3.10. No Algoritmo 9, vemos que adições são realizadas apenas quando $k_i \neq 0$; logo, uma maior quantidade de potência é despendida para dígitos não nulos. Portanto os intervalos curtos denominados D correspondem a iterações em que $k_i = 0$, enquanto intervalos longos denominados S correspondem a iterações em que $k_i \neq 0$. Essa informação torna viável descobrir a chave por meio de ataques de força bruta, pois apenas $1/3$ dos dígitos são não nulos.

A solução mais simples contra SPA consiste em inserir operações redundantes no algoritmo de multiplicação (Algoritmo 10), de modo que a sequência de operações elementares envolvidas sejam realizadas em igual proporção. Comparando o novo *power trace* obtido (Figura 3.11) não é possível diferenciar adições de multiplicações.

Algoritmo 10 Binary NAF method for scalar multiplication resistant to SPA

Entrada: $P \in E(\mathbb{F}_p), k \in \mathbb{N}$
Saída: $Q = kP \in E(\mathbb{F}_p)$
1: $(k_{t-1}, k_{t-2}, ..., k_1, k_0) \leftarrow NAF(k)$
2: $Q \leftarrow \infty$
3: **for** $i = t-1$ **to** 0 **do**
4: $\quad Q_0 = 2Q_0$
5: $\quad Q_1 = Q_0 + P$
6: $\quad Q_0 = Q_{k_i}$
7: **end for**
8: **return** Q

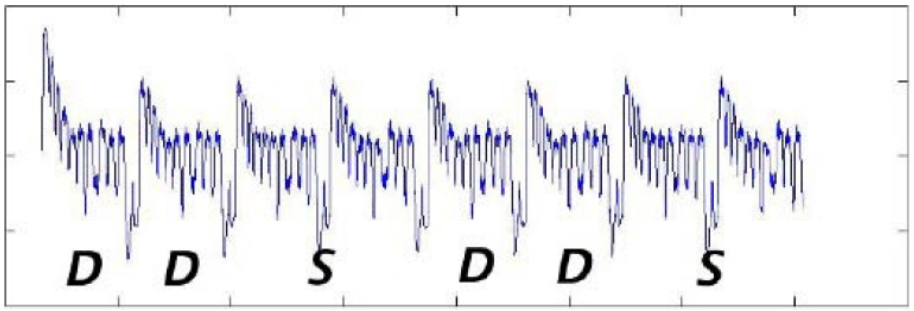


Figura 3.11: Consumo de potência durante cálculo de kP [Hankerson et al. 2004].

3.4.2.3. Ataque de potência diferencial (DPA)

Quando a variação do consumo de potência não é sensível o suficiente em relação às operações executadas por um dispositivo, o adversário pode monitorar como o consumo se comporta em relação ao valor de uma determinada variável. Nesse ataque, primeiramente detetamos uma variável V, influenciada durante um processo de decriptação ou assinatura digital por um texto m e uma porção desconhecida da chave privada. A partir disso, definimos a função de seleção $V = f(k', m)$.

O adversário então coleta milhares de *power traces*, determinando indutivamente todos os bits que compõem a chave privada através do cálculo da derivada dessa função. Para cada bit k'_i corretamente previsto obtemos uma derivada não nula para os valores de k' e m, caso contrário a derivada é nula. O processo é repetido até que cada k'_i seja determinando [Hankerson et al. 2004]. Esse modelo de ataque é conhecido como Análise Diferencial de Potência (DPA ou *Differential Power Analysis*).

3.4.2.4. Análise diferencial de potência sobre ECDSA

Ainda que o Algoritmo 10 tenha sido adotado, podemos aplicar um DPA sobre o processo de ECDSA.

Determinada uma variável V cujo valor influencie o consumo de potência, e uma função de seleção f tal que $V = f(k', m)$, o adversário coleta milhares de *power traces*, estima o tamanho que a porção k' ocupa na chave privada e separa os dados coletados em dois grupos de acordo com o valor previsto de V.

Suponha que o adversário colete os *power traces* durante os cálculos de $kP_1, kP_2, ..., kP_r$, para os quais foi usado o Algoritmo 4 (*Double-and-add always*), que repetimos aqui para facilitar a leitura.

Algoritmo 4 Double-and-add-always

Entrada: $P \in E(\mathbb{F}_p), k = (k_{t-1}, \ldots, k_1, k_0)_2 \in \mathbb{N}$
Saída: $kP \in E(\mathbb{F}_p)$
1: $R_0 \leftarrow P$
2: **for** i from $t-2$ **downto** 0 **do**
3: $\quad R_0 \leftarrow 2R_0$
4: $\quad R_1 \leftarrow R_0 + P$
5: $\quad b \leftarrow k_i$
6: $\quad R_0 \leftarrow R_b$
7: **end for**
8: **return** R_0

Como $P_1, P_2, ..., P_r$ são públicos, ele precisa determinar apenas k. Dado $R_0 = \infty$,

	R_0	R_0	k_{t-1}	$R_0 \leftarrow R_{k_{t-1}}$
1	∞	P	1	P
2	...	...	...	...
3	...	...	...	...
...	...	...	...	...

Tabela 3.1: $k = (1, k_{t-2}, k_{t-3}, ..., k_1, k_0)$

o passo 3 do Algoritmo 4 é trivial e pode ser distinguido de uma operação não trivial pelo *power trace*; logo, o adversário pode facilmente identificar o bit mais a esquerda cujo valor é 1. Tomando $k_{t-1} = 1$, na segunda iteração do algoritmo temos que $R_0 = 2P$ (se $k_{t-2} = 0$) ou $R_0 = 3P$ (se $k_{t-2} = 1$). Consequentemente, na terceira iteração, o valor $4P$ ser computado apenas se $k_{t-2} = 0$. Definindo $k' = k_{t-2}$ e $m = P_i$ (i-ésimo bit do ponto $4P = (4P_1, 4P_2, ..., 4P_i, ..., 4P_r,)$), a função seletora calcula o valor do bit $4P_i$. Se o gráfico do consumo de potência da função apresentar picos, então $k_{t-2} = 0$, caso contrário $k_{t-2} = 1$. Esse processo é repetido até todos os bits de k serem determinados [Hankerson et al. 2004].

	R_0	R_0	k_{t-1}	$R_0 \leftarrow R_{k_{t-1}}$
1	∞	P	1	P
2	$2P$	$4P$	?	?
3	...	...	...	...
...	...	...	...	...

Tabela 3.2: $k = (1, k_{t-2}, k_{t-3}, ..., k_1, k_0)$

	R_0	R_0	k_{t-1}	$R_0 \leftarrow R_{k_{t-1}}$
1	∞	P	1	P
2	$2P$	$4P$	0	$2P$
3	$4P$	$6P$	...	...
...	...	...	...	...

Tabela 3.3: $k = (1, 0, k_{t-3}, ..., k_1, k_0)$

Se a curva elíptica for gerada sobre um $\mathbb{F}_p$ de característica superior a 3, podemos usar um sistema misto de representação de coordenadas no qual P seja representado em um sistema de coordenadas afins, enquanto R_0 e R_1 são representados em coordenadas jacobianas [Hankerson et al. 2004].

Se $P = (x, y)$ no sistema afim, após a primeira atribuição $R_1 \leftarrow P$ teríamos $R_1 = (x : y : 1)$. Então, R_1 seria aleatorizado com $(\lambda^2 x, \lambda^3 y, \lambda)$ e o algoritmo procederia como o usual. Desse modo o adversário estaria impedido de realizar predições baseadas no valor de um bit específico $4P_i$ em sistemas de coordenadas jacobianas aleatorizadas.

3.4.2.5. *High-order DPA*

Uma variação do ataque de potência diferencial é o *high-order* DPA (HODPA), que é baseado em um conjunto de propriedades estatísticas do sinal de potência. Para obter uma melhor taxa de acerto dos *bits* da chave, é necessária uma quantidade maior de amostras. Além disso, existe uma complexidade maior durante a implementação desse ataque [Gierlichs et al. 2009].

Um ponto importante a ser observado é que esse ataque possui a capacidade de se sobrepor à resistência de uma contramedida de *masking* e obter os *bits* da chave. Isso é possível pela análise tanto do valor mascarado como da própria máscara criando uma relação entre ambos. Um problema encontrado na utilização desse ataque é a identificação do momento em que é necessário obter o sinal referente ao valor mascarado ou a máscara.

3.4.2.6. *Template attacks*

O *Template Attack* (TA) utiliza amostras de potência elétrica para executar uma análise, na tentativa de obter os *bits* da chave privada. É considerado o mais poderoso dentre os

ataques de potência diferencial. Existe uma divisão em duas fases do ataque, a saber, a fase de construção e a fase de correlação. A primeira gera um *template* de um padrão obtido no dispositivo alvo. Na segunda é analisada a correlação entre o *template* gerado e as amostras obtidas durante a execução do dispositivo [Özgen et al. 2016]. Esse ataque pode ser combinado com algoritmos de classificação para diferenciar valores da chave daqueles do restante, como é proposto no trabalho de [Özgen et al. 2016]. Uma outra opção é a utilização de algoritmos de reconhecimento de padrão com aprendizado de máquina.

3.5. Ataques e contramedidas para criptografia simétrica

Implementações inseguras de criptografia simétrica podem ser suscetíveis a diversos tipos de ataques de canal lateral. Mesmo implementações de algoritmos de alto desempenho podem ser vulneráveis a canais laterais de tempo executados localmente (via latência da memória *cache*) ou remotamente (tempo de resposta na comunicação em rede). Contramedidas comuns para criptografia simétrica contra ataques de canal lateral envolvem execução em tempo constante e mascaramento.

3.5.1. Ataques por canal lateral de tempo e contramedidas

Um ataque simples contra a utilização de criptografia simétrica envolve a aplicação de funções de resumo criptográfico para autenticação. Nessas aplicações, é comum que o algoritmo simétrico produza um autenticador ou *hash* das credenciais que precisa ser comparado com o resultado correto. Durante a comparação de cadeias de caracteres, pequenas variações no tempo de execução podem fornecer informação útil para o adversário reduzir enormemente a complexidade de um ataque de busca exaustiva. Ao comparar cadeias, se a interrupção do laço acontecer exatamente na primeira posição diferente, o adversário é capaz de realizar um ataque de busca exaustiva em cada *byte* individualmente, baseado no tempo de resposta. Desta forma, é possível determinar o autenticador de uma mensagem ou *hash* de uma senha, por exemplo, sem conhecimento de segredos. O trecho de código abaixo ilustra uma forma segura de comparação, cujo tempo de execução é invariante e saturado para o pior caso:

```
int cmp_const(const void * a, const void * b,
              const size_t size) {
    const unsigned char *_a = a, *_b = b;
    unsigned char result = 0;
    size_t i;
    for (i = 0; i < size; i++) {
        result |= _a[i] ^ _b[i];
    }
    return result;
}
```

Desvios condicionais dentro de algoritmos criptográficos podem ser outra fonte de problemas, especialmente em processadores modernos com execução especulativa e predição de desvios. A remoção de desvios condicionais envolve a aplicação de técnicas

de programação sem desvios condicionais. A aplicação correta dessas técnicas é altamente dependente do algoritmo sendo estudado, mas uma generalização útil é calcular os dois ramos do desvio condicional simultaneamente e utilizar operações mascaradas para selecionar o valor correto apenas ao final da execução. A ideia é ilustrada pelo trecho de código abaixo para seleção entre dois valores em tempo constante, dependendo de um bit de condição:

```
unsigned select(unsigned a, unsigned b,
                unsigned bit) {
  /* -0 = 0, -1 = 0xFF....FF */
  unsigned mask = - bit;
  unsigned ret = mask & (a^b);
  ret = ret ^ a;
  return ret;
}
```

Outra possibilidade é utilizar uma variante da função de seleção para alterar em tempo constante os valores de entrada do trecho de código protegido ser executado.

Acessos à memória devem também ser cuidadosamente protegidos. No contexto de cifras de bloco, a preocupação se concentra na representação de caixas de substituição como vetores ou introdução de qualquer tabela pré-calculada para acelerar operações sobre bits realizadas pela cifra. O algoritmo AES ilustra muito bem o problema, pois seu desempenho depende enormemente da eficiência das caixas de substituição, motivando o implementador a utilizar tabelas simples. Entretanto, um adversário capaz de monitorar o comportamento da memória *cache* pode determinar que porções da caixa de substituição são usadas na etapa de cifração e recuperar informação sensível [Bernstein 2004, Percival 2005, Bonneau and Mironov 2006, Tromer et al. 2010]. Por exemplo, um ataque *Flush and Reload* utiliza a instrução `clflush` em processadores Intel para eliminar endereços de páginas compartilhados da memória *cache* e verifica se os dados retornam à cache após permitir que o programa atacado execute um pequeno número de instruções [Yarom and Falkner 2014]. Por funcionar no nível mais baixo e compartilhado da memória *cache*, o ataque torna-se possível entre núcleos distintos e contra ambientes virtualizados.

Existem contramedidas de diversos tipos para mitigar o problema. Uma opção simples é adotar arquiteturas com latência de acesso uniforme à memória, como alguns microcontroladores simples. Outra possibilidade é utilizar uma implementação em hardware do algoritmo, quando disponível, que deve oferecer uma superfície de ataque menor. Alternativas mais sofisticadas para implementação em software são *bitslicing*, onde as operações sobre bits são realizadas explicitamente, sem ajuda de tabelas pré-calculadas, e o impacto em desempenho é reduzido pela aplicação do mesmo circuito lógico a bits de diferentes variáveis simultaneamente [Biham 1997, Käsper and Schwabe 2009]. Para tabelas pequenas, também é possível utilizar uma instrução para acesso a tabela armazenada em um registrador [Hamburg 2009] ou percorrer a tabela inteira a cada leitura, utilizando a função de seleção apresentada anteriormente para realizar uma cópia condicional entre o valor lido e um valor atual da variável de interesse.

3.5.2. Ataques de potência e contramedidas

Ataques de potência também são possíveis contra implementações de primitivas simétricas, por exemplo algoritmos como o HMAC para autenticação de mensagens (MAC) utilizando uma função de resumo criptográfico H como fonte de segurança. Para uma chave simétrica k, o HMAC calcula o autenticador para M como

$$\text{HMAC}_k(M) = H((k \oplus opad) \parallel H((k \oplus ipad) \parallel M)),$$

onde $ipad$ é a cadeia produzida pela repetição do valor `0x36` e $opad$ pela repetição do valor `0x5C`. Ataques diferenciais de potência foram demonstrados contra implementações inseguras do HMAC utilizando tanto funções de resumo construídas segundo o paradigma Merkle-Damgård quanto aquelas baseadas em cifras de bloco.

A característica principal de um ataque diferencial de potência é que, em um certo ponto na execução do algoritmo, uma variável combina conhecimento de uma função f sobre um valor secreto fixo com outro valor conhecido pelo adversário. O adversário pode então formular hipóteses sobre o valor secreto fixo e aplicar a função de acordo com um certo modelo de vazamento. Pela captura de traços de consumo de energia, é possível verificar a validade das hipóteses. Supondo que o adversário possua controle do dispositivo, obtenha informação sobre a distância de Hamming entre estados internos consecutivos durante o cálculo instâncias de HMAC, e conheça as mensagens cuja autenticação é verificada, o objetivo é combinar informação fixa desconhecida com informação variável conhecida.

O ataque proposto por [McEvoy et al. 2007] se concentra na primeira execução da função de resumo interna $H(k \oplus ipad)$ ao HMAC, que é invariante em execuções distintas. Esse cálculo irá produzir um valor de resumo intermediário, e o processamento prossegue com a mensagem conhecida M. Portanto, o objetivo do ataque é recuperar o valor intermediário pela formulação de hipóteses verificadas após o processamento de M. Na fase final, o ataque é repetido na função de resumo externa do HMAC, permitindo a forja de autenticadores para mensagens da escolha do atacante, sem conhecimento da chave criptográfica. Um ataque similar pode ser montado contra funções de resumo construídas a partir de cifras de bloco, mesmo quando a cifra de bloco é ideal e resistente contra ataques de canal lateral. [Okeya 2006] demonstrou ser possível recuperar a chave criptográfica completa nesse caso, permitindo forja de autenticadores para mensagens escolhidas pelo adversário.

Ataques cúbicos [Dinur and Shamir 2012] são ataques genéricos de recuperação de chaves que podem ser aplicados automaticamente a qualquer criptossistema em que um único bit de informação pode ser representado por um polinômio de grau pequeno na chave e texto claro, como cifras de fluxo baseadas em registradores de deslocamento. O ataque consiste em somar o bit de saída de todos os possíveis valores de um subconjunto de bits públicos de entrada, escolhidos de forma que o bit resultante seja uma combinação linear de bits secretos. Aplicações repetidas da mesma técnica produzem relações lineares entre bits secretos que podem ser resolvidas para descobrir esses bits. Os autores demonstram que bits públicos dessa forma existem com alta probabilidade quando a cifra aproxima um polinômio aleatório de grau suficientemente pequeno e podem ser encontrados em uma fase de pré-computação. A versão de canal lateral do ataque funciona por

meio da captura de bits individuais que satisfazem as características do ataque, tipicamente por um ataque de potência.

Mascaramento

Mascaramento é uma das contramedidas contra ataques de potência mais investigadas na literatura para proteger valores sensíveis, como texto claro durante a encriptação ou criptogramas durante a decriptação. Como a informação calculada durante esses processos se tornará a saída dos algoritmos, todos os valores intermediários precisam ser protegidos durante todo o tempo. Ao se aplicar mascaramento, um conjunto de n máscaras $m_0, \ldots, m_n$ pseudo-aleatórias é utilizado por meio da operação `XOR` para representar valores intermediários, de forma que a informação vazada por canais laterais não mais está correlacionada com informação secreta legítima. O valor mascarado m com $d+1$ máscaras é dado por $m = \bigoplus_{i=0}^{d} m_i = m_0 \oplus m_1 \oplus \ldots \oplus m_d$. As operações em um corpo binário podem então ser adaptadas para funcionar com valores mascarados:

1. Toda operação linear L sobre um valor mascarado consiste em aplicar a mesma operação sobre as máscaras:

$$L(m) \equiv L(m_0 \oplus m_1 \oplus \ldots \oplus m_d) \equiv L(m_0) \oplus L(m_1) \oplus \ldots \oplus L(m_d)$$

2. Uma operação `NOT` pode ser calculada como:

$$\neg m \equiv \neg m_0 \oplus m_1 \oplus \ldots \oplus m_d$$

3. Uma operação `XOR` entre valores mascarados $a = \bigoplus_{i=0}^{d} a_i$ e $b = \bigoplus_{i=0}^{d} b_i$ pode ser calculada como:

$$a \oplus b \equiv \bigoplus_{i=0}^{d} a_i \oplus \bigoplus_{i=0}^{d} b_i \equiv \bigoplus_{i=0}^{d} (a_i \oplus b_i)$$

4. Uma operação `AND` entre $a = \bigoplus_{i=0}^{d} a_i$ e $b = \bigoplus_{i=0}^{d} b_i$ é mais complicada e exige um algoritmo específico (Algoritmo 11) [Ishai et al. 2003].

Algoritmo 11 Operação `AND` aplicada sobre valores mascarados a e b.

Entrada: Valores (a_i) e (b_i) tais que $\oplus_{i=0}^{d} a_i = a$ e $\oplus_{i=0}^{d} b_i = b$.
Saída: Valores (c_i) tais que $\oplus_{i=0}^{d} c_i = a \wedge b$

```
1:  for i from 0 to d do
2:      r_{i,i} ← 0;
3:      for j from i + 1 to d do
4:          r_{i,j} ← random();
5:          r_{j,i} ← (r_{i,j} ⊕ (a_i ∧ b_j)) ⊕ (a_j ∧ b_i);
6:      end for
7:  end for
8:  for i from 0 to d do
9:      c_i ← a_i ∧ b_i;
10:     for j from 0 to d do
11:         c_i ← c_i ⊕ r_{i,j};
12:     end for
13: end for
```

Estas observações permitem que qualquer algoritmo utilizando operações binárias seja implementado de maneira mascarada. A contramedida causa baixa penalidade em desempenho quando aplicada sobre sequências de operações binárias lineares, mas o impacto aumenta consideravelmente sobre operações binárias não-lineares, que exigem formulação distinta da computação.

3.5.3. Oráculo de preenchimento

Em um ataque de oráculo de preenchimento, um servidor (oráculo) vaza informação sobre quanto do preenchimento de uma mensagem está no formato correto. O adversário entào manipula partes do criptograma para conseguir decriptar (ou algumas vezes encriptar) mensagens utilizando a chave criptográfico do oráculo, mesmo sem conhecê-la. Por se tratar de um ataque ativo, a forma mais simples de mitigá-lo é autenticar criptogramas utilizando um algoritmo de MAC, com verificação realizada em tempo constante. O ataque original foi proposto por Vaudenay [Vaudenay 2002], mas implementações de protocolos criptográficos em que o ataque foi mitigado mostraram-se posteriormente vulneráveis à versão do ataque no canal lateral de tempo *Lucky Thirteen* [AlFardan and Paterson 2013].

Em criptografia simétrica, o ataque é particularamente efetivo contra o modo de operação CBC para cifras de bloco. Suponha que o atacante tenha capturado o vetor de inicialização IV e três blocos de criptograma C_1, C_2, C_3, e queira decriptar o segundo bloco para recuperar M_2, sabendo que C_3 possui preenchimento correto. Se o adversário manipula o último byte de C_1 e submete (IV, C_1, C_2) para o servidor, a decriptação deverá alterar completamente o resultado de M_1 e o último byte de M_2. O servidor então verifica o preenchimento correto de M_2 e retorna essa informação para o adversário. Para descobrir o último byte de M_1, basta somar ao último byte de C_1 uma estimativa do último byte de M_2 e o byte correto de preenchimento, de forma que a ausência de erro de preenchimento indica que a estimativa estava correta. Após descobrir o último byte de M_2 com várias tentativas, o ataque pode continuar sobre os bytes individuais restantes do bloco.

3.6. Ataques e contramedidas para ECC

3.6.1. Níveis em que os ataques SCA podem ser aplicados

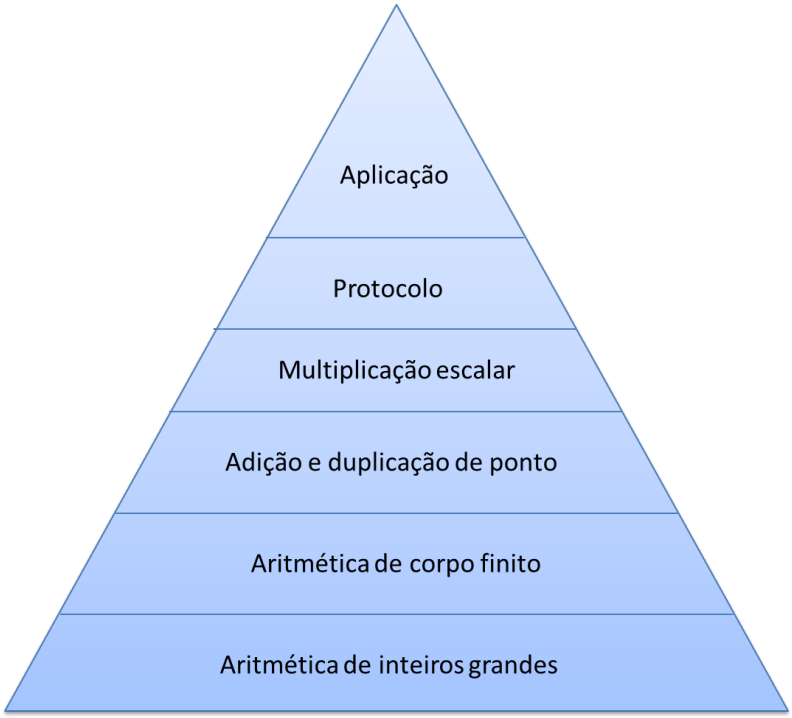


Figura 3.12: Pirâmide de implementação de criptografia baseada em curvas elípticas (ECC). Qualquer uma destas camadas pode ser vulnerável a ataques por canais laterais.

A implementação de protocolos criptográficos baseados em curvas elípticas depende da existência de implementações de um conjunto de operações básicas: multiplicação de ponto por escalar, adição e duplicação de ponto, aritmética de corpo finito e aritmética de inteiros grandes. Protocolos criptográficos são implementados a partir dessas primitivas e, então, utilizados em aplicações as mais variadas. A Figura 3.12 ilustra tais operações em camadas, onde a implementação de uma camada depende apenas da existência de funcionalidades na camada imediatamente inferior.

A representação em camadas, entretanto, esconde a interdependência que existe entre operações. Na prática, em bibliotecas de primitivas e protocolos criptográficos, há interações entre níveis não adjacentes, pois tais interações são relevantes para desempenho e, em alguns casos, segurança.

Em uma implementação de uma dada aplicação, qualquer um dos níveis da pirâmide pode estar vulnerável a ataques por canais laterais. Com relação ao canal de tempo, por exemplo, se uma operação não é de tempo constante com relação à chave ou valores intermediários dependentes da chave, então todas as operações em níveis superiores vazam informações por diferenças potencialmente observáveis de tempo.

3.6.2. Ataques de tempo e SPA ao algoritmo double-and-add-not-always

3.6.2.1. Ataque de tempo ao algoritmo double-and-add-not-always

Considerando que o SPA de tempo se baseia na variação de tempo do algoritmo relativamente ao valor da chave, o sinal mais simples de que uma implementação é insegura são desvios condicionais (*if*s) cujo comportamento dependa apenas de *bits* da chave privada.

Nesse contexto, o ataque de tempo aplicado a uma implementação do RSA de Kocher [Kocher 1996] pode ser aplicado a implementações de curvas elípticas: tudo o que faz o adversário é medir o tempo de execução do algoritmo, advinhar os *bits* da chave, e validar o resultado testando a chave numa operação de decriptação.

No trabalho de [Danger et al. 2013] é descrito um ataque a uma implementação genérica de ECSM, que segue os seguintes passos: primeiramente o adversário coleta o tempo de execução de diferentes ECSMs com o mesmo escalar e diferentes pontos base. Para cada ECSM, simula a computação usando um simulador de software com exatamente a mesma implementação do chip alvo, "chutando"o valor do bit i do escalar.

Agora suponha, sem perda de generalidade, que a hipótese é de que o valor do bit seja zero. O adversário então separa os diferentes tempos de execução em dois conjuntos, S_1 e S_2. Caso uma redução seja necessária ao final da execução, o tempo obtido é armazenado em S_2, senão é armazenado em S_1. Após toda a execução, é feita uma média dos tempos armazenados em S_1 e S_2 e, se a diferença entre as médias for aproximadamente o tempo de execução da redução, a hipótese estava correta.

3.6.2.2. Ataque SPA ao algoritmo double-and-add-not-always

O algoritmo *double-and-add-not-always* executa em tempo constante; contudo, o ataque SPA (com ou sem power model) pode ser aplicado para distinguir os padrões no *trace* das iterações com apenas DBL (onde o *bit* da chave é igual a 0) daquelas com DBL+ADD (com *bit* da chave igual a1). Um ataque deste tipo segmenta/divide o *trace* de potência de uma execução do ECSM em *subtraces*, cada uma contendo uma operação de ponto (ADD ou DBL).

Se o tempo de execução de uma operação de ADD for diferente do tempo de DBL, então o comprimento dos *subtraces* revela onde estão os ADDs e consequentemente os bits do escalar. Se o tempo da operação ADD e o tempo da DBL forem iguais, e uma fórmula unificada para ADD and DBL é utilizada, então, se for aplicada correlação (coeficiente de correlação de Pearson) entre todos os pares de *subtraces*, o resultado será que a correlação será mais alta para os pares de *subtraces* cuja operação correspondente é a mesma (i.e., (ADD,ADD) ou (DBL,DBL)), identificando portanto as operações de ponto e consequentemente os *bits* do escalar.

Mesmo que a implementação seja vulnerável a SPA é desejável que execute em tempo constante, o que cria uma base para a implementação de outras contramedidas.

3.6.3. Algoritmo Double-and-add-always de Coron [Coron 1999]

O algoritmo *double-and-add-always* de [Coron 1999] (Algoritmo 4) utiliza uma adição falsa de ponto conhecida como *dummy point addition*, quando o bit escalar k_i é 0, tornando a sequência de operações computadas durante a multiplicação escalar independente do valor do escalar.

Portanto, o adversário não consegue, em princípio, adivinhar o valor do *bit* k_i por SPA. Uma desvantagem desse método é a sua baixa eficiência. Ele requer $nA + nD$ operações no corpo, um aumento de 33% nas operações do corpo em comparação com a versão desprotegida binária do algoritmo *left-to-right*.

3.6.3.1. Ataque de duplicação de Fouque e Valette [Fouque and Valette 2003]

O ataque de duplicação de Fouque-Valette [Fouque and Valette 2003] é baseado no fato de que é possível detetar se dois valores intermediários são iguais quando o algoritmo calcula a multiplicação escalar para pontos escolhidos P e $2P$. Diversos algoritmos protegidos contra SPA são vulneráveis ao ataque de Fouque e Valette, como o algoritmo clássico *left-to-right* binário, incluindo as variações do mesmo, como o *double-and-add-always* de Coron.

No algoritmo de Coron (Algoritmo 4), a soma parcial é calculada da seguinte maneira: $S_m(P) = \sum_{i=1}^{m} k_{n-i}2^{m-i}P = \sum_{i=1}^{m-1} k_{n-i}2^{m-1-i}(2P) + k_{n-m}P = S_{m-1}(2P) + k_{n-m}P$. Assim, o resultado intermediário do algoritmo no passo m quando a entrada for P, será igual ao resultado intermediário no passo $m-1$ quando a entrada for $2P$, se e somente se $k_{n-m} = 0$. Portanto, um adversário pode obter o escalar secreto comparando a computação de duplicação no passo $m+1$ para P e no passo m para $2P$ para recuperar o bit k_{n-m}. Se ambas as computações forem idênticas, $k_{n-m} = 0$, senão $k_{n-m} = 1$. Isso mostrou que com apenas duas requisições de multiplicação escalar escolhidas pelo adversário, é possível recuperar todos os *bits* do escalar. [1]

3.6.4. Contramedidas

3.6.4.1. Aleatorização do escalar (Scalar Randomization-SR)

Aleatorização do escalar é aplicada no início da multiplicação escalar, da seguinte maneira:

- Aleatoriamente selecione $r \in_R \{0,1\}^n$, para um valor pequeno de n. O valor $n = 32$ é uma escolha razoável, que balanceia segurança e eficiência.
- Calcule $k' \leftarrow k + r|E|$;
- Utilize k' no lugar de k.

O custo da aplicação dessa contramedida depende diretamente de: geração dos bits pseudo-aleatórios de r; n iterações da ECMS; as adições e multiplicações para calcu-

[1] O adversário coleta um traço de energia para a computação de kP e um para a computação de $k(2P)$. Para cada iteração $m = 1,\ldots,n$ ele executa o ataque como descrito e encontra k_{n-m}.

lar k' e n bits da SRAM. Já no quesito eficiência é necessário verificar que se é vulnerável a ataques de *template* online (OTA) [Batina et al. 2014], entre outros.

3.6.4.2. Realeatorização de coordenadas projetivas (Projective Coordinates (Re)randomization (CR) [Coron 1999]

As coordenadas projetivas usadas no Algoritmo Montgomery Ladder são $(X : Z)$. Os passos para aplicar essa contramedida são:

- Gere um valor pseudo-aleatório $\lambda \mathbb{F}_p \backslash \{0\}$;
- Calcule $Z_2 \leftarrow \lambda$ e $X_2 \leftarrow u \cdot \lambda$, onde u é a coordenada x do ponto P da entrada;
- Utilize $P' = (X_2 : Z_2)$ no lugar de P.

Existe um custo de geração de $\lceil log_2(p) \rceil$ bits aleatórios. Além disso, algumas vantagens ligadas a essa randomização das coordenadas, como por exemplo a resistência a ataques online de *template* [Batina et al. 2014], pode ser aplicada para as curvas de Edwards, *twisted* Edwards e Montgomery. Por fim, a cada iteração do método de ECSM ela pode ser aplicada, sendo apenas necessária a coordenada x no caso de Montgomery Ladder.

3.6.4.3. Point Blinding (PB) [Coron 1999]

Essa contramedida pode ser aplicada antes da primeira multiplicação escalar, seguindo os passos seguir:

- Gerar um ponto aleatório R no subgrupo;
- Pré-calcule e armazene $S = kR$;
- No início de cada operação escalar calcule $T \leftarrow P + R$ e $Q \leftarrow kT$;
- Atualize R e S da seguinte maneira, com t aleatório:
  - $R \leftarrow (-1)^t 2R$;
  - $S \leftarrow (-1)^t 2S$;
- Retorne $W = Q - S$.

O custo dessa contramedida é de 2 adições (ECADD) e 2 duplicações (ECDBL) elípticas, e memória SRAM para valores temporários. Provavelmente também é utilizada memória não volátil para armazenar R e S. *Point Blinding* protege contra SVA horizontal [Murdica et al. 2012], RPA [Goubin 2003], e ZPA [Akishita and Takagi 2003], mas é vulnerável a OTA [Batina et al. 2014].

3.6.4.4. Particionamento de escalar (Scalar Splitting (SS) [Clavier and Joye 2001])

A aplicação dessa contramedida segue os seguintes passos, onde k é o escalar original de n bits:

- Gere um inteiro $k_1 < k$ aleatoriamente e calcule:
  1. $k_2 \leftarrow k - k_1$;
  2. $Q \leftarrow k_1 P$;
  3. $T \leftarrow [k_2]P$;
  4. $R \leftarrow Q + T$.

O custo dessa contramedida é a execução de 1 (multiplicação escalar) ECSM de n bits e 1 adição (ECADD), que pode ser reduzido se for empregado o truque de Shamir para multiplicação escalar dupla (versão regular) [Ciet and Joye 2003]. A eficácia está ligada à resistência aos ataques TA, SPA clássico e CPA clássico. As variantes dessa contramedida são: *Euclidean splitting* [Ciet and Joye 2003] e *multiplicative splitting* [Trichina and Bellezza 2003], ambas com a mesma eficácia da versão original, também conhecida como *additive splitting*.

3.6.5. Ataques Horizontais (HA)

Ataques horizontais (HA) são uma metodologia para ataques por canal lateral, cujos alvos são as principais operações criptográficas em protocolos baseados em RSA e ECC, a exponenciação modular e a multiplicação escalar, respectivamente. Em teoria, tais ataques permitem recuperar os bits do expoente/escalar secreto através da análise de *traces* individuais, isto é, apenas um único *trace* obtido do alvo é suficiente; portanto, são eficazes contra implementações protegidas por contramedidas como SR, CR, PB e SS.

Um requisito básico dos ataques horizontais é o conhecimento do algoritmo de multiplicação escalar. De posse de tal informação, o adversário pode escolher, dentre outros, os seguintes métodos ou
emphdistinguishers: correlation analysis, collision-correlation analysis e *cluster analysis*.

O método de correlation analysis [Clavier et al. 2010] segue o mesmo princípio da análise de potência por correlação (CPA)[2] aplicada a um conjunto de *traces*, na configuração vertical. A diferença para o contexto horizontal é de que um único *trace* é dividido em vários segmentos e para cada um destes segmentos um valor intermediário hipotético é atribuído, em relação a um chute sobre o valor da chave. A correlação entre amostra e valor hipotético é calculada do mesmo modo que CPA, e os modelos de vazamentos usualmente utilizados são o peso de Hamming e a distância de Hamming. Este método funciona contra implementações protegidas somente com SR, ou quando CR é também aplicada mas o parâmetro aleatório utilizado é curto, e ataques por força bruta ao valor de tal parâmetro são viáveis.

O método de collision-correlation analysis [Bauer et al. 2013, Bauer and Jaulmes 2013, Clavier et al. 2012, Witteman et al. 2011b, Walter 2001]

[2] *Correlation power analysis*.

computa a correlação ou distância euclideana entre diferentes segmentos de um *trace*. O objetivo principal é identificar a ocorrência de um mesmo dado intermediário em diferentes partes de um *trace*, e com isso derivar os bits do escalar secreto. Para tanto, o adversário deve ter conhecimento do algoritmo de ECSM empregado. Em teoria, este método é viável contra implementações envolvendo combinações de contramedidas clássicas.

A maioria das formas de ataques horizontais requer pré-processamento avançado dos *traces*, caracterização e avaliação de vazamento antes da aplicação de distinguishers. Os principais problemas da abordagem horizontal são de que extrair o vazamento a partir de um único *trace* tipicamente apresenta fortes limitações e desafios, como o alto nível de ruído e a indisponibilidade de amostras rotuladas. Em particular, métodos de avaliação de vazamento, como o TVLA [Goodwill et al. 2011], requerem amostras rotuladas e isso não é possível quando a contramedida SR é aplicada.

Métodos baseados em aprendizado não supervisionado, mais especificamente aqueles baseados em *clustering*, foram recentemente aplicados para resolver tais limitações e têm se mostrado capazes de produzir resultados práticos. Heyszl et al [Heyszl et al. 2014] propõem aplicar classificação por *clustering* a um único *trace* para possibilitar a identificação de classes específicas de operações; este método funciona bem para medições com baixo ruído e requer uma estação de EM composta de múltiplas sondas. Perin et al. [Perin et al. 2014], consideram uma abordagem heurística baseada na diferença de médias para a seleção de pontos de interesse. Além disso, ambas soluções usam um único *trace* como entrada para a etapa de avaliação de vazamento, o qual pode ter sido muito afetado por uma grande quantidade de ruído. Perin e Chmielewski [Perin and Chmielewski 2015] fornecem uma metodologia para ataques horizontais baseados em *clustering* que foca em corrigir as deficiências dos trabalhos mencionados anteriormente.

3.6.6. Ataque HCA ao Montgomery Ladder c/ SR + CRR

3.6.6.1. Preparação: filtragem, segmentação e alinhamento

Em ataques horizontais, devido a problemas como o alto nível de ruído presente em um *trace*, fenômenos como clock drift[3] e variações no tempo em que o dispositivo de medição (osciloscópio) inicia a medição após o recebimento do sinal de trigger (*trigger jitter*), é fundamental o pré-processamento dos *traces* medidos antes de se iniciar a análise, particularmente as operações de filtragem, segmentação e alinhamento.

Filtragem. A Section 3.6.6.1 mostra um *trace* não filtrado e o mesmo *trace* após aplicação de filtro baixa, onde pode-se identificar com mais clareza características do sinal, como periodicidade e amplitude, bem como as rodadas ou iterações. É recomendável que filtros analógicos sejam aplicados, sempre que possível, de modo que o sinal que é digitalizado pelo osciloscópio contenha apenas as frequências desejadas, permitindo o uso da menor resolução (range) vertical suportada pelo dispositivo, o que pode não ocorrer caso picos

[3] *Clock drift* ou *clock jitter* é o desvio do sinal de clock real em relação ao sinal verdadeiramente periódico de referência. Devido ao clock drift, o sinal de clock real não é periódico, mas sim aproximadamente periódico.

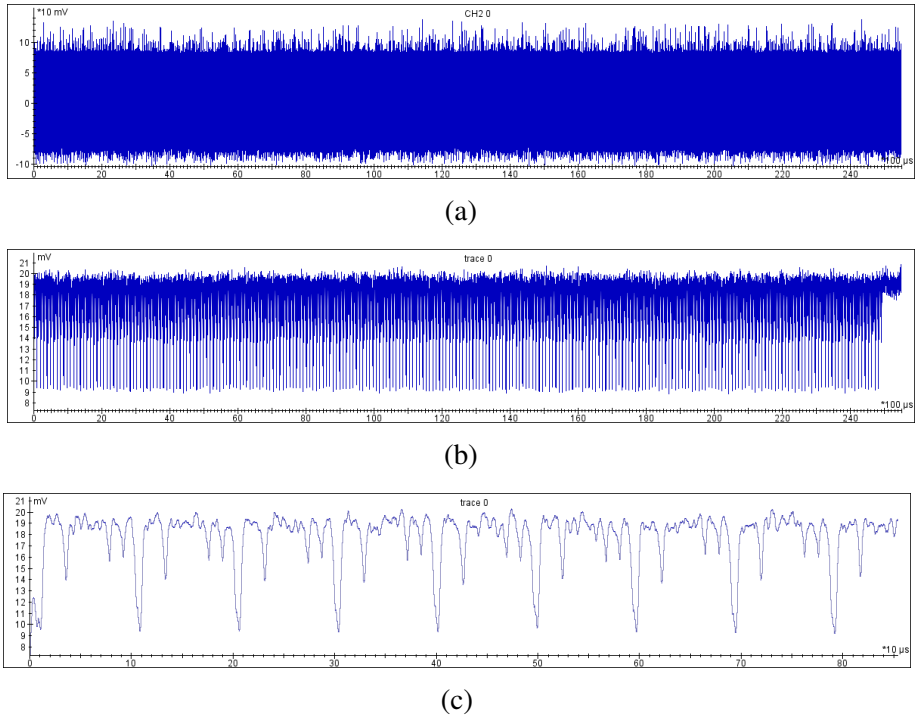


Figura 3.13: *Trace* de radiação eletromagnética de uma execução de ECSM com algoritmo Montgomery Ladder: (a) original não filtrado; (b) após aplicação de filtro baixa e (c) zoom no início deste último. *Traces* processados com Inspector [Riscure 2016].

em frequências indesejadas estejam presentes.

Segmentação. Os *traces* de potência medidos tipicamente correspondem à execução completa da operação criptográfica; sendo assim, são contíguos e contêm todas as rodadas (rounds) ou sub-operações executadas; no contexto da multiplicação escalar, as rodadas são as n iterações do laço do algoritmo de multiplicação escalar implementado. Tais *traces* devem ser primeiramente segmentados em iterações, de modo que um conjunto de n *subtraces* é obtido, cada um contendo as amostras correspondentes à respectiva iteração.

Tal segmentação pode ser realizada de diversas maneiras. Um método ingênuo é identificar os índices das amostras de início e fim da execução do laço da ECSM, e então dividir este segmento em n segmentos de igual (ou quase igual) comprimento, cada qual correspondendo a uma iteração. Tal método apresenta dois problemas: o primeiro é de que em geral é difícil identificar as amostras de início e fim; o segundo é que o comprimento dos segmentos das iterações podem variar devido ao clock *jitter*. Um método mais robusto, que ameniza tais dificuldades, é aplicar um filtro passa baixa forte, de modo a identificar segmentos do *trace* que se repetem de uma iteração para outra; localizar então picos nestes segmentos cuja distância entre si é aproximadamente a mesma (Tal distância é o valor aproximado do comprimento daquela iteração.) e por fim cortar o *trace* original utilizando tais comprimentos como referência. No entanto, devido às dificuldades anteriormente mencionadas, é provável que a segmentação obtida não seja perfeita, e portanto amostras do início de uma iteração poderão estar presentes no *trace* da iteração anterior,

bem como amostras do fim de uma iteração poderão estar no início do *trace* da iteração seguinte; algo análogo acontece com relação à primeira e à última iterações e as partes do *trace* externas ao laço da ECSM.

Alinhamento. Dado que a segmentação provavelmente não será perfeita, uma operação aritmética realizada no intervalo de amostras $tr_i[s..e]$ no *trace* da iteração i muito provavelmente ocorrerá em um itervalo diferente $tr_j[s'..e']$ no *trace* da iteração j. Logo, é necessário alinhar todos os *subtraces* de iterações da ECSM. A Section 3.6.6.1 mostra alguns *traces* resultantes da segmentação, antes e após alinhamento estático por correlação. Diversos algoritmos para alinhamento de *traces* para SCA são propostos na literatura, dentre eles o alinhamento estático e o alinhamento elástico [Woudenberg et al. 2011]. O alinhamento estático é um método simples que consiste, grosso modo, em escolher um *trace* de referência e deslizar incrementalmente o *trace* a ser alinhado sobre o de referência e computar a distância entre eles de acordo com alguma métrica, p.ex. o coeficiente de correlação de Pearson; após deslizar um dado número de posições dentro de uma janela, considera-se que o *trace* estará alinhado se for deslocado (*shifted*) até a posição cuja distância foi mínima.

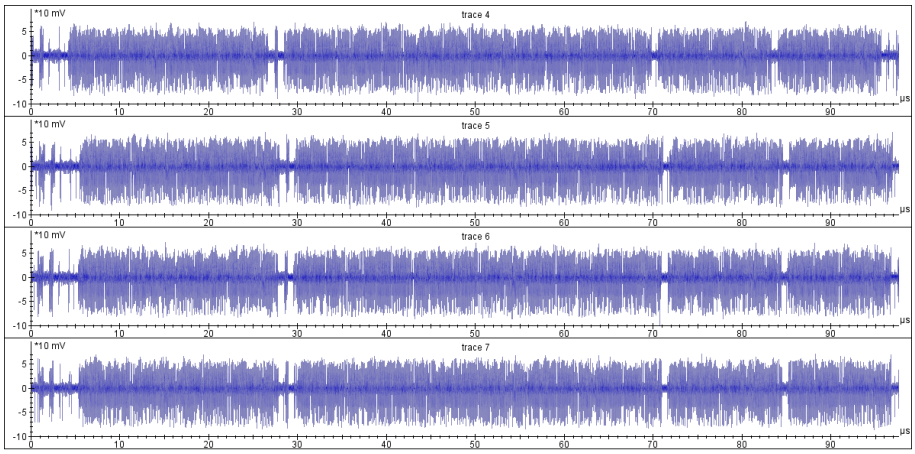
3.6.6.2. Algoritmos para *clustering*

Em SCA, métodos de análise baseados em aprendizado de máquina, tais como Support Vector Machines [Bartkewitz and Lemke-Rust 2013], Random Forests [Lerman et al. 2014], análise de séries temporais [Lerman et al. 2013] e análise nebulosa (*Fuzzy analysis*) [Saeedi and Kong 2014] tem sido recentemente empregados como uma alternativa ao método de *template attack* no contexto de ataques *profiled*, em particular quando a distribuição das amostras difere muito da distribuição gaussiana; e também no contexto de ataques não *profiled*, através da utilização de algoritmos de *clustering* não supervisionados.

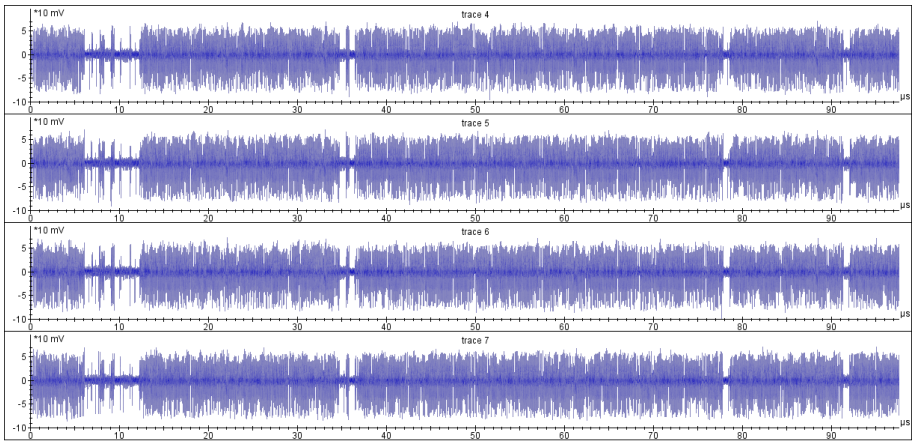
Dentre os algoritmos para *clustering* empregados com sucesso no contexto de ataques horizontais estão o K-Means [Forgy 1965, Lloyd 1982], Fuzzy K-Means [Dunn 1973] e o Expectation-Maximization (EM) [Dempster et al. 1977]. O K-Means é um algoritmo de *clustering* rígido, isto é, cada instância (uma amostra, no contexto de HCA) é atribuída (rotulada) a um único cluster. Fuzzy K-Means and EM, por outro lado, são algoritmos de *clustering* suaves (*soft*), pois tem como saída uma matriz de probabilidades de associação, onde a cada instância está associado o grau de vínculo desta com cada um dos clusters. Referimos o leitor aos livros sobre aprendizado de máquina [Alpaydin 2014, Witten and Frank 2011, Han et al. 2011, Bishop 2007, Duda et al. 2001] para descrições destes algoritmos e variantes.

3.6.6.3. Análise com chave conhecida

A análise com chave conhecida consiste em determinar os pontos de interesse, isto é, os índices de amostra, onde o vazamento é mais forte, com base no conhecimento da chave. Devido à necessidade de conhecimento do valor chave/escalar, tal análise é empregada somente na fase de teste do ataque HCA para determinar, por exemplo, quantos *traces* são

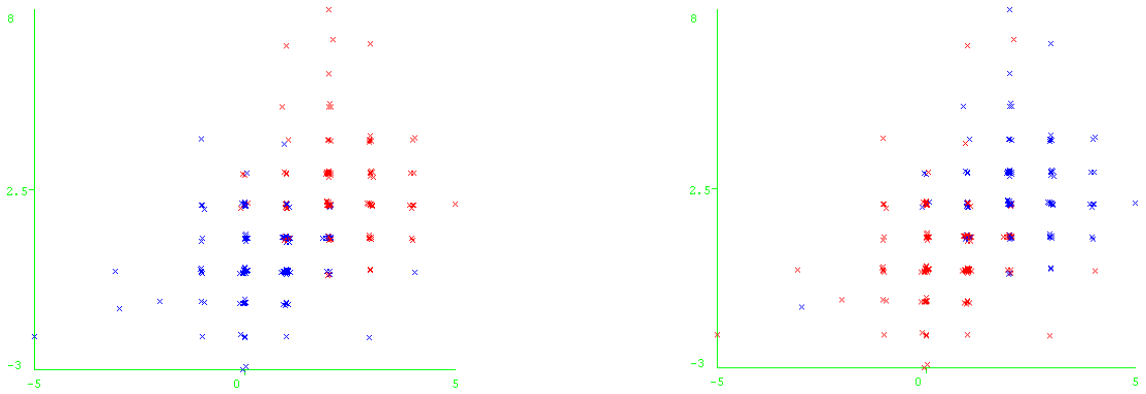


(a)



(b)

Figura 3.14: *Traces* EM de iterações da ECSM: (a) logo após segmentação, desalinhados; e (b) após alinhamento. *Traces* processados com Inspector [Riscure 2016].



(a) (b)

Figura 3.15: Visualização dos *clusters* obtidos pelo algoritmo EM em múltiplas dimensões/atributos. Cada dimensão corresponde ao índice de um POI, de um total de 20 POIs. Entretanto, para o propósito de visualização, apenas duas destas dimensões são exibidas. Em (a) estão os clusters obtidos pelo algoritmo; e em (b) os rótulos corretos. Amostras com a mesma cor foram rotuladas com o mesmo valor do bit.

necessários como entrada para a avaliação de vazamento de um dado dispositivo, de modo que os pontos de interesse obtidos por aquela correspondam aos previstos e sabidamente relevantes obtidos pela análise com chave conhecida.

Tal análise consiste em aplicar o algoritmo de clusterização no conjunto de amostras presentes em um dado índice de amostra (m), obtendo-se dois grupos de amostras: o primeiro grupo corresponde às amostras rotuladas com o valor b ($b \in \{0,1\}$, mas não se sabe se 0 ou 1) e o segundo grupo corresponde ao valor oposto $\bar{b}$ (Figure 3.15).

O conhecimento da chave é utilizado para determinar quantos bits tiveram o seu valor corretamente identificado no agrupamento. Como não se sabe o valor de b, isto é, o rótulo de cada grupo, o seguinte procedimento é adotado. Toma-se $b = 0$ e conta-se o número de bits corretamente identificados (n_c); o valor $n - n_c$ (n é o comprimento em bits do escalar) é então o número de bits corretamente identificados caso a rotulação esteja errada (isto é, o correto é $b = 1$). Finalmente, considera-se $max\{n_c, n - n_c\}$ o número de bits corretamente identificados.

O procedimento acima descrito é repetido para todos os índices de amostra, e o resultado são os pontos em que o vazamento de bits da chave é mais intenso. A ordenação destes pontos em ordem decrescente do número de bits corretamente identificados fornece os pontos de interesse.

3.6.6.4. Avaliação de vazamento

Técnicas de avaliação de vazamento determinam se um dispositivo criptográfico está vazando informação por canal lateral, com base em método estatístico e um modelo de va-

zamento. Em [Meynard et al. 2011] os autores testaram informação mútua (MIA)[4] como um método para localizar vazamento no domínio da frequência e, consequentemente, encontrar as bandas de frequência no *traces* de EM do RSA em que as diferenças entre quadrados e multiplicações são maiores. O Welch t-test é um outro método estatístico que pode ser empregado para este fim, p.ex., em metodologias como a TVLA (cf. Section 3.8.3.1). Os autores de [Mather et al. 2013] demonstraram como empregar t-test e MIA para localizar vazamento no domínio do tempo.

No escopo de ataques horizontais, tais métodos são empregados sem que haja conhecimento do valor da chave secreta ou de números aleatórios gerados e/ou usados pelo dispositivo. Portanto, são aplicáveis em um cenário realista onde o adversário não tem qualquer controle (escrita ou leitura) da chave do dispositivo ou outra informação secreta, em particular quando contramedidas como SR e CRR são aplicadas e não podem ser desabilitadas. Os pontos em que o vazamento é mais intenso, obtidos pela aplicação de um método para análise de vazamento, são tomados como pontos de interesse (POI). O valor das amostras em tais pontos são posteriormente utilizados na fase de ataque para recuperação da chave.

O método de análise de vazamento proposto em [Perin and Chmielewski 2015] demonstra como múltiplos *traces* podem ser combinados para a avaliação de vazamento, no contexto de ataques horizontais ao RSA. Tal método é baseado em *clustering* e funciona mesmo se o dispositivo empregar qualquer combinação das contramedidas clássicas aplicadas a implementações da exponenciação modular: exponent blinding, message or modulus randomization.[5]

3.6.6.5. Avaliação de vazamento baseada em *clustering*

Descrevemos nesta seção como o método de análise de vazamento de Perin e Chmielewski [Perin and Chmielewski 2015] pode ser adaptado a uma implementação do algoritmo Montgomery Ladder para multiplicação escalar. Supomos que deseja-se identificar o valor do bit do escalar utilizado em cada iteração do laço principal deste algoritmo através de algum vazamento direto ou indireto deste valor. Sejam n_0 e n_1 o número de bits 0 e 1 em um *trace*. A razão n_0/n_1 é aproximadamente constante, tendendo a 1, partindo da premissa de que os bits do escalar são gerados aleatoriamente. Devido à contramedida SR, o escalar efetivamente utilizado no laço da ECSM varia entre uma execução e outra da ECSM, e portanto difere de um *trace* para outro.

O método tem as seguintes premissas sobre o modelo de vazamento:

- **Premissa 1**: em um *trace* i, o valor médio para o conjunto de amostras em um índice m que correspondem às iterações cujo bit são 0 ou 1 são, respectivamente, $\mu_0^i + \gamma_0^i$ e $\mu_1^i + \gamma_1^i$, onde γ_k^i é um ruído aleatório com distribuição normal, $k = 0, 1$.
- **Premissa 2**: para todos os *traces* i, as médias μ_0^i e μ_1^i são constantes.

[4]Mutual Information Analysis.

[5]Exponent blinding e message randomization são equivalentes às contramedidas SR e CRR para ECC, respectivamente

Seja um *trace i* e as amostras localizadas em um índice *m* deste *trace*. A saída do algoritmo de *clustering* quando aplicado a este conjunto de amostras são dois centróides, $c_{0,m}$ e $c_{1,m}$ e dois clusters de amostras $\{g_{0,m}\}$ e $\{g_{1,m}\}$ contendo $p_{0,m}$ e $p_{1,m}$ elementos cada, respectivamente, tal que $p_{0,m} + p_{1,m} \approx n_0 + n_1$.

Temos então, para todo *trace i* e todo índice de amostra *m* deste *trace*, o seguinte conjunto de parâmetros: $c_{k,i}$, $\{g_{k,m}\}$, $p_{k,m} = |\{g_{k,m}\}|$ e $\sigma^2_{k,m} = Var(\{g_{k,m}\})$, para $k = 0, 1$. Este conjunto de parâmetros é utilizado como entrada para uma dentre as seguintes funções estatísticas, também conhecidas como *distinguishers*: diferença de médias (DoM), soma dos quadrados das diferenças (SOSD), soma dos quadrados dos t-values (SOST) e MIA.

Defina o seguintes parâmetros, para $k = 0, 1$:

$$r_{k,m} = \frac{min\{p_{k,m}, n_k\}}{max\{p_{k,m}, n_k\}}, \tag{4}$$

$$\beta_m = r_{0,m} \cdot r_{1,m}. \tag{5}$$

As funções distinguisher DoM, SOSD, SOST podem ser então definidas do seguinte modo[6]:

$$\begin{aligned}
\text{DoM} &: l_{\text{DOM},m} = |c_{0,m} - c_{1,m}| \\
\text{SOSD} &: l_{\text{SOSD},m} = |c_{0,m} - c_{1,m}|^2 \\
\text{SOST} &: l_{\text{SOST},m} = \left(\frac{|c_{0,m} - c_{1,m}|}{\sqrt{\frac{\sigma^2_{0,m}}{p_{0,m}} + \frac{\sigma^2_{1,m}}{p_{1,m}}}} \right)^2
\end{aligned}$$

A função *distinguisher* é aplicada em cada índice de amostra *m*, para cada *trace i*. O valor resultante é somado para todos os *traces*, a média é calculada e o valor é ajustado pelo coeficiente β_m, isto é, $\bar{l}_{\text{D},m} = \beta_m \frac{1}{N} \sum_{i=1}^{N} l^{(i)}_{\text{D},m}$, onde $D \in \{$ DoM,SOSD,SOST,MIA $\}$. O valor $\bar{l}_{\text{D},m}$ é, portanto, o valor estimado do vazamento no índice de amostra *m*, segundo a função *distinguisher D*.

A aplicação de algoritmos de *clustering* fornece uma estimativa para as médias $\mu_{k,m}$. Por causa do somatório usado na definição de $\bar{l}_{\text{D},m}$ e das premissas acima, o ruído $\gamma_{k,m}$ em cada amostra *m* é eliminado se o número de *traces* processados é suficientemente grande. A Figure 3.16 mostra o valor estimado do vazamento em cada índice de amostra para o *distinguisher* SOST aplicado a *traces* provenientes de uma implementação do algoritmo Montgomery Ladder.

[6]Referimos o leitor a [Perin and Chmielewski 2015] para a definição da função MIA neste caso.

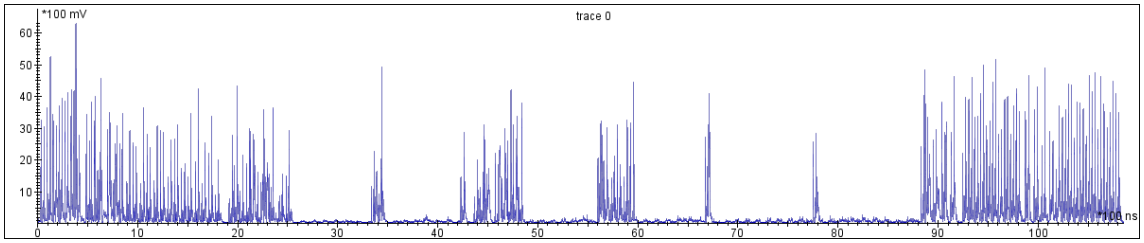


Figura 3.16: Estimativa de vazamento baseada em clusters utilizando o *distinguisher* SOST. Visualizado no Inspector [Riscure 2016].

3.6.6.6. Ataque para recuperação de chave

A etapa de ataque consiste na recuperação propriamente dita do escalar utilizando apenas um único *trace*, de uma única execução da ECSM. Esta etapa recebe como entradas o conjunto de *traces* e uma lista de n_{poi} POIs, e consiste nos seguintes passos:

Agrupamento. Para cada vetor de amostras em cada POI, é aplicado um dos algoritmos de *clustering* (K-Means, Fuzzy K-Means ou EM). O resultado são dois clusters para cada vetor de amostras, um correspondente ao bit 0 e o outro ao bit 1, e uma matriz de associação M_{nx2}, cujas entradas $m_{i,k}$ são a probabilidade de que a amostra i pertença à classe $k \in \{0,1\}$. Tal rotulamento das amostras pode ser visto como um escalar aproximado. A saída deste passo são n_{poi} escalares aproximados/candidatos, n_{poi} matrizes de associação.

Estimação do escalar final. Neste passo os escalares aproximados são combinados em um escalar final. Para tanto, um classificador estatístico (Majority Rule, Log-likelihood ou estimação de Bayes) é empregado.

Regra da Maioria. Este classificador simplesmente toma o rótulo da amostra de cada POI obtido pelo agrupamento (i.e., chute para o valor do bit) como um voto e então considera o valor que recebeu a maioria dos votos como o valor real do bit.

Log-likelihood e Estimação de Bayes. Estes são classificadores paramétricos; no contexto de SCA o modelo gaussiano é tipicamente adotado. Referimos o leitor a [Perin et al. 2014, Perin and Chmielewski 2015] para uma descrição do uso de tais classificadores no contexto de ataques horizontais baseados em *clustering*.

Cálculo do grau de confiança. Neste passo é calculado o grau de confiança (*confidence score*) no valor de cada bit recuperado. O grau de confiança é um número real entre 0 (total incerteza) e 1 (total certeza). O cálculo do grau de confiança depende de particularidades do ataque sendo realizado; exemplos: [Nascimento et al. 2016, Perin and Chmielewski 2015].

Cálculo da taxa de sucesso e nível de confiança. Se o valor do escalar é conhecido, isto é, se o ataque não está sendo aplicado na prática em um alvo real, mas sim sendo testado, então podem ser calculados também a taxa de sucesso e o nível de confiança do ataque.

A taxa de sucesso é simplesmente a média do número de bits do escalar que são corretamente recuperados quando o ataque é aplicado a um grande conjunto de *traces* de execuções completas da ECSM.

O nível de confiança é calculado da seguinte forma. Sejam C_{wrong} e C_{right} o conjunto de graus de confiança para os bits cujo valor recuperado está, respectivamente, errado ou certo, e $C_{\text{all}} = C_{\text{wrong}} \cup C_{\text{right}}$. Calcule $c_{\text{max,wrong}} = \max\{C_{\text{wrong}}\}$, $n_{\text{known\_wrong}} = |c \in C_{\text{all}}, c \leq c_{\text{max,wrong}}|$ e $n_{\text{known\_right}} = |C_{\text{all}}| - n_{\text{known\_wrong}}$. O nível de confiança é então definido por $\text{conf\_level} = n_{\text{known\_right}}/(n_{\text{known\_right}} + n_{\text{known\_wrong}})$ e representa a fração dos bits que foram corretamente recuperados com alta confiança, isto é, com confiança acima do limiar $c_{\text{max,wrong}}$. O nível de confiança indica a qualidade dos graus de confiança obtidos, isto é, quão bem eles permitem separar os bits do escalar cujo valor foi corretamente recuperado daqueles cujo valor recuperado está errado.

Ambos taxa de sucesso e nível de confiança são indicadores do sucesso de um ataque horizontal, e em última instância, se este é viável ou não, dadas as seguintes condições, dentre outras: qualidade e adequação do aparato de medição, SNR dos *traces* medidos, segmentação e alinhamento dos *traces*, qualidade dos pontos de interesse obtidos na etapa de avaliação de vazamento.

3.6.7. Ataques *template* versus Ataques horizontais

Precondições e limitações dos ataques baseados em *template*: Ataques baseados em *template* são os mais poderosos ataques do tipo SCA, segundo a Teoria da Informação [Chari et al. 2003]. No entanto, ataques baseados em *template* só podem ser realizados quando a contramedida SR não é aplicada ou quando pode ser desabilitada durante a fase de criação de *templates (profiling)*, caso contrário os *templates* não podem ser criados. Uma outra limitação deste tipo de ataque é de que dispositivos diferentes, mesmo que sejam do mesmo modelo, mesmo lote, etc., têm imperfeições únicas resultantes do processo de fabricação as quais resultam em diferenças no consumo de potência e radiação eletromagnética. Tais diferenças podem ser grandes o suficiente de modo que os *templates* gerados a partir dos *traces* provenientes do dispositivo de *profiling* não sejam bons modelos do vazamento observado no dispositivo alvo do ataque, assim reduzindo a taxa de sucesso do ataque [Elaabid and Guilley 2012].

Aplicabilidade. Até então estes ataques só foram demonstrados em CPUs embarcadas de 8, 16 e 32 bits, devido ao alto nível de SNR (*Signal-to-Noise Ratio*) que pode ser obtido na medição no consumo de potência e EM nestes dispositivos. Quando o SNR é baixo, além de haver pouco vazamento de dados (*data-leakage*) explorável do valor da chave ou valores intermediários derivados deste, o alinhamento dos *subtraces* torna-se também inviável, devido à inexistência de intervalos próximos da ocorrência da operação-alvo em que as amostras tem valores idênticos ou semelhantes em todos os *subtraces*.

3.7. Recuperação de chaves com erros em criptosistemas baseados no (EC)DLP

Devido ao ruído, vazamento de dado (por canal lateral) não relacionado à chave secreta, e outros aspectos que interferem com a análise por canal lateral (p.ex., desalinhamento,

clock jitter), o escalar final obtido por ataque SCA realizado a partir de um único *trace* provavelmente conterá erros, isto é, o valor de alguns dos bits recuperados estará incorreto.

Se a quantidade de bits incorretos (erros) é suficientemente pequena, então um ataque de força bruta pode ser viável, mesmo que não se saiba a localização de tais bits incorretos. A complexidade de tal busca, isto é, o número de escalares que devem ser testados até o escalar correto ser encontrado é $\binom{n}{s}2^s$, onde n é o comprimento do escalar em bits e s é número de erros. Considerando um escalar de $n = 256$ bits[7], o valor máximo de s para concluir tal busca em tempo aceitável é 6, o que significa que aproximadamente 2^{56} escalares precisam ser testados.

Se a quantidade de bits incorretos for maior, então o adversário necessita saber a localização dos possíveis bits incorretos no escalar recuperado para corrigi-los em tempo viável. Neste caso, a noção de bits suspeitos pode ser usada como referência para a seleção de bits do escalar com respeito a um ataque de força bruta. Um bit é considerado *suspeito* se o grau de confiança deste é menor do que o maior grau de confiança de qualquer bit falsamente/erroneamente identificado. Este último limiar (*threshold*) é determinado experimentalmente na fase de *profiling*.

Vamos supor que para um dado *trace* o escalar recuperado tenha $s = 54$ bits suspeitos, de um total de 254 bits. Para recuperar tal escalar, se este foi completamente aleatorizado pela contramedida SR, o adversário precisa realizar $O(2^{254})$ operações, no pior caso, o que geralmente não é prático.[8]

Para melhorar esta complexidade de força bruta, há duas opções. A primeira abordagem é tentar explorar a distribuição dos bits suspeitos nos conjuntos de bits recuperados incorretamente e corretamente (Figure 3.17). Enquanto há uma clara tendência dos bits incorretos terem um grau de confiança menor, a área da intersecção entre as duas distribuições é grande. Ainda assim, pode ser possível explorar esta tendência com um ataque de força bruta informado [Lange et al. 2015], priorizando os bits com os menores graus de confiança. Infelizmente esse ataque funciona bem se os bits contendo erros são adjacentes e este não é o caso no nosso contexto.

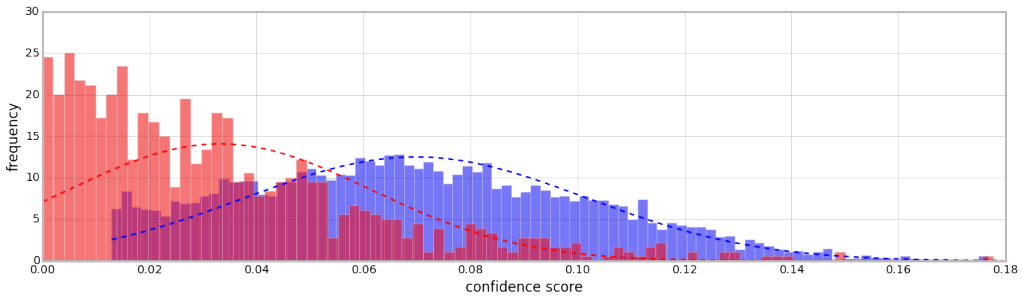


Figura 3.17: Distribuição de graus de confiança de bits suspeitos para um conjunto de 1000 *traces* de execuções completas da ECSM. Vermelho: bits recuperados incorretamente; azul: bits corretamente recuperados porém considerados suspeitos.

[7]comprimento típico do escalar para uma curva no nível de segurança de 128 bits

[8]Nesta subseção são consideradas apenas complexidades de pior caso.

3.7.1. Algoritmo de Gopalakrishnan, Theriault e Yao [Gopalakrishnan et al. 2007a]

Alternativamente, ou combinado com uma busca de força bruta informada, os autores de [Nascimento et al. 2016] aplicaram o segundo algoritmo de [Gopalakrishnan et al. 2007b] ao Montgomery Ladder, e este é o objeto desta seção. Tal algoritmo é baseado no paradigma de balanceamento entre tempo e memória (*time-memory trade-off*) e foi originalmente projetado para cadeias de cálculos de quadrados e multiplicação.

Descrevemos como o algoritmo funciona tomando como exemplo um escalar recuperado, contendo $s = 54$ bits suspeitos. Vamos representar os índices deste bits como uma lista ordenada em ordem decrescente: $i_s, \ldots i_1$, onde cada $i_j \in \{0, \ldots 254\}$ e $s \geq j \geq 1$; note que há 255 bits no total. Seja x o índice $i_{\lfloor \frac{s}{2}+1 \rfloor}$. Seja a o número representado pela cadeia de bits correspondente à parte esquerda do escalar (mais significativa) antes de x (incluindo i_x) e seja b o número correspondente à cadeia de bits da parte direita (menos significativa). Seja $y = 254 - i_x$ o comprimento da parte direita. Além disso, sabemos que $R = kP$, onde R é o ponto resultante, k é o escalar correto a ser recuperado e P é o ponto de entrada.

Então, claramente $R = kP = [a \cdot 2^{i_x} + b]P = [a]([2^{i_x}]P) + [b]P$. Se denotarmos $[2^{i_x}]P$ por H, então a equação acima se reduz a

$$R - [b]P = [a]H. \tag{6}$$

Podemos usar Equation (6) para verificar a corretude do chute. Seguindo [Gopalakrishnan et al. 2007b], usamos uma técnica de equilíbrio entre tempo e memória para acelerar uma busca exaustiva. Para cada "chute"do valor de a, computamos $[a]H$ e armazenamos todos os pares $(a, [a]H)$ em uma tabela. Então, ordenamos todos os pares pelo valor do ponto $A = [a]H$.

A seguir, "chutamos"um valor para b e computamos $Z = R - [b]P$. Se nosso chute para b estiver correto, então Z está presente na tabela, e o valor do escalar z correspondente (tal que $Z = [z]P$) é imediatamente obtido pois está na mesma linha desta tabela. Se Z está presente, então a busca termina e o escalar correto é $s^* = z||a$.

Como há aproximadamente $2^{\frac{s}{2}}$ possibilidades para a e b, a complexidade de tempo é $O(2^{\frac{s}{2}})$ operações. Como há $2^{\frac{s}{2}}$ possibilidades para a, a tabela tem aquele número de entradas e a complexidade de espaço é $O(2^{\frac{s}{2}})$. Desta forma nós limitamos a complexidade de tempo para $O(2^{\frac{s}{2}})$ (cf. [Gopalakrishnan et al. 2007b] para uma análise de complexidade detalhada), a qual é da ordem de 2^{54} para o escalar de exemplo.

Dado um conjunto de *traces* provenientes de medições da ECSM no dispositivo alvo, não sabemos qual *trace* contém o menor número de bits suspeitos pois não sabemos o grau de confiança máximo de um bit incorretamente identificado. No entanto, este valor pode ser determinado atacando alguns *traces* para os quais o escalar aleatorizado é conhecido.

No trabalho [Nascimento et al. 2016], os autores determinaram que, para o dispositivo e implementação em software da curva *Curve25519* considerados, o número 54 de bits suspeitos cobre todos os bits identificados incorretamente para pelo menos um

trace dentro de um conjunto de $m = 100$ *traces*. Logo, o ataque completo para recuperar o valor correto do escalar funciona do seguinte modo: o algoritmo acima descrito é executado para cada um dos *m traces*, em sequência; o ataque pára quando o algoritmo encontra um ponto na tabela, e portanto o escalar foi determinado; se nenhum dos pontos foi encontrado na tabela, para nenhum dos *traces*, então o ataque falhou.

Como o algoritmo é executado m vezes, a complexidade do ataque completo é multiplicada por m. A complexidade totaliza $O(n \cdot 2^{\frac{s}{2}})$ operações e $O(n \cdot 2^{\frac{s}{2}})$ em memória. Para o ataque da seção anterior, isso corresponde a $O\left(m \cdot 2^{27}\right) = O\left(2^{32}\right)$ operações.

3.8. Ferramentas

3.8.1. Ferramentas para verificação de tempo constante

Os primeiros métodos para verificação formal de contramedidas para canais laterais de tempo foram construídos a partir de análise estática. [Molnar et al. 2005] propõem métodos para detetar canais laterais de controle de fluxo e transformar código fonte em C para eliminar as vulnerabilidades, abrangendo ataques de tempo e tratamento de erros. [Coppens et al. 2009] modificam um compilador para converter comandos condicionais de forma que o código *Assembly* resultante não mais tenha comportamento no tempo dependente dos dados processados. [Lux and Starostin 2011] propõem uma ferramenta para detetar potenciais canais laterais em implementações em Java de algoritmos criptográficos, baseados em anotações do programador. Em [Köpf et al. 2012], os autores propõem um método novo baseado em limitantes superiores automaticamente derivados na quantidade de informação sobre a entrada que o adversário consegue extrair de um programa a partir da observação do comportamento da *cache* durante a execução. [Doychev et al. 2015] propõem métodos para calcular aproximações precisas da quantidade de informação vazada que podem ser observadas nos canais laterais a seguir: transições de estado na *cache*, traços de acertos e erros, e tempo de execução. Os autores também sugerem provas formais de segurança para contramedidas como pré-carga de endereços e padrão de acesso independente de dados.

Ainda considerando ataques de tempo, outras abordagens para verificação foram propostas recentemente. O trabalho [Almeida et al. 2013] considera as políticas em alto nível adotadas na implementação da biblioteca NaCl: ausência de desvios condicionais e endereçamento de vetores dependentes de dados; formalizam as políticas e propõem um método de verificação formal baseado em auto-composição, demonstrando-o pela aplicação no código *Assembly* otimizado de algumas funções da biblioteca. [Langley 2012] propõe um método de análise dinâmico baseado no módulo `memcheck` da ferramenta *Valgrind*, que amplifica sua capacidade para reconhecer dados não-inicializados na granularidade de bits. A ferramenta *FlowTracker*[9] [Rodrigues et al. 2016] foi proposta recentemente para verificar comportamento constante de código compilado pela análise estática de fluxo de informação na representação intermediária LLVM. As vantagens da ferramenta são a facilidade de descrição das interfaces e a baixa intrusão, já que nenhuma alteração é necessária no código. CT-Verif [Almeida et al. 2016] é uma ferramenta seguindo uma abordagem similar que fornece garantias formais adicionais, mas exige alteração do código pelo verifica-

[9] `http://cuda.dcc.ufmg.br/flowtracker/`

dor.

3.8.2. Ferramentas para verificação de implementações contra ataques de potência

Verificação formal de implementações em software de algoritmos criptográficos contra análise de potência é um assunto que tem sido pesquisado recentemente. Por exemplo, Maggi et al. [MAGGI 2013, Agosta et al. 2013] propuseram um método baseado em análise de fluxo de dados para identificar dependências entre as instruções executadas e dados secretos. O método é implementado em um compilador LLVM como uma passada especializada operando no nível de representação intermediária, portanto ela é agnóstica em arquitetura, suportando quaisquer das arquiteturas de computador suportadas por LLVM. A ferramenta automaticamente instancia contramedidas de masking à implementação.

Mais recentemente, Bayrak et al [Bayrak et al. 2013, Bayrak et al. 2014] mostraram como reduzir o problema de verificação da resistência de uma implementação a vazamento por canais de potência a um conjunto de problemas SAT, os quais podem ser eficientemente tratados pelos resolvedores SAT atuais. Tal método, em princípio, trata das limitações da abordagem baseada em análise de fluxo de informação.

Resolvedores baseados em teorias do módulo de satisfabilidade (SMT) foram aplicados por Eldib et al [Eldib and Wang 2014, Eldib et al. 2014b, Eldib et al. 2014a, Eldib et al. 2014c] a este problema. Estes últimos também propuseram métodos para a aplicação automatizada de contramedidas.

3.8.3. Métodos empíricos para análise de vazamentos

Avaliações de segurança de dispositivos criptográficos com respeito a canais laterais compreende duas fases: *medição* e *análise*. A saída ou resultado de tal avaliação deve ser Falhou (*Fail*) ou Passou (*Pass*). O resultado de tal avaliação deve ser interpretado segundo as restrições do processo de avaliação, tais como a acurácia do equipamento de teste, expertise técnica dos avaliadores e tempo disponível para a avaliação. A medição dos *traces* e suas limitações deve ser levada em consideração, caso contrário a fase posterior, de análise, pode ser prejudicada ou invalidada, resultando em falsos positivos, ou pior, em falsos negativos.

As metodologias de avaliação atuais (p.ex., Common Criteria [Criteria 2014]) consistem na realização de uma bateria de ataques por canais laterais conhecidos contra o dispositivo sob teste (DUT) [10], numa tentativa de recuperar a chave. Apesar disto, a rápida evolução das técnicas de análise por canais laterais atuais propostas na literatura tem exigido um nível crescente de expertise dos testadores e um aumento no tempo requerido para avaliação. Mesmo quando todas as tentativas de ataque falham, vazamentos residuais no canal lateral avaliado podem ainda estar presentes, o que pode revelar novos caminhos de ataque (*attack paths*) para um adversário.

[10] Device under test.

3.8.3.1. Test Vector Leakage Assessment (TVLA)

Por estas razões o NIST organizou um workshop em 2011 [NIST 2011] para encorajar o desenvolvimento de métodos de teste, métricas e ferramentas para avaliação da eficácia de mitigações contra ataques não-invasivos a módulos criptográficos.

Nesse workshop, a CRI[11] propôs a metodologia *Test Vector Leakage Assessment* (TVLA) [Goodwill et al. 2011] com o propósito de resolver os problemas acima. Os autores dessa metodologia consideram que duas figuras de mérito são importantes: a eficácia, no sentido de que ela é reprodutível e é um indicador confiável da resistência atingida pelo dispositivo; e a relação custo-benefício, isto é, na palavra dos autores, "validar um nível moderado de resistência (p.ex., FIPS 140 nível 3 ou 4) não deve requerer uma quantidade excessiva de tempo de teste por algoritmo ou de habilidade do operador de teste" [Goodwill et al. 2011]. A abordagem da metodologia TVLA difere fundamentalmente das estratégias de avaliação focadas em ataque atualmente empregadas, adotando uma estratégia caixa-preta com foco na detecção de vazamento.

A fase de medição na TVLA é baseada na aquisição de *trace* de canal lateral quando vetores de teste padronizados são fornecidos como entrada para a implementação sob teste, e estabelece requisitos para os equipamentos e setup de medição, alinhamento e pré-processamento dos *traces*.

A fase de análise compreende teste de hipótese estatístico, mais especificamente, o t-test de Welch, o qual é capaz de detetar diferentes tipos de vazamento e permite ao analista identificar pontos no tempo que merecem investigação adicional.

A metodologia TVLA até então foi aplicada aimplementações do AES em hardware e software [Goodwill et al. 2011, Cooper et al. 2013, Mather et al. 2013], e implementações em software do RSA [Witteman et al. 2011a] e ECC (multiplicação escalar com base variável) [Nascimento et al. 2015]. Bem recentemente, os autores da TVLA detalharam mais a aplicação da metodologia ao RSA e como adaptá-la aos esquemas ECDSA e ECDH. [Tunstall and Goodwill 2016].

3.8.3.2. Outras metodologias para análise de vazamentos

Outras metodologias, baseadas em informação mútua contínua [Chothia and Guha 2011] e discreta [Chatzikokolakis et al. 2010] também foram propostas. Oswald et al [Mather et al. 2013] analisaram as metodologias [Goodwill et al. 2011], [Chatzikokolakis et al. 2010] e [Chothia and Guha 2011], e concluiu que elas têm poder estatístico similar. O trabalho recente de Schneider e Moradi [Schneider and Moradi 2016] aborda como realizar o teste t (em [Goodwill et al. 2011]) em ordens mais altas e como estendê-lo para o contexto multivariado.

[11] Empresa "Cryptography Research", atualmente incorporada à Rambus.

3.8.3.3. Aplicação da metodologia TVLA para ECC

Nesta subseção mostramos a aplicação da metodologia TVLA a uma implementação para microcontrolador AVR do esquema ECDH utilizando a curva *Curve25519* e o algoritmo Montgomery Ladder para ECSM protegida com a contramedida de aleatorização de coordenadas projetivas (CR) [Nascimento et al. 2015]. Especificamente, os conjuntos de vetores de teste na Table 3.4) foram selecionados para serem usados para a fase de medição de consumo de potência, os quais cobrem casos normais e especiais da aritmética de corpo finito e de grupo. A Table 3.5 mostra categorias de valores especiais usados nos conjuntos 4 e 5 para a função *compute shared secret* do esquema ECDH-Curve25519.

Tabela 3.4: Conjuntos de vetores de teste para análise de vazamento SPA (k é um escalar secreto e P é um ponto).

# Conj.	Propriedades	Descrição
1	k constante, P constante	Este é o baseline. Os testes comparam os *traces* dos outros conjuntos contra este.
2	k constante, P varia	Meta é detetar relações sistemáticas entre consumo de potência e o valor de P.
3	k varia, P constante	Meta é detetar relações sistemáticas entre consumo de potência e o valor de k.
4	k constante, P especial	Casos de borda dos algoritmos utilizados.
5	k especial, P constante	Casos de borda dos algoritmos utilizados.

Tabela 3.5: Categorias de valores especiais para n e q na função *compute shared secret* em ECDH-Curve25519 (q é um ponto codificado, n é um escalar codificado e l é a ordem do subgrupo).

Cat. #	Propriedades
1	$q \in \{0, 1, ..., 1023\}$
2	$q \in \{p_{25519} - 1, ..., p_{25519} - 1024\}$
3	$n \in \{0, ..., 1023\}$
4	$n \in \{l - 1, ..., l - 1024\}$
5	q tem um alto peso de Hamming (≥ 230)
6	q tem um baixo peso de Hamming (≤ 25)

Fase de aquisição. Os autores [Nascimento et al. 2015] capturaram 200 *traces* de potência para cada um dos conjuntos de vetores de teste, totalizando 1000 *traces*. A fase de análise de vazamento é idêntica à proposta na metodologia TVLA para o RSA [Witteman et al. 2011a], e é conduzida da seguinte forma. Sejam $\{DS_1, \ldots, DS_5\}$ os conjuntos de *traces* de potência correspondentes aos conjuntos de vetores de teste selecionados.

O teste completo consiste em executar os testes pareados descritos em [Witteman et al. 2011a] para cada um dos seguintes pares de datasets: $\{(DS_1, DS_2)$, $\ldots$, $(DS_1, DS_5)\}$. Se qualquer um dos testes anteriores falha, então o resultado da avaliação da implementação é *FALHOU*. Caso contrário, o resultado é *PASSOU*. Nós escolhemos o limiar de confiança $C = 4.5$, o mesmo valor usado na metodologia TVLA para o RSA [Witteman et al. 2011a].

Fase de análise SPA. A Figure 3.18 mostra a estatística t para um pequeno intervalo de índices de amostra [12] (i.e., instantes de tempo), para uma execução do teste t para grupo A $(S_{A,1}, S_{A,2})$ de vetores selecionados de DS_1 e DS_3, e o mesmo teste executado sobre o grupo (independente) B $(S_{B,1}, S_{B,2})$. [13]

A estatística t para o grupo A está acima do limiar $C = 4.5$ em um instante de tempo, significando uma possível dependência forte entre o consumo de potência e o valor da chave naquele instante. Mas, como este evento não ocorreu ao mesmo tempo e na mesma direção para o grupo B, ele é considerado um falso positivo pela metodologia e assim é descartado.

Os resultados de teste para cada par de conjuntos de vetores de teste $\{(DS_1, DS_2), \ldots, (DS_1, DS_5)\}$ mostrou que em poucos instantes de tempo o valor da estatística t para um dos grupos esteve acima de 4.5 ou abaixo de -4.5, mas nunca para ambos os grupos ao mesmo tempo. Portanto, pode-se concluir que a implementação passou a avaliação de vazamento SPA segundo a metodologia TVLA.

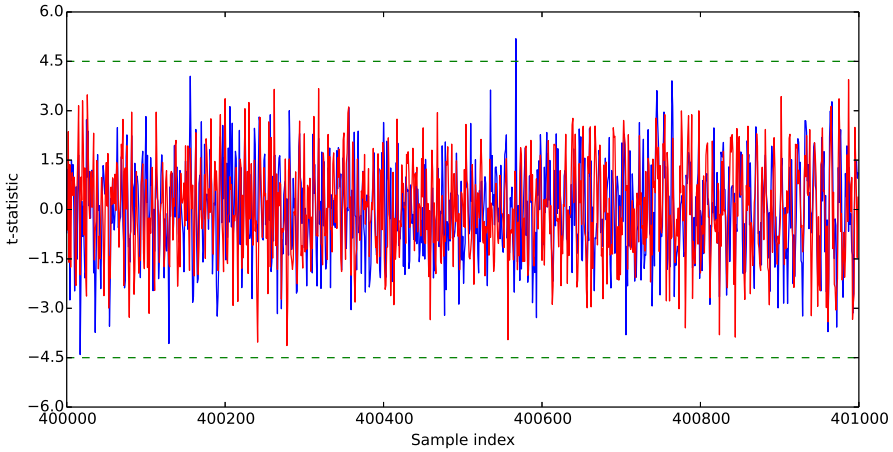


Figura 3.18: t-estatística versus índice de amostra para o experimento comparando DS_1 e DS_3, para dois grupos de *traces* independentes; grupo A (azul) e grupo B (vermelho).

3.8.4. Ferramentas para recuperação de chaves com erros em criptossistemas baseados no (EC)DLP

Lange et al [Lange et al. 2015] propuseram um algoritmo chamado ε-*enumeration* para computação do *rank* de uma chave usando como entrada as probabilidades/graus de confiança obtidos do ataque SCA juntamente com uma variação do algoritmo kangaroo de Pollard. O código fonte da implementação deste algoritmo foi disponibilizada pelos autores [Vredendaal 2014].

Os autores de [Nascimento et al. 2016] disponibilizaram o código fonte da implementação do algoritmo de Gopalakrishnan et al [Gopalakrishnan et al. 2007b]

[12] Este intervalo de tempo foi selecionado porque ele ilustra um intervalo onde os valores da estatística t são altos comparados com os os instantes de tempo

[13] Os grupos A e B são uma partição dos conjuntos de vetores de teste DS_1 e DS_3: $(S_{A,1} \subset DS_1,\ \ S_{A,2} \subset DS_3)$ e $(S_{B,1} = DS_1 \setminus S_{A,1},\ \ S_{B,2} = DS_3 \setminus S_{A,2})$.

(cf. Section 3.7.1) para recuperação de chaves com erros [Nascimento 2016]. Tal implementação foi otimizada para ECSM por Montgomery Ladder na Curve25519.

Referências

[AES 2001] (2001). FIPS 197 - Advanced Encryption Standard (AES). Technical report, National Institute of Standards and Technology.

[SHA 2012] (2012). FIPS 180-4 - Secure Hash Standard (SHA). Technical report, National Institute of Standards and Technology.

[Aciiçmez et al. 2007] Aciiçmez, O., Koç, c. K., and Seifert, J.-P. (2007). On the power of simple branch prediction analysis. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 312–320, New York, NY, USA. ACM.

[Agosta et al. 2013] Agosta, G., Barenghi, A., Maggi, M., and Pelosi, G. (2013). Compiler-based Side Channel Vulnerability Analysis and Optimized Countermeasures Application. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 81:1—-81:6, New York, NY, USA. ACM.

[Akishita and Takagi 2003] Akishita, T. and Takagi, T. (2003). Zero-Value Point Attacks on Elliptic Curve Cryptosystem. pages 218–233.

[AlFardan and Paterson 2013] AlFardan, N. J. and Paterson, K. G. (2013). Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 526–540. IEEE Computer Society.

[Almeida et al. 2016] Almeida, J. B., Barbosa, M., Barthe, G., Dupressoir, F., and Emmi, M. (2016). Verifying constant-time implementations. In Holz, T. and Savage, S., editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 53–70. USENIX Association.

[Almeida et al. 2013] Almeida, J. B., Barbosa, M., Pinto, J. S., and Vieira, B. (2013). Formal verification of side-channel countermeasures using self-composition. *Sci. Comput. Program.*, 78(7):796–812.

[Alpaydin 2014] Alpaydin, E. (2014). *Introduction to machine learning*. MIT press.

[Bartkewitz and Lemke-Rust 2013] Bartkewitz, T. and Lemke-Rust, K. (2013). *Efficient Template Attacks Based on Probabilistic Multi-class Support Vector Machines*, pages 263–276. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Batina et al. 2014] Batina, L., Chmielewski, L., Papachristodoulou, L., Schwabe, P., and Tunstall, M. (2014). Online Template Attacks. In *Progress in Cryptology – INDOCRYPT 2014*, volume 1977, pages 21–36.

[Bauer and Jaulmes 2013] Bauer, A. and Jaulmes, É. (2013). Correlation analysis against protected SFM implementations of RSA. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8250 LNCS:98–115.

[Bauer et al. 2013] Bauer, A., Jaulmes, É., Prouff, E., and Wild, J. (2013). Horizontal and Vertical Side-Channel Attacks against Secure {RSA} Implementations. In *CT-RSA*, pages 1–17.

[Bayrak et al. 2014] Bayrak, A., Regazzoni, F., Novo Bruna, D., Brisk, P., Standaert, F., and Ienne, P. (2014). Automatic Application of Power Analysis Countermeasures. *Computers, IEEE Transactions on*, PP(99):1.

[Bayrak et al. 2013] Bayrak, A. G., Regazzoni, F., Novo, D., and Ienne, P. (2013). Sleuth: automated verification of software power analysis countermeasures. In *Cryptographic Hardware and Embedded Systems-CHES 2013*, pages 293–310. Springer.

[Bernstein 2004] Bernstein, D. J. (2004). Cache-timing attacks on AES. URL: `http://cr.yp.to/papers.html#cachetiming`.

[Bernstein 2005] Bernstein, D. J. (2005). Chacha20, a variant of salsa20. `https://cr.yp.to/chacha/chacha-20080120.pdf`.

[Bernstein et al. 2012] Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., and Yang, B.-Y. (2012). High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89.

[Bertoni et al. 2008] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. (2008). On the indifferentiability of the sponge construction. In Smart, N. P., editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer. `http://sponge.noekeon.org/`.

[Biham 1997] Biham, E. (1997). A fast new DES implementation in software. In Biham, E., editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer.

[Bishop 2007] Bishop, C. (2007). *Pattern Recognition and Machine Learning*. Springer.

[Bonneau and Mironov 2006] Bonneau, J. and Mironov, I. (2006). Cache-Collision Timing Attacks Against AES. In Goubin, L. and Matsui, M., editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 201–215. Springer.

[Brown et al. 2001] Brown, M., Hankerson, D., López, J., and Menezes, A. (2001). *Software Implementation of the NIST Elliptic Curves Over Prime Fields*, pages 250–265. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Brumley and Boneh 2003] Brumley, D. and Boneh, D. (2003). Remote timing attacks are practical. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 1–1, Berkeley, CA, USA. USENIX Association.

[Chari et al. 2003] Chari, S., Rao, J. R., and Rohatgi, P. (2003). Template Attacks. *CHES 2002*, 2523:13–28.

[Chatzikokolakis et al. 2010] Chatzikokolakis, K., Chothia, T., and Guha, A. (2010). Statistical Measurement of Information Leakage. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *LNCS*, pages 390–404. Springer.

[Chothia and Guha 2011] Chothia, T. and Guha, A. (2011). A statistical test for information leaks using continuous mutual information. *Proceedings - IEEE Computer Security Foundations Symposium*, pages 177–190.

[Ciet and Joye 2003] Ciet, M. and Joye, M. (2003). ({Virtually}) Free Randomization Techniques for Elliptic Curve Cryptography. In *Information and Communications Security*, pages 348–359.

[Clavier et al. 2012] Clavier, C., Feix, B., Gagnerot, G., Giraud, C., Roussellet, M., and Verneuil, V. (2012). {ROSETTA} for Single Trace Analysis. pages 140–155.

[Clavier et al. 2010] Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., and Verneuil, V. (2010). *Horizontal Correlation Analysis on Exponentiation*, pages 46–61. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Clavier and Joye 2001] Clavier, C. and Joye, M. (2001). Universal Exponentiation Algorithm - A First Step towards Provable SPA-Resistance. In Koç, Ç., Naccache, D., and Paar, C., editors, *Cryptographic Hardware and Embedded Systems - CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 300–308. Springer Berlin / Heidelberg.

[Cooper et al. 2013] Cooper, J., Demulder, E., Goodwill, G., Jaffe, J., and Kenworthy, G. (2013). Test Vector Leakage Assessment (TVLA) methodology in practice (Extended Abstract). Technical report, Cryptography Research Inc.

[Coppens et al. 2009] Coppens, B., Verbauwhede, I., Bosschere, K. D., and Sutter, B. D. (2009). Practical mitigations for timing-based side-channel attacks on modern x86 processors. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, pages 45–60. IEEE Computer Society.

[Coron 1999] Coron, J.-S. (1999). Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems*, pages 292–302. Springer.

[Costello and Longa 2015] Costello, C. and Longa, P. (2015). Fourq: four-dimensional decompositions on a q-curve over the mersenne prime. *IACR Cryptology ePrint Archive*, 2015:565.

[Criteria 2014] Criteria, C. (2014). Common Criteria v3.1. Technical report, Common Criteria.

[Damgård 1989] Damgård, I. (1989). A design principle for hash functions. In Brassard, G., editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer.

[Danger et al. 2013] Danger, J.-L., Guilley, S., Hoogvorst, P., Murdica, C., and Naccache, D. (2013). A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards. *Journal of Cryptographic Engineering*, 3(4):241–265.

[Dempster et al. 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

[Denis 2006] Denis, T. S. (2006). *BigNum Math: Implementing Cryptographic Multiple Precision Arithmetic*. Syngress Publishing.

[Dinur and Shamir 2012] Dinur, I. and Shamir, A. (2012). Applying cube attacks to stream ciphers in realistic scenarios. *Cryptography and Communications*, 4(3-4):217–232.

[Doychev et al. 2015] Doychev, G., Köpf, B., Mauborgne, L., and Reineke, J. (2015). Cacheaudit: A tool for the static analysis of cache side channels. *ACM Trans. Inf. Syst. Secur.*, 18(1):4.

[Duda et al. 2001] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern classification*. John Wiley & Sons.

[Düll et al. 2015] Düll, M., Haase, B., Hinterwälder, G., Hutter, M., Paar, C., Sánchez, A. H., and Schwabe, P. (2015). High-speed curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. *Des. Codes Cryptography*, 77(2-3):493–514.

[Dunn 1973] Dunn, J. C. (1973). A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters.

[Elaabid and Guilley 2012] Elaabid, M. and Guilley, S. (2012). Portability of templates. *Journal of Cryptographic Engineering*, pages 63–74.

[Eldib and Wang 2014] Eldib, H. and Wang, C. (2014). Synthesis of Masking Countermeasures against Side Channel Attacks. In Biere, A. and Bloem, R., editors, *Computer Aided Verification SE - 8*, volume 8559 of *Lecture Notes in Computer Science*, pages 114–130. Springer International Publishing.

[Eldib et al. 2014a] Eldib, H., Wang, C., and Schaumont, P. (2014a). SMT-Based Verification of Software Countermeasures against Side-Channel Attacks. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 62–77. Springer.

[Eldib et al. 2014b] Eldib, H., Wang, C., Taha, M., and Schaumont, P. (2014b). QMS: Evaluating the Side-Channel Resistance of Masked Software from Source Code. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, pages 1–6. ACM.

[Eldib et al. 2014c] Eldib, H., Wang, C., Taha, M., and Schaumont, P. (2014c). SC Sniffer - Side-channel leak sniffer.

[Forgy 1965] Forgy, E. W. (1965). Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769.

[Fouque and Valette 2003] Fouque, P. and Valette, F. (2003). The doubling attack - *Why Upwards Is Better than Downwards*. In Walter, C. D., Koç, Ç. K., and Paar, C., editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 269–280. Springer.

[Gierlichs et al. 2009] Gierlichs, B., Batina, L., Preneel, B., and Verbauwhede, I. (2009). Revisiting higher-order DPA attacks: Multivariate mutual information analysis. *IACR Cryptology ePrint Archive*, 2009:228.

[Goodwill et al. 2011] Goodwill, G., Jun, B., Jaffe, J., and Rohatgi, P. (2011). A testing methodology for side channel resistance validation. Technical report, CRI.

[Gopalakrishnan et al. 2007a] Gopalakrishnan, K., Thériault, N., and Yao, C. Z. (2007a). Solving Discrete Logarithms from Partial Knowledge of the Key. In *INDOCRYPT*, pages 224–237.

[Gopalakrishnan et al. 2007b] Gopalakrishnan, K., Thériault, N., and Yao, C. Z. (2007b). Solving discrete logarithms from partial knowledge of the key. In K. Srinathan, C. Pandu Rangan, M. Y., editor, *Progress in Cryptology – INDOCRYPT 2007*, volume 4859 of *LNCS*, pages 224–237. Springer.

[Goubin 2003] Goubin, L. (2003). A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. pages 199–210.

[Hamburg 2009] Hamburg, M. (2009). Accelerating AES with Vector Permute Instructions. In Clavier, C. and Gaj, K., editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 18–32. Springer.

[Han et al. 2011] Han, J., Pei, J., and Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

[Hankerson et al. 2003] Hankerson, D., Menezes, A. J., and Vanstone, S. (2003). *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Hankerson et al. 2004] Hankerson, D., Vanstone, S., and Menezes, A. J. (2004). *Guide to elliptic curve cryptography*. Springer.

[Hennessy and Patterson 2002] Hennessy, J. L. and Patterson, D. A. (2002). *Computer Architecture: A Quantitative Approach (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann.

[Heyszl et al. 2014] Heyszl, J., Ibing, A., Mangard, S., Santis, F., and Sigl, G. (2014). Clustering Algorithms for Non-profiled Single-Execution Attacks on Exponentiations. In Francillon, A. and Rohatgi, P., editors, *CARDIS*, pages 79–93, Cham. Springer International Publishing.

[Ishai et al. 2003] Ishai, Y., Sahai, A., and Wagner, D. (2003). Private circuits: Securing hardware against probing attacks. In Boneh, D., editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer.

[Jean-Pierre et al. 2006] Jean-Pierre, O. A., pierre Seifert, J., and Çetin Kaya Koç (2006). Predicting secret keys via branch prediction. In *in Cryptology – CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007*, pages 225–242. Springer-Verlag.

[Jr. et al. 2016] Jr., M. A. S., Silva, M. V., Alves, R. C., and Shibata, T. K. (2016). Lightweight and escrow-less authenticated key agreement for the internet of things. *Computer Communications*, pages –.

[Käsper and Schwabe 2009] Käsper, E. and Schwabe, P. (2009). Faster and timing-attack resistant AES-GCM. In Clavier, C. and Gaj, K., editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 1–17. Springer.

[Koblitz 1987] Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209.

[Kocher 1996] Kocher, P. C. (1996). Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Koblitz, N., editor, *16th Annual International Cryptology Conference (CRYPTO 1996)*, volume 1109 of *LNCS*, pages 104–113. Springer.

[Kocher et al. 1999] Kocher, P. C., Jaffe, J., and Jun, B. (1999). Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK. Springer-Verlag.

[Köpf et al. 2012] Köpf, B., Mauborgne, L., and Ochoa, M. (2012). Automatic quantification of cache side-channels. In Madhusudan, P. and Seshia, S. A., editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 564–580. Springer.

[Lange et al. 2015] Lange, T., van Vredendaal, C., and Wakker, M. (2015). Kangaroos in side-channel attacks. In Joye, M. and Moradi, A., editors, *Smart Card Research and Advanced Applications*, volume 8968 of *LNCS*, pages 104–121. Springer.

[Langley 2012] Langley, A. (2012). Ctgrind: Checking that functions are constant time with Valgrind. `https://github.com/agl/ctgrind`.

[Lerman et al. 2013] Lerman, L., Bontempi, G., Ben Taieb, S., and Markowitch, O. (2013). *A Time Series Approach for Profiling Attack*, pages 75–94. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Lerman et al. 2014] Lerman, L., Bontempi, G., and Markowitch, O. (2014). Power analysis attack: an approach based on machine learning. *International Journal of Applied Cryptography*, 3(2):97–115.

[Lloyd 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137.

[Luby and Rackoff 1988] Luby, M. and Rackoff, C. (1988). How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386.

[Lux and Starostin 2011] Lux, A. and Starostin, A. (2011). A tool for static detection of timing channels in java. *J. Cryptographic Engineering*, 1(4):303–313.

[MAGGI 2013] MAGGI, M. (2013). Automated side channel vulnerability detection and countermeasure application via compiler based techniques.

[Mather et al. 2013] Mather, L., Oswald, E., Bandenburg, J., and Wójcik, M. (2013). Does My Device Leak Information? An a priori Statistical Power Analysis of Leakage Detection Tests. In *Advances in Cryptology-ASIACRYPT 2013*, pages 486–505. Springer.

[McEvoy et al. 2007] McEvoy, R. P., Tunstall, M., Murphy, C. C., and Marnane, W. P. (2007). Differential power analysis of HMAC based on sha-2, and countermeasures. In Kim, S., Yung, M., and Lee, H., editors, *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*, volume 4867 of *Lecture Notes in Computer Science*, pages 317–332. Springer.

[McGrew and Viega 2004] McGrew, D. A. and Viega, J. (2004). The security and performance of the galois/counter mode (GCM) of operation. In Canteaut, A. and Viswanathan, K., editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer.

[Merkle 1979] Merkle, R. C. (1979). *Secrecy, authentication, and public key systems.* PhD thesis, Stanford University.

[Meynard et al. 2011] Meynard, O., Réal, D., Flament, F., Guilley, S., Homma, N., and Danger, J. L. (2011). Enhancement of simple electro-magnetic attacks by pre-characterization in frequency domain and demodulation techniques. In *2011 Design, Automation Test in Europe*, pages 1–6.

[Miller 1986] Miller, V. S. (1986). Use of elliptic curves in cryptography. In Williams, H. C., editor, *Proceedings of CRYPTO 85*, pages 417–426. Springer. Lecture Notes in Computer Science No. 218.

[Molnar et al. 2005] Molnar, D., Piotrowski, M., Schultz, D., and Wagner, D. (2005). The program counter security model: Automatic detection and removal of control-flow side channel attacks. In Won, D. and Kim, S., editors, *Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers*, volume 3935 of *Lecture Notes in Computer Science*, pages 156–168. Springer.

[Murdica et al. 2012] Murdica, C., Guilley, S., Danger, J.-L., Hoogvorst, P., and Naccache, D. (2012). Same Values Power Analysis Using Special Points on Elliptic Curves. In Schindler, W. and Huss, S., editors, *Constructive Side-Channel Analysis and Secure Design*, volume 7275 of *Lecture Notes in Computer Science*, pages 183–198. Springer Berlin / Heidelberg.

[Nascimento 2016] Nascimento, E. (2016). SAC 2016 - Implementation of algorithm for ECDLP with errors based on a time-memory tradeoff. `https://github.com/enascimento/SCA-ECC-keyrecovery`.

[Nascimento et al. 2016] Nascimento, E., Chmielewski, L., Oswald, D., and Schwabe, P. (2016). Attacking embedded ecc implementations through cmov side channels. In *23rd Conference on Selected Areas in Cryptography (SAC 2016), St John's, Canada, August 10-12, 2016.*

[Nascimento et al. 2015] Nascimento, E., López, J., and Dahab, R. (2015). Efficient and secure elliptic curve cryptography for 8-bit avr microcontrollers. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 289–309. Springer.

[NIST 2011] NIST (2011). Non-Invasive Attack Testing Workshop.

[Okeya 2006] Okeya, K. (2006). Side channel attacks against hmacs based on block-cipher based hash functions. In Batten, L. M. and Safavi-Naini, R., editors, *Information Security and Privacy, 11th Australasian Conference, ACISP 2006, Melbourne, Australia, July 3-5, 2006, Proceedings*, volume 4058 of *Lecture Notes in Computer Science*, pages 432–443. Springer.

[Percival 2005] Percival, C. (2005). Cache missing for fun and profit. In *Proceedings of BSDCan 2005*.

[Perin and Chmielewski 2015] Perin, G. and Chmielewski, L. (2015). A Semi-Parametric Approach for Side-Channel Attacks on Protected RSA Implementations. In *CARDIS*.

[Perin et al. 2014] Perin, G., Imbertl, L., Torres, L., Maurine, P., and Montpellier, R. A. (2014). Attacking Randomized Exponentiations Using Unsupervised Learning. In *CARDIS*.

[Preneel et al. 1993] Preneel, B., Govaerts, R., and Vandewalle, J. (1993). Hash functions based on block ciphers: A synthetic approach. In Stinson, D. R., editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer.

[Riscure 2016] Riscure (2016). Riscure B.V. - Inspector SCA. `https://www.riscure.com/security-tools/inspector-sca`.

[Rivain 2011] Rivain, M. (2011). Fast and regular algorithms for scalar multiplication over elliptic curves. *IACR Cryptology ePrint Archive*, 2011:338.

[Rodrigues et al. 2016] Rodrigues, B., Pereira, F. M. Q., and Aranha, D. F. (2016). Sparse representation of implicit flows with applications to side-channel detection. In Zaks, A. and Hermenegildo, M. V., editors, *Proceedings of the 25th International Conference on Compiler Construction, CC 2016, Barcelona, Spain, March 12-18, 2016*, pages 110–120. ACM.

[Saeedi and Kong 2014] Saeedi, E. and Kong, Y. (2014). Fuzzy analysis of side channel information. *2014, 8th International Conference on Signal Processing and Communication Systems, ICSPCS 2014 - Proceedings*, pages 1–5.

[Schneider and Moradi 2016] Schneider, T. and Moradi, A. (2016). Leakage assessment methodology - extended version. *J. Cryptographic Engineering*, 6(2):85–99.

[Sedra and Smith 1997] Sedra, A. S. and Smith, K. C. (1997). *Microelectronic circuits*, chapter 4. Oxford University Press, Inc., 4th edition.

[Shannon 1949] Shannon, C. E. (1949). Communication theory of secrecy systems. *Bell Systems Technology Journal*, 28:657–715.

[Silberschatz et al. 2004] Silberschatz, A., Galvin, P. B., and Gagne, G. (2004). *Operating System Concepts*. Wiley.

[Trichina and Bellezza 2003] Trichina, E. and Bellezza, A. (2003). *Implementation of Elliptic Curve Cryptography with Built-In Counter Measures against Side Channel Attacks*, pages 98–113. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Tromer et al. 2010] Tromer, E., Osvik, D. A., and Shamir, A. (2010). Efficient Cache Attacks on AES, and Countermeasures. *Journal of Cryptology*, 23(1):37–71.

[Tunstall and Goodwill 2016] Tunstall, M. and Goodwill, G. (2016). Applying TVLA to Public Key Cryptographic Algorithms. Technical report, Eprint.

[Vaudenay 2002] Vaudenay, S. (2002). Security flaws induced by CBC padding - applications to ssl, ipsec, WTLS ... In Knudsen, L. R., editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546. Springer.

[Vredendaal 2014] Vredendaal, C. (2014). Implementation of e-enumeration algorithm for DLP-based cryptosystems. http://scarecryptow.org/publications/sckangaroos.html.

[Walter 2001] Walter, C. D. (2001). Sliding Windows Succumbs to Big Mac Attack. In *CHES*, pages 286–299.

[Witteman et al. 2011a] Witteman, M., Jaffe, J., and Rohatgi, P. (2011a). Efficient side channel testing for public key algorithms: RSA case study. Technical report, CRI.

[Witteman et al. 2011b] Witteman, M. F., van Woudenberg, J. G. J., and Menarini, F. (2011b). Defeating {RSA} Multiply-Always and Message Blinding Countermeasures. pages 77–88.

[Witten and Frank 2011] Witten, I. H. and Frank, E. (2011). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[Woudenberg et al. 2011] Woudenberg, J. G. J. V., Witteman, M. F., and Bakker, B. (2011). Improving Differential Power Analysis by Elastic Alignment. In *CT-RSA*, pages 104–119.

[Yarom and Falkner 2014] Yarom, Y. and Falkner, K. (2014). FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In Fu, K. and Jung, J., editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 719–732. USENIX Association.

[Özgen et al. 2016] Özgen, E., Papachristodoulou, L., and Batina, L. (2016). Template attacks using classification algorithms. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 242–247.