Capítulo 1

Introdução a Ataques por Canais Secundários

João Paulo Fernandes Ventura [1] , Ricardo Dahab [2]

Abstract

Embedded electronic devices capable of communicating usually implement secure cryptographic algorithms and schemes which make use of secret keys. However, these hardware devices may reveal partial (or even total) information about these keys, through unsuspected channels such as eletromagnetic emissions and power consumption traces. Attacks using these secondary means, i.e., not using the main communication channels with the device, are collectively known as Side-Channel Attacks. This text gives an introduction to this subject, discussing examples and some known countermeasures.

Resumo

Dispositivos eletrônicos embarcados capazes de comunicarem-se usualmente implementam algoritmos e esquemas criptográficos que empregam chaves sigilosas. Contudo, esses dispositivos podem revelar informações parciais (ou até completas) sobre essas chaves, por canais insuspeitos como medições de emissões eletromagnéticas ou de consumo de potência. Ataques que usam esses canais secundários, isto é, alternativos ao canal principal de comunicação com o dispositivo, são conhecidos por Ataques por Canais Secundários (ou também Ataques por Canais Colaterais). Este texto faz uma breve introdução ao assunto, discutindo exemplos e algumas contramedidas conhecidas.

1.1. Introdução

Os modelos atuais de comunicação consistem na crescente troca de informações processadas digitalmente através de canais inseguros (Figura 1.1). Portanto, fica a cargo das entidades envolvidas garantirem a privacidade, integridade e autenticidade tanto dos dados como também das próprias entidades. O provimento de tais requisitos de segurança é comumente obtido com o uso de técnicas criptográficas.

[1]Este trabalho foi parcialmente financiado pelo CNPq
[2]Este trabalho foi parcialmente financiado pela FAPESP e pelo CNPq

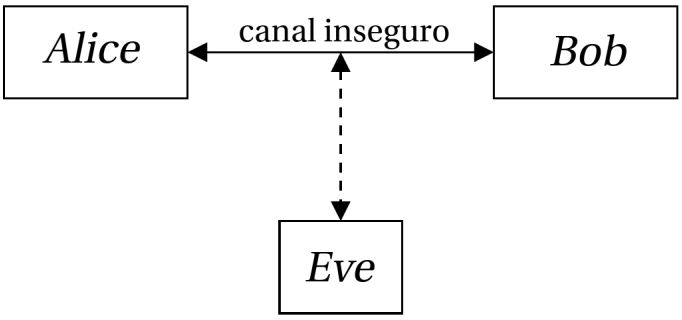


Figura 1.1. Entidades Alice e Bob se comunicam através de um canal inseguro enquanto adversário Eve tenta obter acesso a mensagem.

As primeiras técnicas modernas implementadas são os chamados *modelos criptográficos simétricos*, os quais utilizam algoritmos como TEA, DES, Triple DES e AES [Stinson 2002]. Apesar das implementações de tais algoritmos serem muito eficientes, tanto em *hardware* quanto em *software*, sua principal desvantagem advém do fato das entidades comunicantes necessitarem previamente estabelecer através de um canal seguro a chave secreta a ser utilizada (Figura 1.3(a)).

Os algoritmos criptográficos assimétricos como RSA [Rivest et al. 1978], DSA [El Gamal 1985], ECC [Koblitz 1987, Miller 1986], etc, foram criados com o intuito de eliminar o entrave do acordo de chaves (Figura 1.3(b)). Entretanto, devido à maior complexidade dos cálculos efetuados e do maior número de bits necessários para compor as chaves, esses protocolos são muito menos eficientes do que os simétricos.

Em ambos os casos, a segurança do método baseia-se na premissa de que a chave utilizada, seja a chave secreta no modelo simétrico, seja na chave privada no modelo assimétrico, não pode ser obtida observando-se somente informações públicas ou obtidas do canal inseguro. Do ponto de vista de tal premissa baseiam-se as seguintes hipóteses:

1. *Intratabilidade computacional:* a quantidade de tempo e recursos computacionais necessários para quebrar um esquema criptográfico é tão grande que torna proibitiva qualquer tentativa. O crescente poder de processamento dos computadores é o maior inimigo dessa premissa, forçando a adoção de chaves com comprimentos cada vez maiores.

2. *Ausência de falhas na especificação:* ainda que a obtenção da chave através de ataques por força bruta seja inviável, o esquema criptográfico não deve possuir falhas que viabilizem a descoberta da chave secreta.

Motivação

Ainda que as duas hipóteses anteriores sejam válidas, não existem garantias plenas de que a implementação de um método criptográfico seja segura. Informações sensíveis podem vazar através de canais secundários tais como medição do consumo de potência, emanações eletromagnéticas, etc, não previstos durante a implementação. Isso é ainda mais dramático em dispositivos eletrônicos embarcados, como *smart cards*, SIM Cards e sensores RFID porque a natureza exposta de seus circuitos os torna muito mais vulneráveis

Figura 1.2. Smart card, SIM card e sensor RFID.

a esse tipo de vazamentos. A coleta e análise dessas informações podem viabilizar um ataque contra o método criptográfico; esse novo modelo de ataque é denominado Ataque por Canais Secundários ou Ataques por Canais Concorrentes.

A proposta desse trabalho é fazer uma breve introdução a esse modelo de ataque, com foco em dispositivos embarcados.

Organização deste documento

Além desta, compõem este documento as seguintes seções:

1.2. Esquemas criptográficos de interesse: apresentação de conceitos matemáticos básicos e alguns esquemas criptográficos utilizados nas seções seguintes.

1.3. Tipificação dos ataques: categorização dos ataques de acordo com a grandeza física que pode fornecer informações sensíveis sobre a chave de um esquema criptográfico.

1.4. Exempos de ataques por canais secundários: apresentação de ataques e possíveis soluções.

1.5. Considerações finais: discussão sobre todos os resultados apresentados.

1.2. Esquemas criptográficos de interesse

Nesta seção, inicialmente será apresentado o conceito da teoria dos números, com o objetivo de embasar a descrição algoritmos simétricos e assimétricos. Ainda que esses algoritmos sejam vastamente conhecidos pela comunidade, é necessário que o leitor tenha conhecimento de certas partes fundamentais, uma vez que são elas que viabilizam os ataques.

1.2.1. Conceitos básicos da algebra e Teoria dos Números

Grupos e Corpos Finitos

Seja $\mathbb{S}$ um conjunto e $\diamond$ uma operação qualquer sobre elementos desse conjunto. O par $(\mathbb{S}, \diamond)$ é um grupo obedecer as seguintes propriedades:

1. *Fechamento* : $\forall a, b \in \mathbb{S}, a \diamond b \in \mathbb{S}$.

2. *Comutatividade* : $\forall a,b \in S, a \diamond b = b \diamond a$.
3. *Associatividade*: $\forall a,b,c \in S, (a \diamond b) \diamond c = a \diamond (b \diamond c)$.
4. *Existência de elemento neutro*: $\exists n \in \mathbb{S} \mid \forall a \in \mathbb{S}, a \diamond n = n \diamond a = a$.
5. *Existência de inverso* : $\forall a \in \mathbb{S}, \exists i \in \mathbb{S} \mid a \diamond i = i \diamond a = n$, onde n é o elemento neutro.

Um grupo é *finito* se $\mathbb{S}$ é um conjunto finito. Nesse caso, a *ordem* de um elemento $a \in \mathbb{S}$ é o menor inteiro t tal que:

$$\underbrace{a \diamond a \diamond \ldots . \diamond a}_{t} = n$$

Desse modo definimos um *corpo* como um conjunto $\mathbb{F}$ munido de duas operações, adição (denotada por $+$) e multiplicação (denotada por $\cdot$) tais que:

1. $(\mathbb{F}, +)$ formam um grupo abeliano com elemento neutro denotado por 0.
2. $(\mathbb{F} \setminus \{0\}, \cdot)$ formam um grupo abeliano com elemento neutro denotado por 1.
3. $\forall a,b,c \in \mathbb{F} : (a+b) \cdot c = a \cdot c + b \cdot c$.

Um corpo é finito quando $\mathbb{F}$ é finito e quando esse for o caso, definimos a *ordem do corpo finito* como $|\mathbb{F}| = q$. Existe apenas um corpo finito $\mathbb{F}$ de ordem q se e somente se $q = p^m$, sendo p um número primo denominado característica de $\mathbb{F}$. Dois corpos finitos de mesma ordem são iguais, a menos de um isomorfismo entre seus elementos. Assim denotamos o *corpo finito* com p^m elementos por $\mathbb{F}_{p^m}$.

Um corpo é dito *primo* se $m = 1$. Um corpo primo de ordem q pode ser definido tomando-se $\mathbb{F}_q = \mathbb{Z}_q = \{0,1,2,...q-1\}$ e as operações de soma e multiplicação usuais de *módulo q*, isto é, $(a+b) = (a+b) \bmod q$ e $(a \cdot b) = (a \cdot b) \bmod q$ onde $x \bmod p$ é o resto da divisão de x por q. Já corpos primos de ordem $q = 2^m$ são denominados *corpos binários*. Seus elementos são polinômios com coeficientes em $\{0,1\}$ e grau máximo igual a $m-1$:

$$F_{2^m} = \left\{ \sum_{i=0}^{m-1} a_i x^i : a_i \in \{0,1\} \right\}$$

Neste caso, para $a,b \in \mathbb{F}_{2^m}$, definimos $a+b$ e $a \cdot b$ módulo um polinômio irredutível $f(x)$ de grau m. Elementos de $\mathbb{F}_{2^m}$ podem ser representados como cadeias de m bits como mostra a Tabela 1.1.

0	0000	x^2	0010	x^3	0100	x^3+x^2	1100
1	0001	x^2+1	0011	x^3+1	0101	x^3+x^2+1	1101
x	0010	x^2+x	0011	x^3+x	0110	x^3+x^2+x	1110
$x+1$	0011	x^2+x+1	0011	x^3+x+1	0111	x^3+x^2+x+1	1111

Tabela 1.1. Elementos de $\mathbb{F}_{2^4}$ com polinômio irredutível $f(x) = x^4+x+1$.

Algoritmo Estendido de Euclides

Sejam a e n números naturais com $n > 0$ e $n > a$, o maior divisor comum d de a e n pode ser facilmente obtido utilizando o algoritmo de euclides que usa a seguinte propriedade indutiva:

$$d = mdc(a,n) = mdc(n \bmod a, a)$$

O Algoritmo de Euclides pode ser estendido de forma a calcular $x, y \in \mathbb{Z}$ tais que $d = ax + by$ (Algoritmo 1.1). Supondo $d = mdc(a,n) = 1$, ao final da execução do algoritmo teríamos:

$$ax + ny = 1$$

Portanto $ax = 1 - xy$, ou $(a \cdot x) \bmod n = 1$, o que implica que x é o inverso multiplicativo de a. Denotamos esse inverso por $a^{-1} \bmod n$. Dessa forma, o Algoritmo Estendido de Euclides é uma forma de determinar inversos multiplicativos em grupos abelianos multiplicativos.

No restante desse texto usaremos a seguinte notação: $Z_n = \{0, 1, 2, ..., n-1\}$ e $Z_n^* = \{1, 2, ..., n-1\}$, para n um inteiro positivo. Também, quando $a \bmod n = b \bmod n$, para a e b inteiros quaisquer e n um inteiro positivo, escrevemos $a \equiv b \pmod{n}$ e dizemos que *a e b são congruentes módulo n*.

Algoritmo 1.1. Algoritmo Estendido de Euclides

Entrada: inteiros positivos a e n tal que $a \leqslant n$
Saída: $d = mdc(a,n)$ e inteiros x, y tais que $ax + ny = d$

01. $u \leftarrow a$, $v \leftarrow n$
02. $x_1 \leftarrow 1$, $y_1 \leftarrow 1$
03. Enquanto $u \neq 0$ faça
04. $\quad q \leftarrow \left\lfloor \frac{v}{u} \right\rfloor$, $r \leftarrow v - qu$, $x \leftarrow x_2 - qx_1$, $y \leftarrow y_2 - qy_1$
05. $\quad d \leftarrow v$, $v \leftarrow x_2$, $y \leftarrow y_2$
06. Retorne (d, x, y)

Teorema chinês do resto

Seja $M = \prod_{i=1}^{r} m_i$, onde $\{m_1, m_2, ..., m_i, m_{i+1}, ..., m_r\}$ onde cada m_i é um inteiro positivo e $mdc(m_i, m_j) = 1$. Suponha o sistema:

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\ldots \\ x &\equiv a_r \pmod{m_r} \end{aligned}$$

O *Teorema Chinês do Resto* (*Chinese Remainder Theorem* ou CRT) fornece uma solução única para esse sistema dada por:

$$x = \sum_{i=1}^{r} a_i M_i y_i \bmod M,\ M_i = \frac{M}{m_i}\ e\ y_i = M_i^{-1} \bmod m_i$$

Operações aritméticas com números extensos requerem um grande tempo de processamento, sendo uma das mais caras a exponenciação modular. O Teorema Chinês do Resto é um método que permite que x mod M, para $M = \prod_{i=0}^{r} m_i$, possa ser calculado a partir do cálculo de x mod $m_1, ..., x$ mod m_r que são quantias bem menores, possibilitando uma redução do custo de processamento.

1.2.2. Sistemas simétricos

Fundamentos

Em esquemas criptográficos de chaves simétricas as entidades envolvidas na comunicação primeiramente entram em acordo de que a chave utilizada é secreta e autêntica e só então realizar a comunicação através do canal inseguros [Hankerson et al. 2003]. Exemplos de sistemas simétricos são o DES (*Data Encryption Algorithm*), o TripleDES e o AES (*Advanced Encryption Standard*) [Daemen and Rijmen 2002].

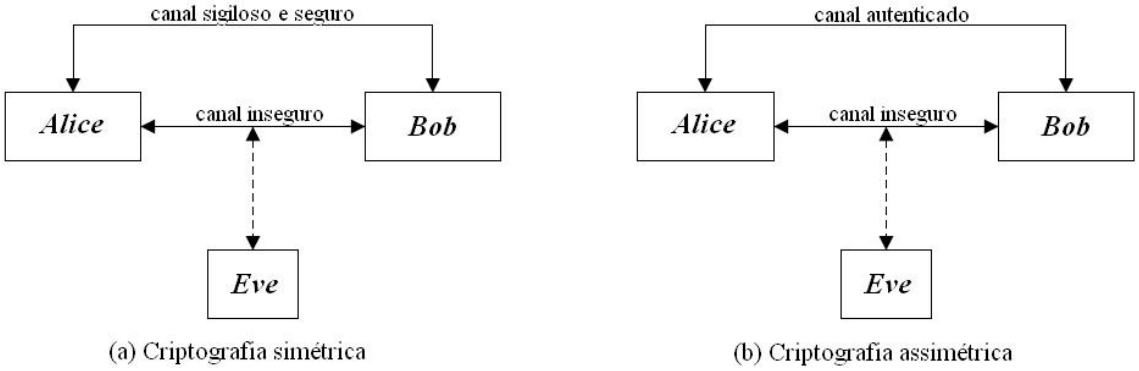


Figura 1.3. Criptografia simétrica versus criptografia assimétrica. [Hankerson et al. 2003]

Apesar de implementações tanto em *hardware* como em *software* serem extremamente eficientes, existem dois inconvenientes nesse esquema criptográfico:

- **Distribuição das chaves:** o acordo da chave secreta deve ser feito através de um canal secreto e autenticado. Isso pode ser feito fisicamente (através de um entregador confiável). Outra maneira é utilizar uma TTP (*Trusted Third Party* ou Terceira Parte Confiável) que inicialmente estabelece um acordo de chaves com todas as entidades comunicantes e distribui outras chaves entre elas conforme a necessidade.
- **Gerenciamento das chaves:** em um sistema com N entidades, cada uma delas precisaria armazenar $N-1$ chaves secretas. Ainda que seja utilizado a TTP para fazer requisição de chaves sob demanda, ela se tornaria um gargalo na comunicação [Hankerson et al. 2003].

AES: Advanced Encryption Standard

Um dos primeiros esquemas simétricos de criptografia criados foi o DES (*Data Encryption Standard*) pela IBM em 1973 a pedido do *National Bureau of Standads* (atualmente conhecido como *National Institute of Standards and Technology*). Ele foi um dos padrões mais utilizados na história. Com o crescimento da capacidade computacional desde sua criação, em 1997 um consórcio mostrou ser viável quebrar o DES e portanto o NIST lançou um processo de seleção de algoritmos para substitui-lo. O algoritmo vencedor do concurso foi criado por Vincent Rijmen e Joan Daemen e denominado Rijndael. Originalmente o Rindael suportava mensagens composta por blocos de 128, 160, 192, 224 e 256 bits utilizando chaves de 128, 160, 192, 224 ou 256 bits. Porém a versão sob o nome de AES suporta apenas blocos de 128 bits cifrados com chaves de 128, 192 ou 256 bits.

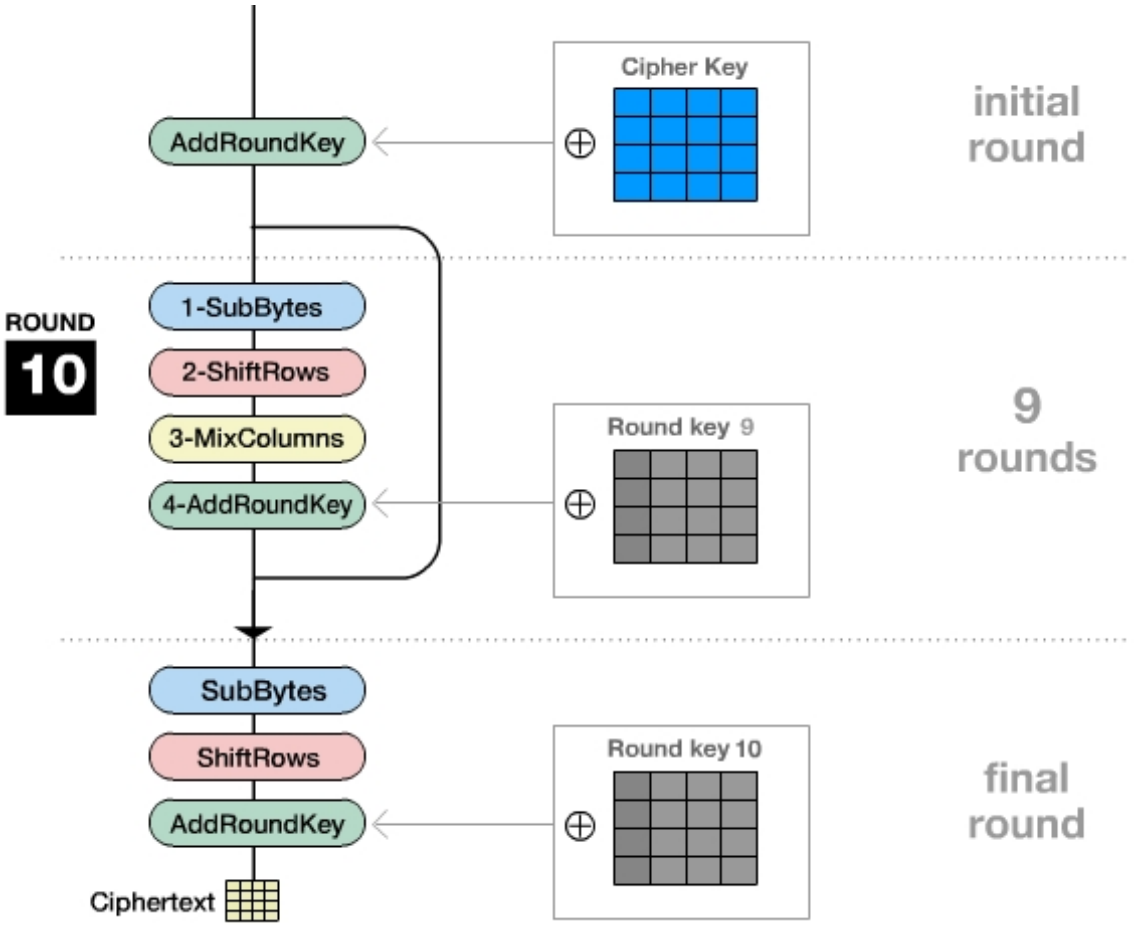


Figura 1.4. *Advanced Encryption Standard* [Zabala 2009].

No AES a unidade de encriptação das mensagens é um bloco de 128 bits, que podem ser visualizados como uma matriz 4×4 com termos de 1 *byte*. Como as operações são feitas *byte-a-byte* e cada *byte* possui 8 bits, as operações podem ser entendidas como operações sobre corpos finitos binários da forma F_{2^8}. O processo de encriptação consiste em realizar *rounds* (ou rodadas) de transformações repetidamente e os resultados intermediários da encriptação são armazenados em uma matriz 4×4 denominada $State$.

Como mostra a Figura 1.4, as etapas que podem compor uma rodada são:

1. ***AddRoundKey:*** nesse passo a subchave é combinada com a matriz de estados. Em cada rodada, uma nova subchave é derivada a partir da chave principal utilizando o algoritmo de escalonamento de chaves do AES, sendo que as subchaves possuem o mesmo tamanho da matriz de estados. A operação de adição consiste em realizar um *ou-exclusivo* de cada um dos bits que compõem as matrizes (Figura 1.5).

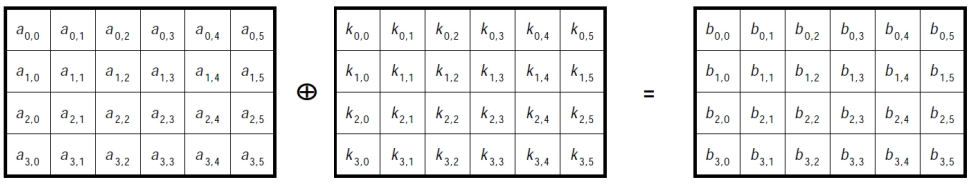


Figura 1.5. *AddRoundKeys* [Daemen and Rijmen 2002].

2. ***SubBytes:*** cada *byte* do vetor é atualizado utilizando uma de matriz de substituição com termos de 8 bits denominada S-BOX. É essa matriz é o que garante parte da aleatoriedade do encriptador. A S-BOX é gerada a partir dos inversos multiplicativos sobre F_{2^8} (Figura 1.6).

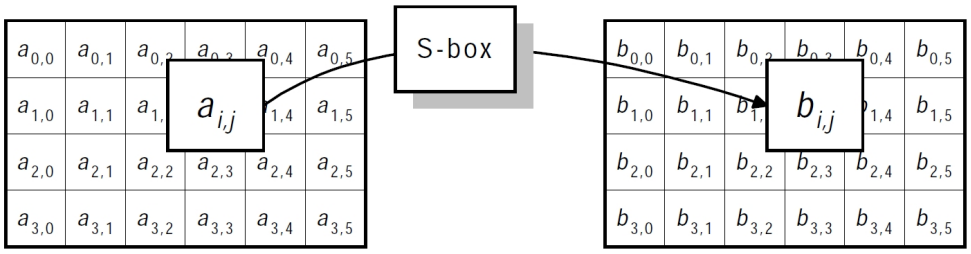


Figura 1.6. *SubBytes* [Daemen and Rijmen 2002].

3. ***ShiftsRows***: nesse estágio as linhas são rotacionadas de um certo número i de termos à esquerda, sendo i o índice da linha da matriz. Logo, enquanto a primeira linha ($i = 0$) não sofre rotação, a quarta linha ($i = 3$) é rotacionada de três termos (Figura 1.7).

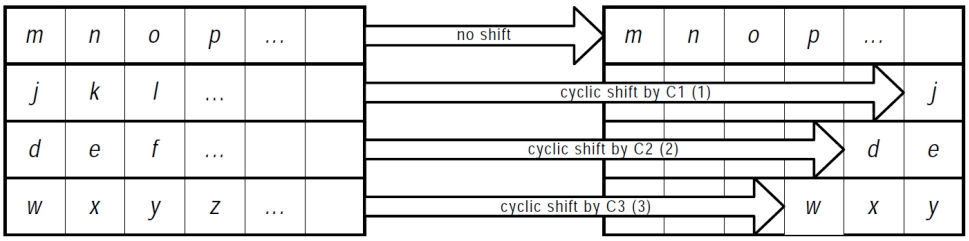


Figura 1.7. *ShiftRows* [Daemen and Rijmen 2002].

4. ***MixColumns:*** nesse estágio, as colunas da matriz *State* são interpretadas como elementos de $F_{2^8}(x)$, sendo $f(x) = x^4 + 1$ o polinômio irredutível utilizado. Cada coluna é multiplicada por $c(x) = (03, 01, 01, 02) \in F_{2^8}(x)$. Essa operação pode ser visualizada como uma multiplicação de matrizes (Figura 1.8).

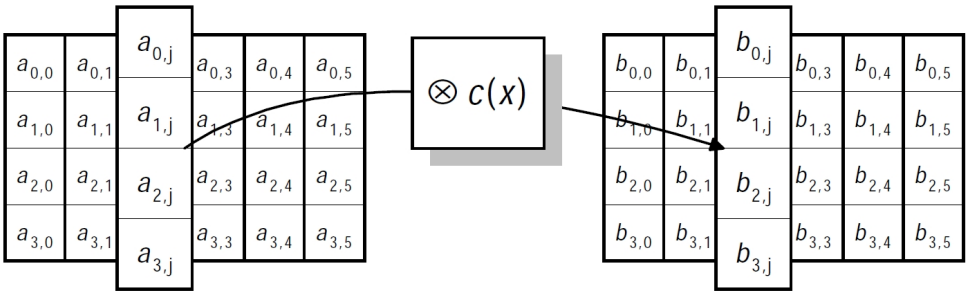


Figura 1.8. *MixColumns* [Daemen and Rijmen 2002].

1.2.3. Sistemas assimétricos

Sistemas assimétricos, também conhecidos como sistemas de chave pública, foram inicialmente propostos [Diffie and Hellman 1976] para solucionar as questões da distribuição e gerenciamento das chaves secretas na criptografia simétrica. Nesse esquema criptográfico, as entidades não precisam garantir o sigilo das chaves de encriptação mas apenas sua autenticidade. Cada entidade deve escolher um par (e,d), correspondendo respectivamente a *chave pública* (para encriptar) e a *chave privada* (para decriptar), sendo computacional inviável descobrir a chave privada partindo apenas da chave pública. Esses sistemas também precisam prover mecanismos para garantir o sigilo das mensagens, o não repúdio das entidades e a autencididade de mensagens e entidades [Stinson 2002, Hankerson et al. 2003].

Apesar de resolver os inconvenientes de sistemas simétricos, sistemas de chave pública são substancialmente mais lentos. Por isso, a prática é a utilização de sistemas híbridos, nos quais utiliza-se o sistema assimétrico apenas para estabelecer chaves para serem utilizadas em sistemas simétricos.

1.2.3.1. Sistemas RSA

Os Algoritmos 1.2, 1.3 e 1.4 descrevem os procedimentos para geração de chaves, encriptação e decriptação, respectivamente, do RSA. A notação $a \in_R [x,y]$ significa escolher um número a aleatoriamente dentro do intervalo $[x,y]$.

É importante notar que todas as operações descritas nesses algoritmos podem ser implementadas eficientemente. Em particular, o cálculo da chave privada d no passo 04 do Algoritmo 1.2 é feito com o Algoritmo Estendido de Euclides.

A robustez criptográfica do RSA vem da dificuldade reconhecida do problema da fatoração de inteiros grandes, no caso o inteiro n, e da também reconhecida dificuldade de inversão da função de encriptação:

$$m^e \bmod n$$

sem o conhecimento da chave privada d.

Algoritmo 1.2. Geração do par de chaves do RSA
Entrada: parâmetro seguro l
Saída: chave pública (e,n) e chave privada (d)

01. Escolher aleatóriamente dois primos p e q de $\frac{l}{2}$ *bits de comprimento*.
02. Calcular $n \leftarrow pq$ e $\phi(n) \leftarrow (p-1)(q-1)$
03. Escolher $e \in_R [1, \phi(n)]$ tal que $mdc(e, \phi(n)) = 1$
04. Calcular um inteiro d tal que $d \in [1, \phi(n)]$ e $ed \equiv 1 \pmod{\phi(n)}$

Algoritmo 1.3. Encriptação básica do RSA
Entrada: chave pública (e,n) e texti claro $m \in [0, n-1]$
Saída: texto encriptado c

01. Calcular $c \leftarrow m^e \bmod n$.
02. Retornar c

Algoritmo 1.4. Decriptação básica do RSA
Entrada: chave pública (e,n), chave privada d, texto encriptado c
Saída: texto claro m

01. Calcular $m \leftarrow c^d \bmod n$.
02. Retornar m

Para assinar uma mensagem usando o RSA, procede-se como descrito no Algoritmo 1.5, onde h é uma função de resumo criptográfico (*hash function*). O leitor interessado pode encontrar maiores detalhes em [Stinson 2002].

Algoritmo 1.5. Assinatura básica do RSA
Entrada: chave pública (e,n), chave privada d, texto claro m
Saída: assinatura de m

01. Calcular $s \leftarrow h(m)^d \bmod n$, onde $h(m)$ é o resumo da mensagem m.
02. Retornar s

1.2.3.2. Sistemas baseados em curvas elípticas

Fundamentos algébricos

Uma curva elíptica E sobre um corpo $\mathbb{F}$ é definida pela seguinte equação, chamada de equação de Weierstrass como [Hankerson et al. 2003]:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

onde $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$ e o discriminante $\Delta \neq 0$, onde

$$\begin{aligned}
\Delta &= -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6, \ e \\
d_2 &= a_1^2 + 4a_2, \\
d_4 &= 2a_4 + a_1 a_3, \\
d_6 &= a_3^2 + 4a_6, \\
d_8 &= a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2.
\end{aligned}$$

Os pontos em E de interesse para nós serão os definidos pelo conjunto $E(\mathbb{F})$, a saber

$$E(\mathbb{F}) = \{(x,y) \in \mathbb{F} \times \mathbb{F} : a_1 xy + a_3 y - x^3 - a_2 x^2 - a_4 x - a_6 = 0\} \cup \{\mathscr{O}\} \qquad (1)$$

onde $\mathscr{O}$ é o ponto no infinito. A Figura 1.9 mostra duas curvas elípticas E_1 e E_2 definidas sobre o corpo $\mathbb{R}$.

Neste texto, nosso interesse é em curvas elípticas sobre corpos finitos. É sabido que os pontos do conjunto $E(\mathbb{F})$ formam um grupo com uma operação de adição bastante peculiar cuja definição precisa não é do escopo desse texto. Para todos os efeitos, nos limitamos a denotá-la por $+$; conseqüentemente denotaremos a soma de k parcelas do ponto P por kP. No restante do texto, usaremos a expressão *curva elíptica* $E(\mathbb{F})$ significando o conjunto definido na equação (1).

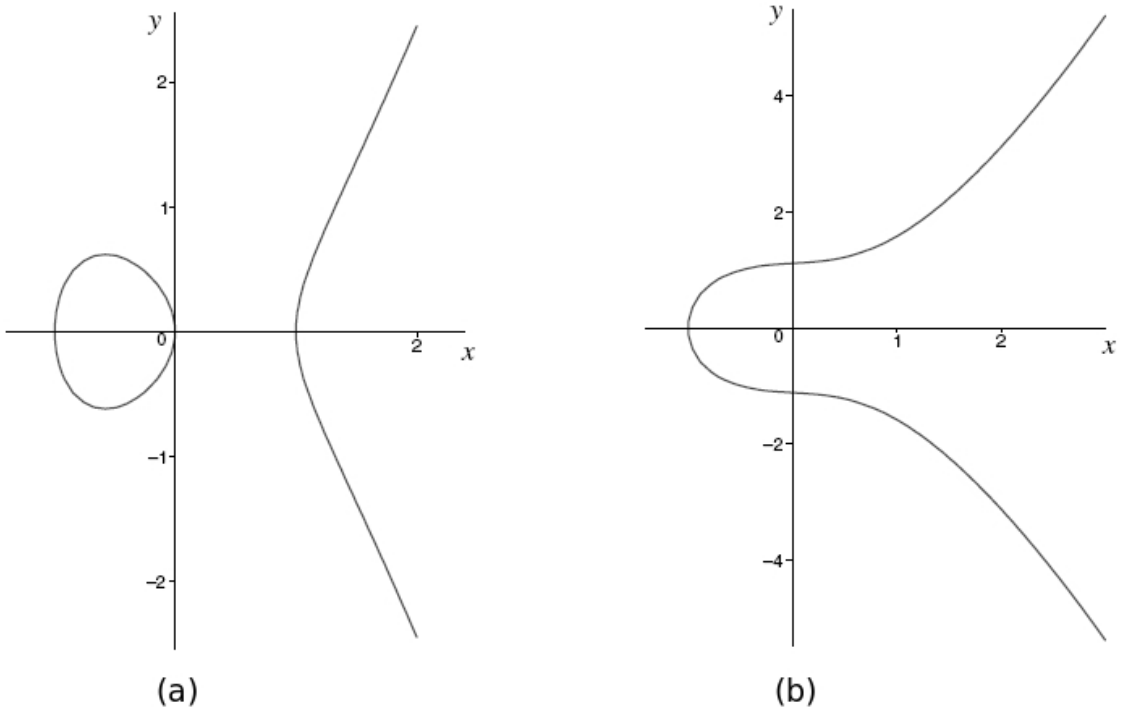


Figura 1.9. Curvas elípticas (a) $E_1 : y^2 = x^3 - x$ **e (b)** $E_2 : y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$ **[Hankerson et al. 2003].**

Esquemas de encriptação e decriptação

Criptografia baseada em curvas elípticas foi inicialmente proposta por Miller em 1986 [Miller 1986] e Neal Koblitz em 1989 [Koblitz 1987].

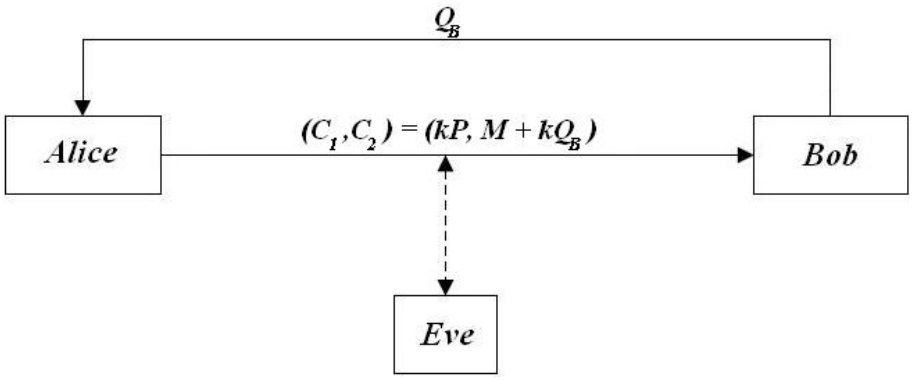


Figura 1.10. Esquema de encriptação baseados em curvas elípticas.

Devido a restrições do poder de processamento e memória e disponíveis, dispositivos embarcados preferencialmente utilizam sistemas ECC. Isso ocorre porque com chaves secretas de poucas centenas de bits é possível alcançar os mesmos níveis de segurança de um sistema *RSA* de milhares de bits.

Seja uma curva elíptica $E(\mathbb{F})$, P um ponto $P \in E(\mathbb{F})$ de ordem n. Para criar sua chave pública, Bob determina um ponto um inteiro $d \in_R [1, n-1]$ e calcula o produto $Q = dP$. Assim sua chave pública é o ponto Q e d sua chave privada. Para enviar uma mensagem para Bob, Alice deve representar sua mensagem m como um ponto $M \in E(\mathbb{F})$. Em seguida ela escolhe $k \in_R [1, n-1]$ e, utilizando os valores públicos P e Q, ela calcula $C_1 = kP$ e $C_2 = M + kQ$. A mensagem encriptada corresponde a esse par de pontos (C_1, C_2) (Algoritmo 1.2).

Algoritmo 1.2. Encriptação básica em curvas elípticas
Entrada: parâmetros públicos $(p, E(\mathbb{F}), P, n)$
Saída: mensagem encriptada (C_1, C_2)

01. Representar a mensagem m como um ponto $M \in E(\mathbb{F}_p)$
02. Escolher $k \in_R [1, n-1]$
03. Calcular $C_1 \leftarrow kP$
04. Calcular $C_2 \leftarrow M + kQ$
05. Retornar (C_1, C_2)

Bob pode facilmente decriptar a mensagem calculando (Algoritmo 1.3):

$$\begin{aligned} M &= C_2 - dC_1 \\ &= C_2 - d(kP) = C_2 - k(dP) \\ &= C_2 - dQ \end{aligned}$$

Algoritmo 1.3. Decriptação básica em curvas elípticas
Entrada: parâmetros $(p, E(\mathbb{F}), P, n)$, chave privada d e texto encriptado (C_1, C_2)
Saída: texto claro m

01. Calcular $M \leftarrow C_2 - dC_1$
02. Extrair mensagem m a partir de M
03. Retornar m

Representação dos pontos

Seja $\mathbb{F}$ um corpo onde cálculo de inversos é computacionalmente mais caro do que a multiplicação. Podemos representar a curva elíptica $E(\mathbb{F})$ no *sistema de coordenadas projetivas*, de modo que a inversão torne-se uma operação mais barata. Sejam c e d inteiros positivos e $\mathbb{F}$ um corpo; podemos definir uma relação de equivalência ($\sim$) sobre o conjunto $\mathbb{F}^3 \setminus \{(0,0,0)\}$ de triplas sobre $\mathbb{F}$ como

$$(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2),\ se X_1 = \lambda^c X_2,\ Y_1 = \lambda^d Y_2,\ Z_1 = \lambda Z_2\ para\ algum\ \lambda \in \mathbb{F}^* :$$

Assim, a *classe de equivalência* contendo $(X, Y, Z) \in \mathbb{F}^3 \setminus \{(0,0,0)\}$ é definida por

$$(X : Y : Z) = \left\{ (\lambda^c X, \lambda^d Y, \lambda Z) : \lambda \in \mathbb{F} \right\},$$

onde $(X : Y : Z)$ é denominado *ponto projetivo* e (X, Y, Z) é um *representante* de $(X : Y : Z)$.

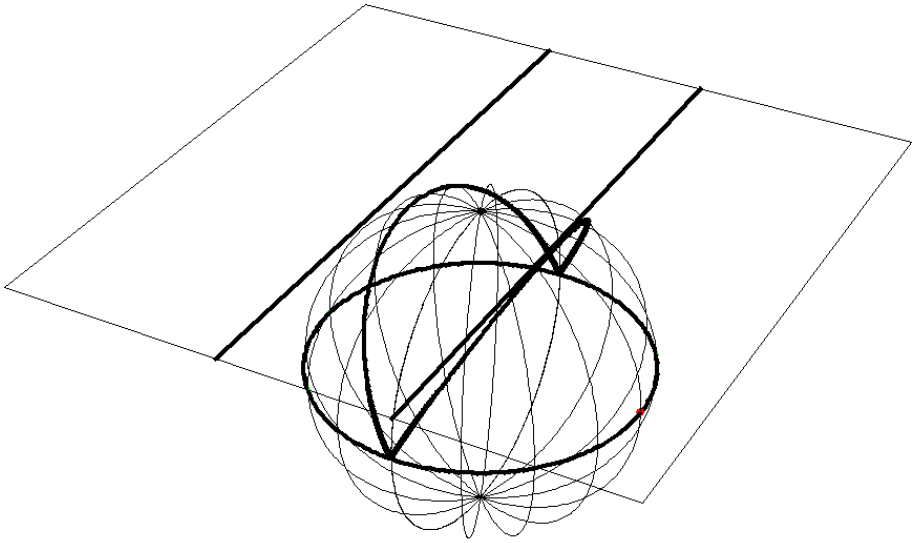


Figura 1.11. Plano projetivo [Hankerson et al. 2003].

Denotando o conjunto de todos os pontos projetivos de $\mathbb{F}$ por $\mathbb{P}(\mathbb{F})$, podemos verificar que existe uma bijeção entre o conjunto de pontos projetivos

$$\mathbb{P}(\mathbb{F})^* = \{(X : Y : Z) : X, Y, Z \in \mathbb{F}, Z \neq 0\}$$

e o conjunto de *pontos afins*

$$\mathbb{A}(\mathbb{F}) = \{(x,y) : x,y \in \mathbb{F}\},$$

sobre o qual a equação de Weierstrass é usualmente definida. O conjunto de pontos projetivos

$$\mathbb{P}(\mathbb{F})^0 = \{(X : Y : Z) : X,Y,Z \in \mathbb{F}, Z = 0\}$$

é denominado *linha no infinito*, sendo que nenhum elemento dos pontos afins é mapeados para esse conjunto. Ilustrando graficamente, vemos na Figura 1.11 a projeção do plano que contém os pontos de $\mathbb{A}(\mathbb{F})$ sobre o espaço projetivo. Todos os pontos no espaço correspondem a $\mathbb{P}(\mathbb{F})^*$, exceto o plano identificado pela circunferência que corresponde ao conjunto $\mathbb{P}(\mathbb{F})^0$.

A forma projetiva da equação de Weierstrass pode ser definida substituindo x por $\frac{X}{Z^c}$ e y por $\frac{Y}{Z^d}$, $Z \neq 0$. O ponto no infinito em $\mathbb{A}(\mathbb{F})$ pode ser conceitualmente interpretado como qualquer reta paralela ao eixo y; então, tomando duas dessas retas paralelas, a projeção do ponto no infinito corresponde à intersecção das projeções das retas que pertence ao plano $\mathbb{P}(\mathbb{F})^0$.

A equação de Weierstrass pode ser projetada sobre diferentes sistemas de coordenadas projetivas, os quais são classificados de acordo com os valores de c e d. Tomando a curva $E(\mathbb{F}_{p^m}) : y^2 = x^3 + ax + b$, $p \notin \{2,3\}$, a equação pode ser projetada sobre os seguintes tipos coordenadas projetivas:

- *Coordenadas padrão:* tomando $c = d = 1$, os pontos afins correspondem a $(\frac{X}{Z}, \frac{Y}{Z})$ e a equação da curva
$$Y^2Z = X^3 + aXZ^2 + bZ^3$$
tem como ponto no infinito $(0 : 1 : 0)$; assim para P = $(X : Y : Z)$ temos $-P = (X : -Y : Z)$.

- *Coordenadas jacobianas:* tomando $c = 2$ e $d = 3$, os pontos afins correspondem a $(\frac{X}{Z^2}, \frac{Y}{Z^3})$ e a equação da curva
$$Y^2 = X^3 + aXZ^4 + bZ^6$$
tem como ponto no infinito $(1 : 1 : 0)$; assim para P = $(X : Y : Z)$ temos $-P = (X : -Y : Z)$.

Outros sistemas de coordenadas não serão apresentados, uma vez que os ataques neste texto utilizam apenas coordenadas padrão e jacobianas. O leitor interessado pode encontrar maiores detalhes em [Hankerson et al. 2003].

Forma não-adjacente

Seja k um número inteiro representado por n bits. Pode-se representar k usando uma forma alternativa, $NAF(k) = (k'_{l-1}, k'_{l-2}, ..., k'_1, k'_0)$, com as seguintes propriedades:

1. $k'_i \in \{-1, 0, 1\}$.

2. Não existem k'_i e k'_{i+1}, ambos não nulos na representação $NAF(k)$.

3. O valor de k é dado por:

$$k = \sum_{i=0}^{l-1} k'_i 2^i$$

As propriedades de $NAF(k)$ são as seguintes:

1. $NAF(k)$ é única.

2. $NAF(k)$ tem a menor quantidade possível de dígitos não nulos.

3. $n \leqslant l \leqslant n+1$.

4. $\frac{2^l}{3} < k < \frac{2^{l+1}}{3}$

5. $NAF(k)$ possui aproximadamente $\frac{2l}{3}$ dígitos nulos

Como exemplo, o número 118_{10} (sistema decimal) equivaleria a $0111\ 0110_2$ e a $1000\ \bar{1}0\bar{1}0_{NAF_2}$, sendo $\bar{1} = -1$. Essa representação é utilizada para acelerar algoritmos de multiplicação de pontos em curvas elípticas porque como a quantidade de dígitos nulos é muito maior em $NAF(k)$ do que na representação usual, uma menor quantidade de somas são realizadas.

1.3. Tipificação dos ataques

Inicialmente serão apresentados canais não previstos pelos quais informações sensíveis sobre a chave secreta podem ser obtidas. Em seguida serão apresentados alguns ataques e suas respectivas contra-medidas.

1.3.1. Análise simples de potência

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.

Supondo que o adversário saiba qual o algoritmo implementado, ele pode determinar em quais instantes o dispositivo realiza operações de matemáticas que utilizem a chave secreta (como assinatura digital ou decriptação de mensagens) e, de acordo com o *power trace*, determinar o valor dos bits que formam a chave. Esse modelo de ataque é denominado Análise Simples de Potência (SPA ou *Simple Power Analysis*).

1.3.2. Análise diferencial de potência

Quando a variação do consumo de potência não é sensível o suficiente em relação as operações executadas por um dispositivo, o adversário pode monitorar como o consumo varia em relação ao valor de uma determinada variável. Nesse ataque, primeiramente detectamos 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. 2003]. Esse modelo de ataque é conhecido como Análise Diferencial de Potência (DPA ou *Differential Power Analysis*).

1.3.3. Análise simples e análise diferencial de campos eletromagnéticos

A passagem de uma corrente elétrica através de qualquer dispositivo eletrônico induz um campo magnético ao seu redor. Assim como a potência consumida, as emanações eletromagnéticas podem variar em função das instruções executadas por um algoritmo criptográfico.

Análise Simples de Ondas Eletromagnéticas (SEMA ou *Simple ElectroMagnetic Analysis*) e Análise Diferencial de Ondas Eletromagnéticas (DEMA ou *Differential ElectroMagnetic Analysis*) são métodos não intrusivos e relativamente baratos de atacar um sistema criptográfico.

Em ataques de consumo de potência, o adversário monitora o consumo de potência de todo um conjunto de unidades lógicas ativas simultaneamente, enquanto em ataques eletromagnéticos o adversário recebe os sinais de todos as unidades do mesmo conjunto e precisa separá-los antes de análisá-los. Apesar da dificuldade maior na coleta de dados, uma vez separados, os sinais eletromagnécitos podem revelar muito mais informações da execução do esquema criptográfico, tornando EMA muito mais ameaçadora do que SPA e DPA [Hankerson et al. 2003].

Estudos utilizando EMA [Agrawal et al. 2003] demonstraram ser possível comprometer a segurança de *smart cards* providos de medidas de proteção contra ataques SPA/DPA. Mais recentemente Oren e Shamir [Oren and Shamir 2007] mostraram que sensores RFID (como os utilizados em passaportes digitais) poderiam ter sua seguraça comprometida em ataques aplicados a uma distância de até 15 metros.

1.3.4. Análise de falhas

Boneh, DeMillo e Lipton [Boneh et al. 2001a] apresentaram pela primeira vez ataques em dispositivos explorando falhas na geração das saídas dos programas. Eles mostraram como era possível induzir dispositivos embarcados a gerar saídas erradas durante a execução de uma assinatura RSA, e descobrir a chave privada.

A ocorrência de erros enquanto um dispositivo realiza operações com a chave privada gera mensagens erradas, mas capazes de oferecer informações substanciais para

um adversário [Boneh et al. 2001b, Hankerson et al. 2003].

Esses erros podem ser inerentes aos dispositivos ou induzidos; logo, a implementação deve ser tolerante a falhas. Contudo, os dispositivos mais expostos a interferências externas são os de menor capacidade de processamento; então, o desenvolvedor precisa lidar com a dualidade da criação de uma implementação robusta porém eficiente.

1.3.5. Análise de mensagens de erro

Durante o processo de encriptação com chaves públicas; por exemplo pelo algoritmo ECIES [Hankerson et al. 2003], mensagens de erro podem ser geradas porque a entrada não estava no formato adequado. Através de medições precisas de tempo, o adversário pode determinar o exato instante da ocorrência de erros ou mesmo obter acesso ao histórico das ocorrências de erro.

Essas informações podem ser suficientes para o adversário ser capaz de desvendar a chave privada [Hankerson et al. 2003].

1.3.6. Análise de tempo

A premissa fundamental de ataques temporais é que o tempo gasto na execução de uma instrução é influenciado por seus respectivos operandos [Hankerson et al. 2003]. 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.

1.4. Exemplos de ataques

1.4.1. 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 1.4).

Algoritmo 1.4. Método NAF binário de multiplicação escalar de um ponto
Entrada: inteiro positivo k e $P \in E(\mathbb{F}_p)$
Saída: kP

01. Calcular $(k'_{l-1}, k'_{l-2}, ..., k'_1, k'_0) \leftarrow NAF(k)$
02. $Q \leftarrow \mathcal{O}$
03. De $i = l-1$ até 0 faça
04. $\quad Q \leftarrow 2Q$
05. $\quad$ Se $k'_i = \quad 1$ então $Q \leftarrow Q + P$
06. $\quad$ Se $k'_i = -1$ então $Q \leftarrow Q - P$
07. Retorne Q

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. Conseqüentemente uma quantidade muito menor de adições (linhas 5 e 6 do Algoritmo 1.4) são efetuadas.

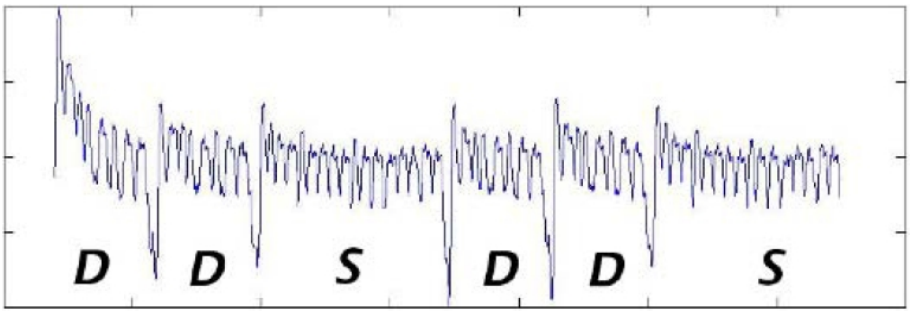


Figura 1.12. Consumo de potência durante cálculo de kP [Hankerson et al. 2003].

Entretanto um adversário que soubesse que o dispositivo implementa um algoritmo *ECDSA* poderia monitorar o consumo de potência do dispositivo utilizando um osciloscópio, obtendo o gráfico mostrado na Figura 1.12. No Algoritmo 1.4, 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 através de ataques por força bruta, pois apenas $1/3$ dos dígitos são não nulos.

Medidas preventivas contra SPA

A solução mais simples contra SPA consiste em inserir operações redundantes no algoritmo de multiplicação (Algoritmo 1.6), de modo que a seqüência de operações elementares envolvidas sejam realizadas em igual proporção. Comparando o novo *power trace* obtido (Figura 1.13) não é possível diferenciar adições de multiplicações.

Algoritmo 1.6. Multiplicação escalar de ponto resistente à SPA
Entrada: inteiro positivo k e $P \in E(\mathbb{F}_p)$
Saída: kP

01. $Q_0 \leftarrow \mathcal{O}$
02. De $i = l-1$ até 0 faça
03. $\quad Q_0 = 2Q_0$
04. $\quad Q_1 = Q_0 + P$
05. $\quad Q_0 = Q_{k_i}$
06. Retorne Q_0

1.4.2. Análise diferencial de potência sobre ECDSA

Ainda que o Algoritmo 1.6 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*,

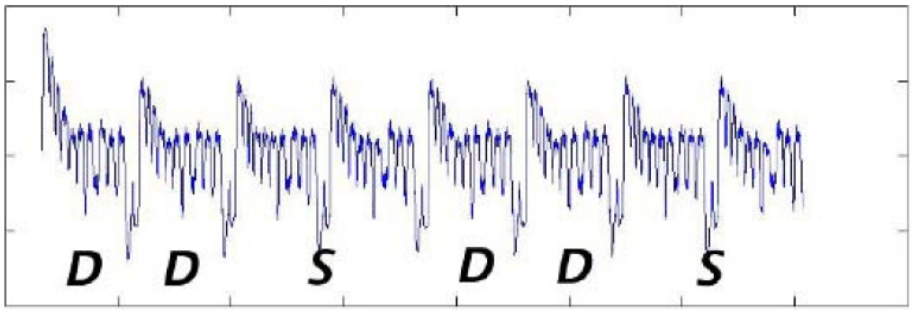


Figura 1.13. Consumo de potência durante cálculo de kP [Hankerson et al. 2003].

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.

No algoritmo de multiplicacao de pontos da curva elíptica (Algoritmo 1.6), suponha que Eve colete *power traces* durante os cálculos $kP_1, kP_2, ..., kP_r$. Como $P_1, P_2, ..., P_r$ são públicos, ele precisa determinar apenas k.

	Q_0	Q_0	k_{t-1}	$Q_0 \leftarrow Q_{k_{t-1}}$
1	$\mathscr{O}$	P	1	P
2	...	...	...	...
3	...	...	...	...
...	...	...	...	...

Tabela 1.2. Adversário conhece apenas $k_{t-1} = 1$.

Dado $Q_0 = \mathscr{O}$, o passo 2.1 é trivial e pode ser distingüir da de uma operação não trivial através do power trace, logo o adversário pode facilmente identificar o bit mais a esquerda cujo valor e 1 (Tabela 1.2). Tomando $k_{t-1} = 1$, na segunda iteração do algoritmo temos que se $k_{t-2} = 0$ então $Q_0 = 2P$; ou se $k_{t-2} = 1$ então $Q_0 = 4P$ (Tabela 1.3).

	Q_0	Q_0	k_{t-1}	$Q_0 \leftarrow Q_{k_{t-1}}$
1	$\mathscr{O}$	P	1	P
2	$2P$	$4P$	?	?
3	...	...	...	...
...	...	...	...	...

Tabela 1.3. Se k_0, então $Q_0 = 2P$. Caso contrário $Q_0 = 4P$.

Conseqüentemente, 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. 2003].

	Q_0	Q_0	k_{t-1}	$Q_0 \leftarrow Q_{k_{t-1}}$
1	$\mathcal{O}$	P	1	P
2	$2P$	$4P$	0	$2P$
3	$4P$	$6P$	...	...
...	...	...	...	...

Tabela 1.4. Nessa iteração, $Q_0 = 4P$ se e somente se $k_{t-2} = 0$.

Medidas preventivas

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 Q_0 e Q_1 são representados em coordenadas jacobianas [Hankerson et al. 2003].

Se $P = (x, y)$ no sistema afim, após a primeira atribuição $Q_1 \leftarrow P$ teríamos $Q_1 = (x : y : 1)$. Então, Q_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.

1.4.3. Análise eletromagnética de um PDA Java

Bibliotecas de segurança das API Java SE e Java ME provém mecanismos de segurança (criptografia, controle de acesso, autenticidade, etc.) usualmente utilizados em algoritmos e protocolos [Gong and Ellison 2003]. Quantidades crescentes de aplicativos baseadas nessa tecnologia são utilizados em dispositivos móveis como celulares e PDAs. Portanto, é necessário garantir que esses *softwares* sejam resistentes a ataques por canais secundários.

O estudo a seguir [Gebotys and White 2008] mostra a viabilidade de um ataque eletromagnético em um PDA que possui uma implementação em Java do algoritmo AES. Os passos desse método de ataque pode ser visto na Figura 1.14

Aquisição dos sinais EM de todo o programa

O primeiro passo consiste em capturar conjuntos de sinais eletromagnéticos do dispositivo enquanto ele executa um algoritmo criptográfico, sendo um conjunto de sinais denominado *frame* ou *trace*. Para a obtenção dos *frames*, a unidade de processamento foi exposta e ligada a um dispositivo de captura, que envia os sinais para um pré-amplificador antes de serem lidos por um osciloscópio.

A captura de sinais deve levar em consideração interferências de outros aplicativos executados concorrentemente no PDA. Em algumas capturas, existem curtos períodos em que a atividade eletromagnética praticamente cessa. Eles correspondem a instantes em que o aplicativo que realiza o encriptação sofreu interrupções realizadas pelo sistema operacional do PDA. Também vemos longos períodos sem atividade, correspondendo aos instantes em que a *thread* do aplicativo foi colocada para dormir (*sleep mode*).

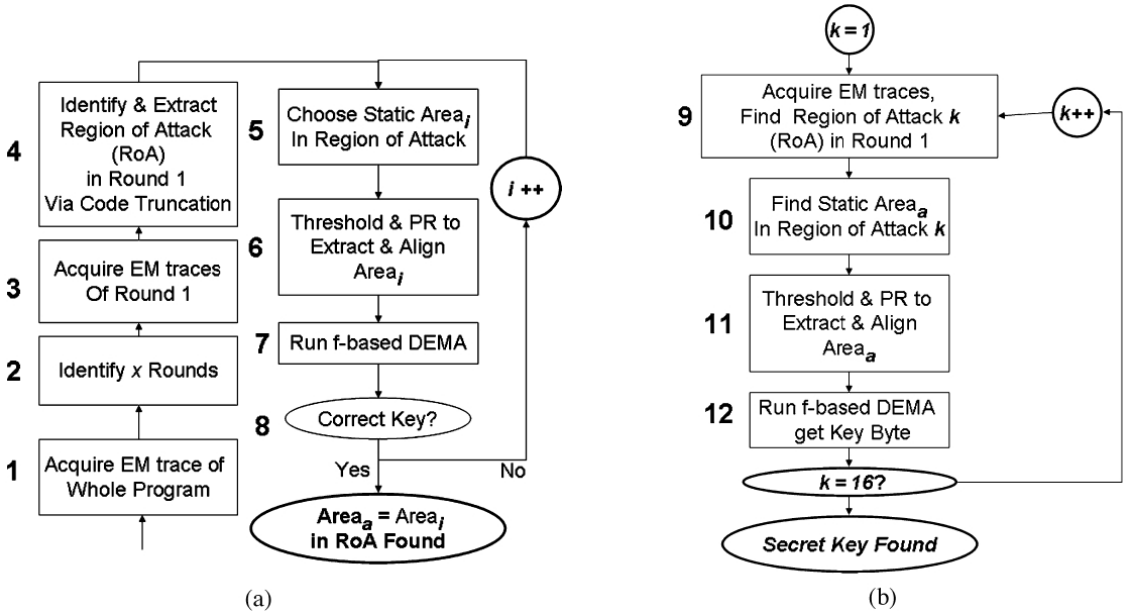


Figura 1.14. Metodologia de caracterização (a) e ataque (b) do PDA [Gebotys and White 2008].

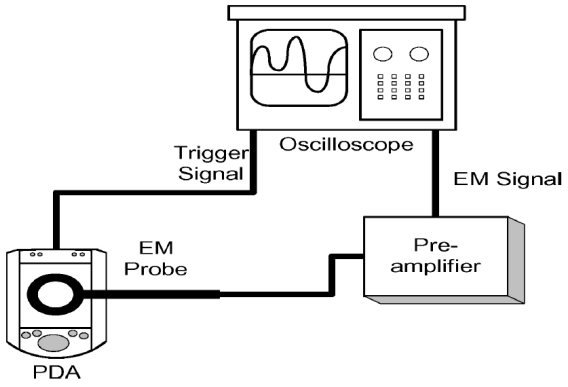


Figura 1.15. Equipamentos utilizados para captura.

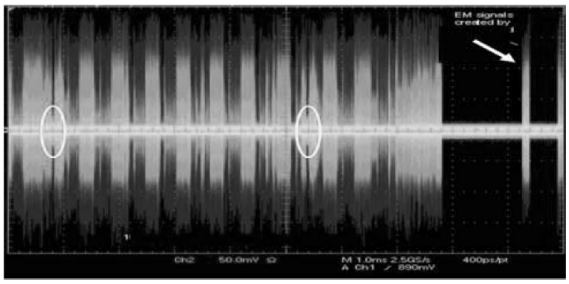


Figura 1.16. Emissão de sinais EM interrompida pelo sistema [Gebotys and White 2008].

Identificação das rodadas

As Figuras 1.17 (a) e (b) mostram os sinais eletromagnéticos adquiridos para a execução do AES com respectivamente 10 e 12 rodadas. Como cada rodada executa a mesma

quantidade de instruções, então podem ser percebidos longos períodos de grande atividade eletromagnética (as rodadas) separadas por curtos períodos de baixa atividade (acesso à tabela S-Box).

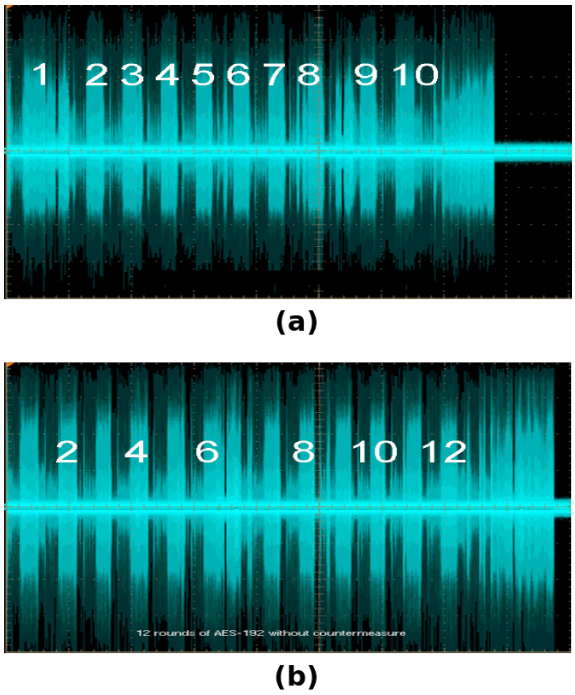


Figura 1.17. Identificação das rodadas do AES [Gebotys and White 2006].

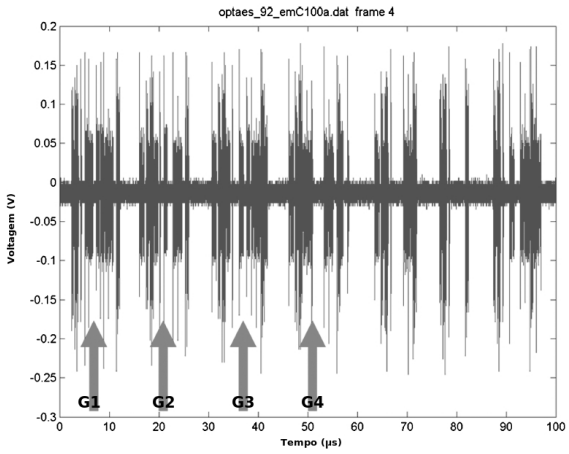


Figura 1.18. Aquisição do *frame* 4 [Gebotys and White 2008].

Adquirir sinais EM da primeira rodada

Para um texto claro 128 bits, apenas o *byte* mais significativo é alterado em cada uma das iterações do algoritmo. Tomando uma execução do AES para uma chave privada $k = 92$, o quarto *frame* capturado é mostrado na Figura 1.18. As quatro setas presentes no gráfico

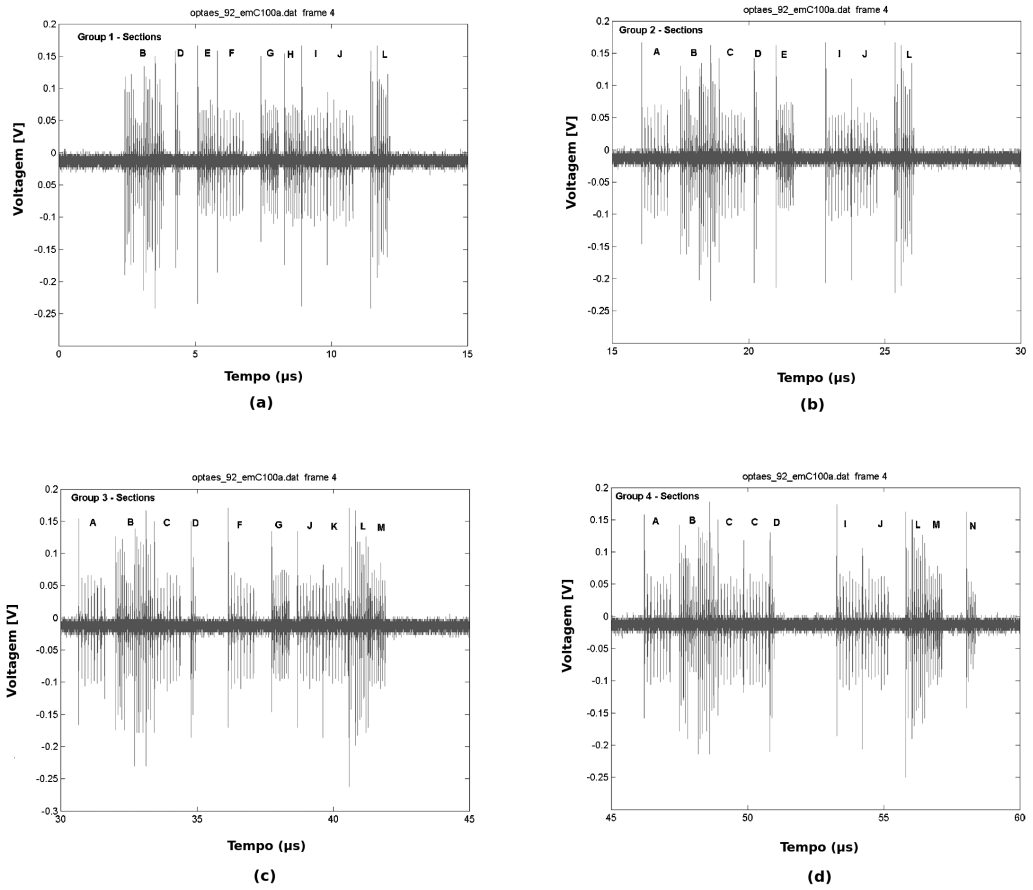


Figura 1.19. Aquisição do *frame* 4 na rodada 1, para grupos de 1 a 4 [Gebotys and White 2008].

são denominadas grupos e numerados de 1 a 4, sendo que eles correspondem aos quatro acessos a tabela para a criação de $t[0]$ (Figura 1.22, linha 6).

A Figura 1.19 mostra os quatro grupos da Figura 1.18 ampliados. Neles vemos 14 áreas identificadas de A a N; entretanto apenas o *frame* 4 possui a região N. Isso indica que a região N provavelmente é uma leitura no vetor $t[\,]$.

Como o vetor possui quatro posições, o comportamento esperado no osciloscópio seria de 16 grupos seguidos de uma atividade magnética de escrita em memória, assinalando o fim de uma rodada. Na Figura 1.20 (a) vemos o grupo 16 seguido das medições das operações $state[\,][\,]$ e $AddRoundKey$ assinaladas respectivamente por círculos e retângulos. Em seguida, vemos na Figura 1.20 (b) o comportamento descrito até agora se repetindo, indicando o início da segunda rodada.

Identificar e extrair região de ataque na primeira rodada via *Code Truncation*

Uma vez identificado o *frame* da primeira rodada, primeiramente as regiões de ataques devem ser extraídas de cada *frame* automaticamente. Os autores desenvolveram um programa de reconhecimento de padrões [Russell and Norvig 2003] capaz de extrair e alinhar

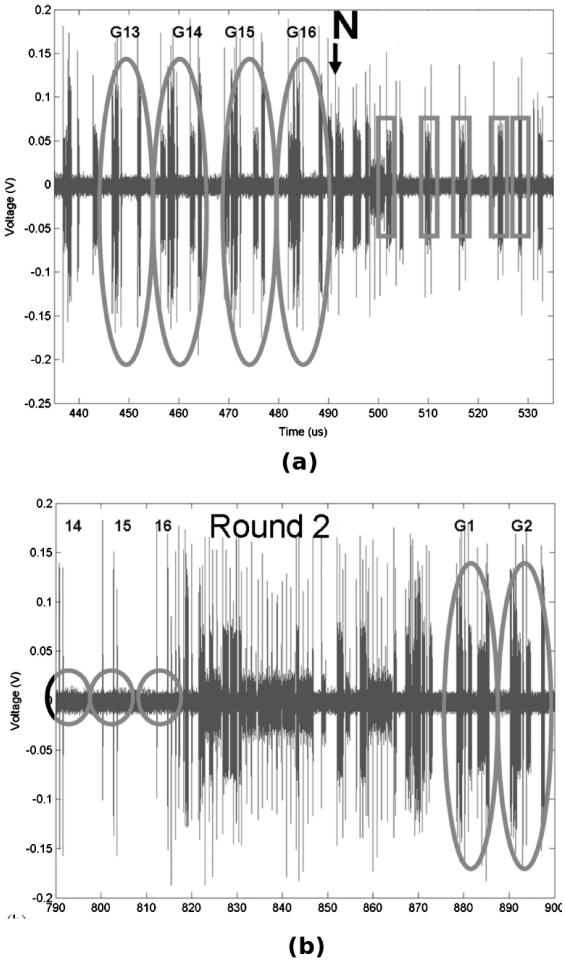


Figura 1.20. (a) Identificação do término da primeira rodada e (b) início da segunda rodada [Gebotys and White 2008].

as regiões de ataque de cada *frame*. Primeiramente o *software* aplica a função $V^*(t)$ em cada *frame*, tal que:

$$V^*(t) = \begin{cases} 1 & \text{se} \quad |V(t)| \geqslant 0.04V \\ 0 & \text{se} \quad |V(t)| < 0.04V \end{cases}$$

Para ilustrar o limiar, ao invés da captura ser realizada sobre uma implementação completa do AES, mas sobre uma versão *truncada* do AES, (o programa utiliza a função $optAEStrunc$ que realiza acesso apenas à região $t[0]$). As figuras 1.21 (a) e 1.21 (b) correspondem respectivamente a um *frame* de execução dessa versão e ao seu *limiar*.

Após o *thresholding*, a região de acesso $t[0]$ é extraída para realização do ataque. O *thresholding* é transformado em regiões de grande atividade (regiões escuras) e regiões de baixa atividade (longos períodos de voltagens nulas). Na região de interesse $t[0]$ existem diversos valores de tensão 0 e 1 misturados. A maior quantidade de zeros contíguos nessa região determina o limitante inferior do *parâmetro de tolerância*. A menor quantidade de zeros antes e depois da região de interesse definem o limitante superior do *parâmetro de*

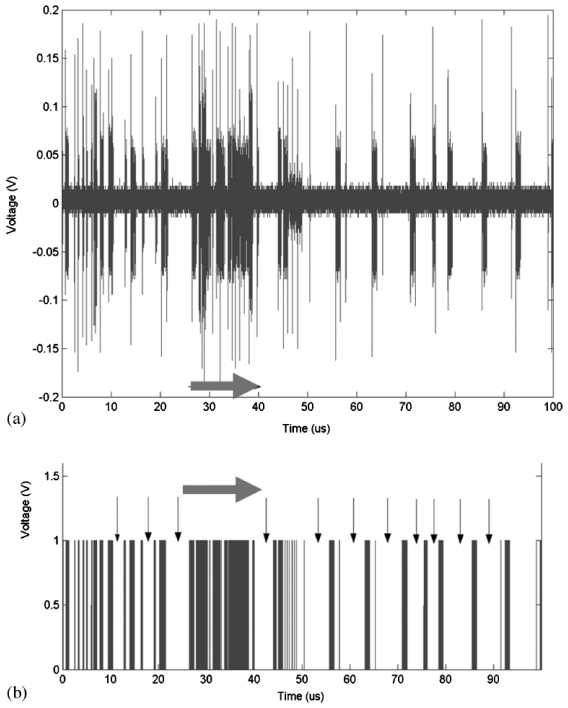


Figura 1.21. (a)Área de ataque extraida do *frame* 3 e (b) resultado da DFA aplicado sobre os sinais obtidos da versão *truncada* do AES[Gebotys and White 2008].

tolerância. O *parâmetro de tolerância* é um conjunto entre esses dois limitantes. O *frame* de 0 e 1 é transformado em regiões de baixa e alta atividade usando esse *parâmetro de tolerância* (nesse caso 1000).

Após a realização do truncamento, os valores de tensão são separados do seguinte modo:

- Uma região contígua com 1000 ou mais zeros no *frame* é transformada em uma região de baixa atividade.
- Regiões com menos do que 1000 zeros são inseridas em regiões vizinhas de alta atividade.

Desse modo ao invés de serem analisados 50 mil valores individuais de tensão, o programa de reconhecimento de padrões passa a lidar com 10 regiões de baixa/alta atividade, como mostra a tabela 1.5. Para os dados obtidos na caracterização, requer-se que:

- Regiões de baixa atividade não possuem de 2000 à 3000 amostragens.
- Regiões de alta atividade vizinhas às de baixa atividade tenham no mínimo 5000 amostragens.

Seguindo os critérios acima o terceiro índice da Tabela 1.5 é o escolhido.

Escolher uma área estática dentro região de ataque

Index	Inicio	Fim	Limitante inferior	Limitante superior
01	5265	6434	1169	1973
02	8371	9558	1187	1217
03	10775	13242	2467	6773
04	20015	22020	2005	3222
05	25242	27866	2624	1069
06	28935	31578	2643	1061
07	32639	35472	2833	515
08	35987	37742	1755	327
09	38069	39270	1201	513
10	39783	42751	2968	514
11	43265	45770	2505	907

Tabela 1.5. *Thresholded Zeros* para aquisição do frame.

Testes para o *software* são desenvolvidos para extrair a região de alta atividade de interesse, a qual é identificada pelo limitante inferior imediatamente antes da mesma e de largura do acesso $t[0]$ de interesse. Neste exemplo, regiões de baixa atividade possuem no mínimo 2000 tensões nulas e no máximo 3000, com as regiões de alta atividade de no mínimo 5000 termos. Se mais de uma região atende a esse critério, o adversário deve escolher a primeira detectada. A seção N foi escolhida para a análise por ocorrer no final de um *frame* (logo era uma forte candidata a corresponder, a uma região de leitura da tabela).

Foram analisados 32 frames e cada um deles foi dividido em $Areas_i$. Alguns deles não apresentaram todas as áreas; entretanto, a área N está presente em todos os frames sendo por isso escolhida para o ataque.

Sexta etapa

Foram extraídas 100 amostras imediatamente ao fim da região de ataque e 100 amostras depois da região de ataque, criando apenas 300 amostras por frame. A Figura 1.21 (a) corresponde aos sinais extraídos da região N.

Frequency-based DEMA

Finalmente, é aplicado o *frequency-based DEMA* sobre o conjunto de sinais extraídos da seção N, revelando corretamente a chave secreta (Figura 1.21 (b)). Baseado nesse ataque sobre a caracterização do PDA (que utilizou a versão truncada do AES), o adversário pode aplicar o ataque sobre a versão real do algoritmo criptográfico.

```
public void optAES(byte[] in, byte[] out)
{
wCount = 0; Copy.copy(state, in); t[0] = 0; t[1] = 0; t[2] = 0; t[3] = 0;
AddRoundKey(state); // xor with expanded key
for (int round = 1; round < Nr; round++)
{   t[0] = tab.Te0(state[0][0]) ^ tab.Te1(state[1][1]) ^ tab.Te2(state[2][2]) ^ tab.Te3(state[3][3]);
    t[1] = tab.Te0(state[1][0]) ^ tab.Te1(state[2][1]) ^ tab.Te2(state[3][2]) ^ tab.Te3(state[0][3]);
    t[2] = tab.Te0(state[2][0]) ^ tab.Te1(state[3][1]) ^ tab.Te2(state[0][2]) ^ tab.Te3(state[1][3]);
    t[3] = tab.Te0(state[3][0]) ^ tab.Te1(state[0][1]) ^ tab.Te2(state[1][2]) ^ tab.Te3(state[2][3]);
    state[0][0] = (byte) (t[0] >> 24);  state[1][0] = (byte) (t[0] >> 16);
    state[2][0] = (byte) (t[0] >> 8);   state[3][0] = (byte) (t[0]);
    state[0][1] = (byte) (t[3] >> 16);  state[1][1] = (byte) (t[3] >> 8);
    state[2][1] = (byte) (t[3]);        state[3][1] = (byte) (t[3] >> 24);
    state[0][2] = (byte) (t[2] >> 8);   state[1][2] = (byte) (t[2]);
    state[2][2] = (byte) (t[2] >> 24);  state[3][2] = (byte) (t[2] >> 16);
    state[0][3] = (byte) (t[1]);        state[1][3] = (byte) (t[1] >> 24);
    state[2][3] = (byte) (t[1] >> 16);  state[3][3] = (byte) (t[1] >> 8);
    AddRoundKey(state); }// xor with expanded key

  t[0] = (tab.Te4(state[2][3]) & 0xff0000L) ^ (tab.Te4(state[0][0]) & 0xff000000L) ^
(tab.Te4(state[0][2]) & 0xff00L) ^ (tab.Te4(state[2][1]) & 0xffL);
  t[1] = (tab.Te4(state[3][2]) & 0xff0000L) ^ (tab.Te4(state[1][3]) & 0xff000000L) ^
(tab.Te4(state[1][1]) & 0xff00L) ^ (tab.Te4(state[3][0]) & 0xffL);
  t[2] = (tab.Te4(state[0][1]) & 0xff0000L) ^ (tab.Te4(state[2][2]) & 0xff000000L) ^
(tab.Te4(state[2][0]) & 0xff00L) ^ (tab.Te4(state[0][3]) & 0xffL);
  t[3] = (tab.Te4(state[1][0]) & 0xff0000L) ^ (tab.Te4(state[3][1]) & 0xff000000L) ^
(tab.Te4(state[3][3]) & 0xff00L) ^ (tab.Te4(state[1][2]) & 0xffL);
  state[0][0] = (byte) (t[0] >> 24); state[1][0] = (byte) (t[1] >> 8);
  state[2][0] = (byte) (t[2] >> 24); state[3][0] = (byte) (t[3] >> 8);
  state[0][1] = (byte) (t[2] >> 16); state[1][1] = (byte) (t[3]);
  state[2][1] = (byte) (t[0] >> 16); state[3][1] = (byte) (t[1]);
  state[0][2] = (byte) (t[0] >> 8); state[1][2] = (byte) (t[1] >> 24);
  state[2][2] = (byte) (t[2] >> 8); state[3][2] = (byte) (t[3] >> 24);
  state[0][3] = (byte) (t[2]); state[1][3] = (byte) (t[3] >> 16);
  state[2][3] = (byte) (t[0]); state[3][3] = (byte) (t[1] >> 16);
  AddRoundKey(state); Copy.copy(out, state); } // xor with expanded key
private void AddRoundKey(byte[][] state)  // AddRoundKey: xor a portion of expanded key with state
{ for (int c = 0; c < Nb; c++)
    for (int r = 0; r < 4; r++)
      state[r][c] = (byte)(state[r][c] ^ rkey[rkeyCount++]); }
```

Figura 1.22. Implementação em Java do AES [Gebotys and White 2008].

Ataque real sobre o PDA

As sete etapas anteriores (Figura 1.14(a)) nos forneceçam o método adequado para identificar a região N como o alvo. Agora o adversário pode aplicar sobre a versão completa do AES similiarmente, porém analisando apenas os sinais da região de ataque (Figura 1.14 (b)). A versão completa do AES executada novamente utilizando como chaves secretas 92, 227, 61 e 158.

A Figura 1.23 (a) mostra o quarto *frame* capturado na execução para $k = 2$. Nela existem quatro regiões similares aos acessos de memória ($t[0], t[1], t[3]$ e $t[4]$) da versão truncada do AES. Por fim a Figura 1.24 apresenta as quatro chaves mencionadas correta-

mente recuperadas.

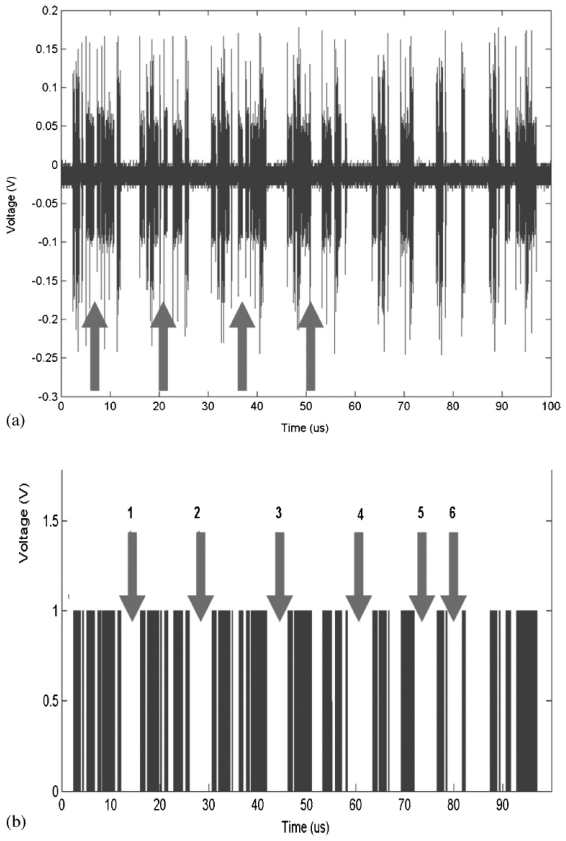


Figura 1.23. (a)Área de ataque extraida do *frame* 4 e (b) resultado da DFA aplicado sobre a os sinais obtidos da versão *completa* do AES [Gebotys and White 2008].

1.4.4. Análise de falhas sobre multiplicação de pontos em curvas elípticas

Trabalhos anteriores [Ciet and Joye 2005] mostraram como seria possível através de falhas no Algoritmo 1.7 gerar uma curva falha $\tilde{E}$ de modo que o cálculo de $\tilde{Q} \leftarrow k\tilde{P}$ fornecesse informações que viabilizassem a resolução do problema do logaritmo discreto e permitisse a descoberta de k. Contudo a medida de defesa contra esse ataque, mostrada no Algoritmo 1.8, consiste simplesmente em verificar se o resultado obtido permanece na curva $E(\mathbb{F})$. A seguir será apresentado uma metodologia desenvolvida por Johannes Blömer, Martin Otto e Jean-Pierre Seifert em 2006 [Blömer et al. 2004] na qual é possível inserir falhas de modo que o resultado de $Q \leftarrow kP$ pertença a $E(\mathbb{F})$ e a chave privada k seja descoberta indutivamente.

Fundamentos

Os resultados apresentados por [Boneh et al. 2001b] foram utilizados [Blömer et al. 2004] para definir a quantidade de multiplicações errôneas necessárias para determinar corretamente a chave privada da seguinte forma:

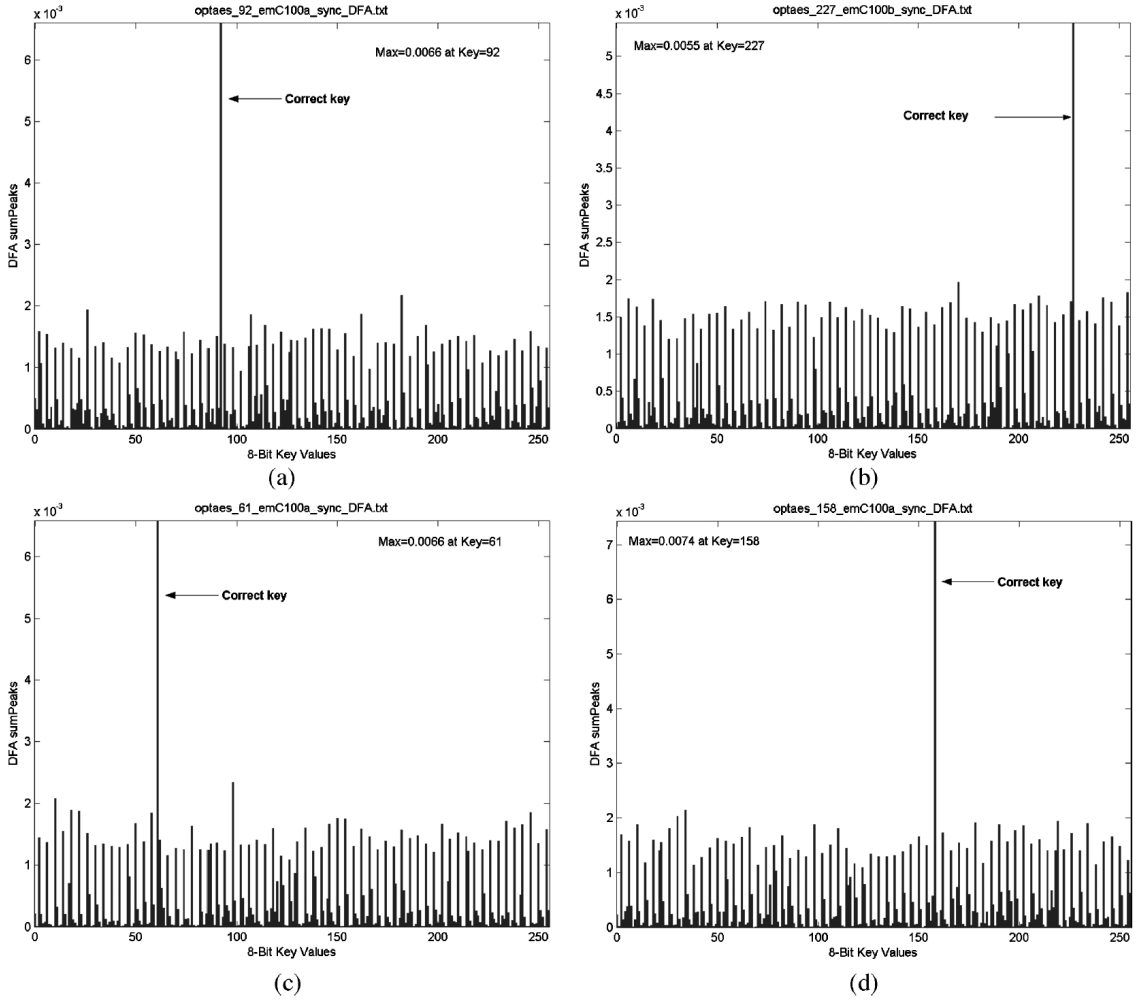


Figura 1.24. Chaves secretas (a) $k = 92$**, (b)** $k = 227$**, (c)** $k = 61$ **e (d)** $k = 158$ **[Gebotys and White 2008].**

Algoritmo 1.7. Multiplicação de pontos de curvas elípticas 2

Entrada: Inteiro positivo k, $P \in E(\mathbb{F})$

Saída: kP

01. Calcular $(k_{n-1}, k_{n-2}, ..., k_0) \leftarrow NAF_2(k)$
02. $Q_n \leftarrow \mathcal{O}$
03. De $i \leftarrow n-1$ até 0 faça
04. $\quad Q'_i \leftarrow 2Q_{i+1}$
05. $\quad$ Se $k_i = 1$ então
06. $\quad\quad Q_i \leftarrow Q'_i + P$
07. $\quad$ Ou se $k_i = -1$ então
08. $\quad\quad Q_i \leftarrow Q'_i - P$
09. $\quad$ Senão
10. $\quad\quad Q_i \leftarrow Q'_i$
11. Se $Q_0 \notin E(\mathbb{F})$ então
12. $\quad Q_0 \leftarrow \mathcal{O}$
13. Retornar Q_0

Seja uma curva elíptica $E(\mathbb{F}_p)$, para $p > 3$ primo. em um sistema de coordenadas projetivas $(x : y : z) \in \mathbb{F}_p^3$:

$$y^3 z \equiv x^3 + Axz^2 + Bz^3 \bmod p,$$

sendo o ponto no infinito representado por $(0 : 1 : 0)$. Definidas as operações de adição e multiplicação de ponto para essa equação, temos:

$$\forall P, Q \in E(\mathbb{F}_p) : P + Q = (0 : 1 : 0) \rightarrow P = (x : y : z) \; e \; Q = -P = (x : -y : z).$$

No Algoritmo 1.8 vemos que a multiplicação $Q = kP$ ocorre da esquerda para a direita. Os valores parcialmente calculados de Q até a iteração i são armazenados em uma variável temporária Q'_i. Desse modo, em uma execução correta da multiplicação escalar, o valor de Q expresso em função de Q'_i é dado por:

$$Q = 2^i Q_i + \sum_{j=0}^{i-1} 2^j k_j P$$

Metodologia do ataque

Estudos realizados por [Boneh et al. 2001a] constataram o seguinte fato:

Seja $x = (x_1, x_2, ..., x_{n-1}, x_n) \in \{0,1\}$e M um conjunto composto por todos os intervalos contíguos de comprimento $m < n$ em x. Se escolhermos $c = \frac{n}{m} \log 2n$ bits de x de forma aleatória, então a chance de cada intervalo de M conter ao menos um determinado bit é de, no mínimo, 50%. Desse modo a chave privada k será recuperada em pedaços de $r \in [1, m]$ bits, sendo $2^m < \#E(\mathbb{F}_p)$ a quantidade de trabalho *offline* aceitável.

O ataque consiste em causar uma falha em uma das iterações da multiplicação de modo a transformar o valor de Q'_i para $-Q'_i \in E(\mathbb{F}_p)$. Nesse sistemas de coordenadas essa tarefa se tornaria mais simples, uma vez que seria necessário apenas trocar o sinal da coordenada y do ponto. Se o adversário for capaz de causar uma falha tal que o durante uma iteração i do Algoritmo 1.7 o valor do bit k_i seja invertido, ao final do cálculo seria obtido um ponto expresso por

$$\tilde{Q} = -2^i Q'_i + \sum_{j=0}^{i} k_j 2^j P$$

Logo, isolando Q'_i na expressões que definem Q e $\tilde{Q}$ temos:

$$\tilde{Q} = -Q + 2L_i(k)$$

sendo o termo $L_i(k) = \sum_{j=0}^{i} k_j 2^j P$ é a parte desconhecida da equação.

O Algoritmo 1.8 descreve os passos para a recuperação da chave privada k de comprimento n. Supomos que $(k_0,k_1,...,k_s,x_{s+1},...,x_{s+r})$ corresponde a um $NAF(k)$ válido e $k_{n-1}=1$, sendo a parcela $(k_0,k_1,...,k_s)$ correspondente aos $s+1$ bits menos significativos da chave conhecidos. O termo *Zero Block Failures* indica o fato de que erros em blocos de zeros não serão detectáveis como erros dentro de um bloco. Para qualquer s, se $k_s = k_{s+1} = ... = k_{s+r} = 0$ então $L_s(k) = L_s(k) = ... = L_s(k) = 0$. Como $\tilde{Q}_1 = -Q + 2L_s$ e $\tilde{Q}_2 = -Q + 2L_{s+r}$ teriam o mesmo valor, seria impossivel determinar a quantidade de zeros caso eles ocorressem na parte caudal de um bloco; portanto, para que o algoritmo opere corretamente, os padrões testados precisam terminar com ± 1.

Algoritmo 1.8. Ataque de mudança se sinal sobre Q'_i.
Entrada: $P \in E(\mathbb{F})$, $k = (k_{n-1},k_{n-2},...,k_0) \in [1, ord(P)]$
Saída: $kP \in E(\mathbb{F})$

01.# FASE 1: Criar saídas falhas
02. $c \leftarrow \frac{c}{m} \log 2n$
03. Criar c saídas falhas induzindo SCF em Q'_i
04. Coletar um conjunto S de diversas saídas falhas $\tilde{Q}$
05.# FASE 2: Recuperação indutiva dos bits da chave secreta
06. $s \leftarrow -1$
07. Enquanto $s < n-1$ faça
08. $\quad L \leftarrow 2\sum_{j=0}^{s} k_j 2^j P$
09. $\quad$ Para todos os comprimentos $r \in [1,m]$ faça
10. $\quad\quad$ Para todos NAF $x = (x_{s+1},x_{s+2},...,x_{s+r})$ com $x_{s+r} \neq 0$ faça
11. $\quad\quad\quad$ # Calcular e verificar o candidato T_x
12. $\quad\quad\quad T_x \leftarrow L + 2\sum_{j=s+1}^{s+r} x_j 2^j P$
13. $\quad\quad\quad$ Para todos os $\tilde{Q} \in S$ faça
14. $\quad\quad\quad\quad$ se $T_x - \tilde{Q} = Q$ então
15. $\quad\quad\quad\quad\quad (k_{s+1},k_{s+1},...,k_{s+r}) \leftarrow (x_{s+1},x_{s+2},...,x_{s+r})$
16. $\quad\quad\quad\quad\quad s \leftarrow s+r$
17.# Caso *Zero Block Failure*
18. Se nenhum dos candidatos satisfizer a fase de verificação, então
19. $\quad$ assumir que $k_{s+1} \leftarrow 0$ e $s \leftarrow s+1$
20. Se $Q = kP$ então
21. $\quad$ retornar k
22. Senão
23. $\quad$ retornar *falha*

Medidas preventivas

A medida preventiva aplicada em estudos anteriores [Boneh et al. 2001a] de falhas sobre curvas elípticas não se mostrou eficiente para esse ataque. Tomando a curva elíptica original $E(\mathbb{F}_p)$, a nova medida preventiva proposta consiste em escolher um número primo pequeno t (60 a 80 bits) para formar uma segunda curva elíptica $E(\mathbb{F}_t)$ que será combinada com a original a fim de criar uma terceira curva $E(\mathbb{F}_{pt})$ sobre a qual serão calculadas

as multiplicações de ponto. Os parâmetros A_{pt} e B_{pt} da curva elíptica no sistema de coordenadas projetivas serão definidos como:

$$\begin{aligned} A_{pt} &\equiv A_p \pmod{p} \\ A_{pt} &\equiv A_t \pmod{t} \\ B_{pt} &\equiv B_p \pmod{p} \\ B_{pt} &\equiv B_t \pmod{t} \end{aligned}$$

e podem ser facilmente computados com auxílio do Teorema Chinês do Resto. Analogamente, um novo ponto gerador P_{pt}, sendo que a ordem de $P_t \in E(\mathbb{F}_t)$ suficientemente larga para evitar ataques de força bruta. No Algoritmo 1.9 vemos que, caso ocorra uma falha no cálculo de $R \leftarrow kP_{pt}$, a congruência $R \equiv kP_t \pmod{t}$ deixaria de ser válida e portando seria detectada uma falha de execução.

Algoritmo 1.9. Multiplicação de pontos de curvas elípticas com tratamento de falhas

Entrada: Inteiro positivo k, $P \in E(\mathbb{F})$
Saída: kP

Inicialização antes da execução do algoritmo
01. Escolher um primo t e uma curva elíptica $E(\mathbb{F}_t)$
02. Determinar os parâmetros da curva combinada $E(F_{pt})$

Multiplicação
03. $Q \leftarrow kP_{pt} \in E(\mathbb{F}_{pt})$ # Usando Algoritmo 1.7
04. $R \leftarrow kP_t \in E(\mathbb{F}_t)$ # Usando Algoritmo 1.7
05. Se $R \neq Q \pmod{t}$ então
06. retornar *falha*
07. Senão
08. retornar $Q \in E(\mathbb{F}_p)$

1.4.5. Análise de falhas sobre assinatura digital RSA

Como mostrado, anteriormente, a exponenciação modular é a base do funcionamento do RSA sendo oss algoritmos que a implementam os mais diversos. O expoente pode ser percorrido da direita para a esquerda ou vice-versa, a redução modular pode ser simples ou utilizar o Teorema Chinês do Resto [Stinson 2002] para torná-la mais rápida, etc. O ataque aqui demonstrado foi aplicado sobre uma implementação percorrendo o expoente da esquerda para a direita (Algoritmo 1.10).

Algoritmo 1.10. Exponenciação binária da esquerda para a direita
Entrada: Inteiros positivos m, e e n
Saída: $m^e \bmod n$

```
# e = (e_{t-1}, e_{t-2}, ..., e_1, e_0)
01. r ← 1
02. De i ← t − 1 até 0 faça
03.     r ← r^2 mod n
04.     Se e_i = 1 então
05.         r ← r × m mod n
06. Retornar r
```

Sabendo que o dispositivo embarcado utiliza a implementação mencionada, o adversário pode forçá-lo saltar um *squaring* (linha 3 do Algoritmo 1.10) através de injeção de falhas. Como não é possível saber o exato instante em que a instrução está sendo executada, outras instruções podem sofrer um salto indesejado. Assim, o adversário também deve ser capaz de verificar se a instrução correta não foi executada.

Seja a assinatura do resumo $H(m)$ de uma mensagem m dada por

$$Sig \leftarrow \prod_{i=0}^{t} H(m)^{d_i \times 2^i} \bmod n$$

Para $k \in \{0,1,...,t\}$, ao saltarmos a $(t-k+1)$-ésima iteração da exponenciação modular, obteremos uma assinatura falha Sig_k tal que:

$$Sig_k \leftarrow \prod_{i=k+1}^{t} H(m)^{d_i \times 2^{i-1}} \times \prod_{i=0}^{k} H(m)^{d_i \times 2^i} \mod n$$

Como as falhas podem ser injetadas em qualquer ponto da execução do programa, o adversário precisa:

1. Determinar um caso base de assinatura falha facilmente verificável.
2. Que a verificação da falha informe ao menos um bit da chave secreta.
3. Que a assinatura falha seja obtida saltando apenas um *squaring*.
4. O caso base seja o passo inicial de um método indutivo.

Sendo que o expoente d é percorrido da esquerda para a direita, a única assinatura que poderia ser verificada dessa maneira é Sig_0 , pois:

$$Sig \leftarrow \begin{cases} (Sig_0)^2 \mod n & \text{se} \quad d_0 = 0, \\ H(m)^{-1} \times (Sig_0)^2 \mod n & \text{se} \quad d_0 = 1 \end{cases}$$

Assim, Eve injetaria falhas até que em algum momento uma das igualdades acima fosse verificada e automaticamente revelando o valor correto de d_0. O processo de injeção de saltos seria reiniciado até que todos os $t, t-1, t-2, ..., 1$ bits restantes fossem indutivamente revelados através da verificação da seguinte igualdade:

$$Sig_k \leftarrow \begin{cases} Sig_{k-1} \mod n & \text{se} \quad d_k = 0 \\ H(m)^{2^{k-1}} \times Sig_{k-1} \mod n & \text{se} \quad d_k = 1 \end{cases}$$

Medidas preventivas

Medidas preventivas contra ataques por inserções usualmente consistem em inserir códigos nas implementações a fim de inviabilizá-los. Porém, uma análise mais profunda revela que a injeção de falhas pode ser estendida contra as próprias medidas de defesa. Conseqüentemente não foi encontrada na literatura nenhuma medida de defesa que não pudesse ser invalidada.

Inserção de Código Redundante

A medida de defesa contra análise simples de potência consistia em inserir operações de modo que a multiplicação deixasse de ser relacionada ao valor de um bit do expoente, passando a ser sempre executada de maneira redundante. Mas o foco desse ataque é a saída gerada pelo dispositivo; então, essa medida de defesa em nada o afeta.

Ofuscar a mensagem

Vamos supor, por exemplo, que a entidade assine o próprio texto claro e não seu resumo. Primeiramente, a entidade escolhe valores aleatórios r_0 e r_i tais que $r_0^{-1} = r_i^d \mod n$. Então, ele ofusca a mensagem do seguinte modo:

$$\begin{aligned} \mu &= m \times r_i \mod n, \\ c &= \mu^d \mod n. \end{aligned}$$

Porém, a inserção de falhas poderia ser estendia ao cálculo de μ, impedindo que a mensagem fosse multiplicada por r_i e permitindo que o ataque descrito previamente fosse aplicado com sucesso.

Ofuscar o expoente

Aqui a entidade usa para cifrar uma mensagem o expoente $d' = d + r \times \phi(n)$, sendo r aleatório. Aqui existem duas possibilidades:

- O valor de d' é armazenado no mesmo registrador que d; então, basta inserir uma falha e impedir que a atribuição $d' \leftarrow d$ ocorra.

- Se d' é armazenado em um registrador diferente, o adversário pode impedir que o cálculo $r \times \phi(n)$ seja realizado, fazendo com que $d' \leftarrow d$.

1.4.6. Análise de tempo sobre preditores de saltos

Atualmente ataques utilizando canais secundários de informação tipicamente são aplicados sobre dispositivos embarcados. Essas plataformas são os principais alvos de ACS porque, comparadas com processadores convencionais, seu hardware de fácil acesso permite que as medidas de suas grandezas físicas sejam efetuadas silenciosamente, i.e. em sua maioria não são invasivas e não interferem na execução dos algoritmos criptográficos. Acreditava-se que a análise de tempo não era tão ameaçadora quanto os outros ataques porque a duração das instruções tanto em smart cards como computadores convencionais são da ordem de nanosegundos. Logo seria inviável estabelecer uma relação de tempo entre operando e operador.

A arquitetura intrínseca aos processadores convencionais (PowerPC, Cell, Intel x86, ARM, etc.) impede medições desse caráter. É impossível acessarmos não invasivamente valores contidos em registradores ou memória cache referentes a programas distintos que estão sendo executados em CPUs convencionais. Além disso essas plataformas executam sistemas operacionais capazes de executar programas concorrentemente (Windows, Linux, Symbian, PalmOS, etc.) e a troca de contexto desses processos faz com que valores imprecisos sejam obtidos nas medições. Poucos trabalhos até 2003 [Brumley and Boneh 2003] abordaram ACS como um risco real contra essas plataformas de modo que essas eram tidas como seguras a essa categoria de ataque.

Paralelismo de instruções e preditor de saltos

Desde o surgimento dos primeiros processadores modernos, diversas técnicas de construção foram criadas visando rápidas melhorias no desempenho dos processadores. Na década de 1960 surgiram processadores com estágios distintos de execução (pipelined processors); na década de 1980 processadores capazes de executar instruções especulativamente [Hennessy and Patterson 2002]. Essas técnicas, que exploram paralalelismo de instruções (ILP ou *Instruction Level Parallelism*) necessitam de mecanismos sofisticados de predição dos saltos (in)condicionais executados durantes a execução do programa, de modo que a unidade de processamento praticamente não fique ociosa.

Estudos recentes [Jean-Pierre et al. 2006, Aciiçmez et al. 2007] demonstraram que seria possível, através de medições das diferenças dos tempos de acertos e erros das unidades de predição de salto, determinar a chave privada de um sistema RSA utilizando um processo espião [Milenkovic et al. 2004]. Apesar desse ataque consistir de análises de tempo, ele também é denominado *Branch Prediction Analysis*, uma vez que o tempo é utilizado para inferir valores do preditor de saltos.

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 um salto condicional, então 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 ele especula 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 tomada. 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, ela deve buscar a instrução do endereço alvo do salto na memória e entregá-la a 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 expulsos 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].

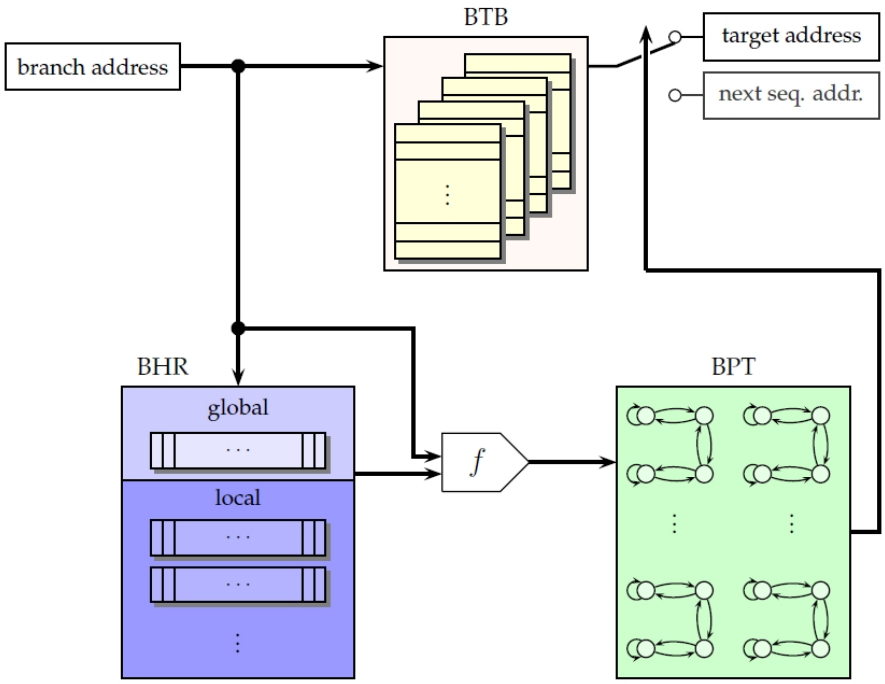


Figura 1.25. Unidade de predição de saltos [Jean-Pierre et al. 2006].

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 assumir que a implementação do RSA utilizou S&M (*Square-and-Multiply exponentiation algorithm*) e MM (*Montgomery Multiplication algorithm* [Hankerson et al. 2003, Denis 2006]) e o BTU possui um autômato finito de apenas dois estados: salto tomado ou não tomado.

Seja d a chave privada, vamos supor que o adversário conhece 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:

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

O adversário calcula o tempo médio de execução na multiplicação de Montgomery em cada conjunto M_i. Sendo $d_i = t, t \in \{0,1\}$, 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 caracter aleatório. Esse é o mesmo processo estatístico da análise diferencial de potência. 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 de simples 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.

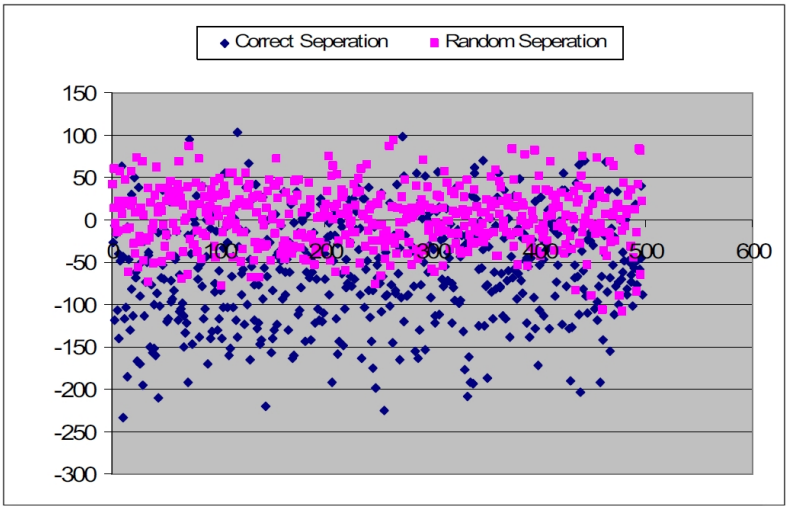
Forçando BPU à mesma predição assincronamente

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; conseqüentemente, sempre ocorrerá um *misspredictio*n 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 *misspredict*ion durante o cálculo de $m^2 \bmod N$.

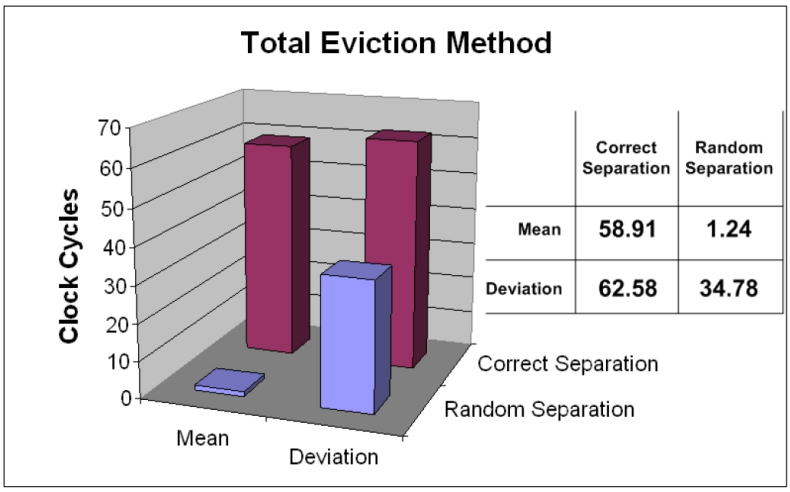
O processo espião remover do BTB o endereço alvo de salto dos seguintes modos:

1. (*Total Eviction Method*: todas as entradas do BTB são expulsas.
2. (*Partial Eviction Method*): um conjunto de entradas do BTB é expulso.
3. (*Single Eviction Method*): apenas endereço de interesse é expulso da tabela.

Obviamente o primeiro método é o de mais simples implementação (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 1.26. 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 1.26 (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 correta e a média das separações aleatórias, denotadas respectivamente por μ_{Y_i} e μ_{X_i}. Analizando todos os pares (μ_{Y_i}, μ_{X_i}), o adversário verifica qual deles teve a diferença mais significativa (Figura 1.26 (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 através da 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%.

Forçando BPU à mesma predição sincronamente

No ataque anterior, o adversário não precisava sincronizar o processo espião com a execução do programa de decriptação. Entretanto se ele fosse capaz de fazê-lo, poderia esvaziar o BTB apenas no passo imediatamente anterior a *iésima* exponenciação, tornando o processo muito mais eficiente. O adversário irá supor que a implementação do RSA utiliza o algoritmo S&M e se a sentença if foi usada como alvo do salto condicional.

O adversário executa o RSA para textos claros conhecidos e mede o tempo de execução. Em seguida reexecuta o programa aplicando imediatamente antes da *iésima* execução uma *single eviction* no BTB. Como o salto é tomado ou não de acordo com o valor de d_i, se ele o supor como tomado, então ocorrerá um *missprediction* e será percebido um atraso na segunda execução. Logo, o adversário pode determinar todos os bits da chave secreta d analisando o tempo de execução de cada iteração.

Trace-Driven Attack

Todas as abordagens anteriores foram focadas na medição dos tempos gastos pelos saltos do programa de decriptação, enquanto esta abordagem monitora os saltos do programa

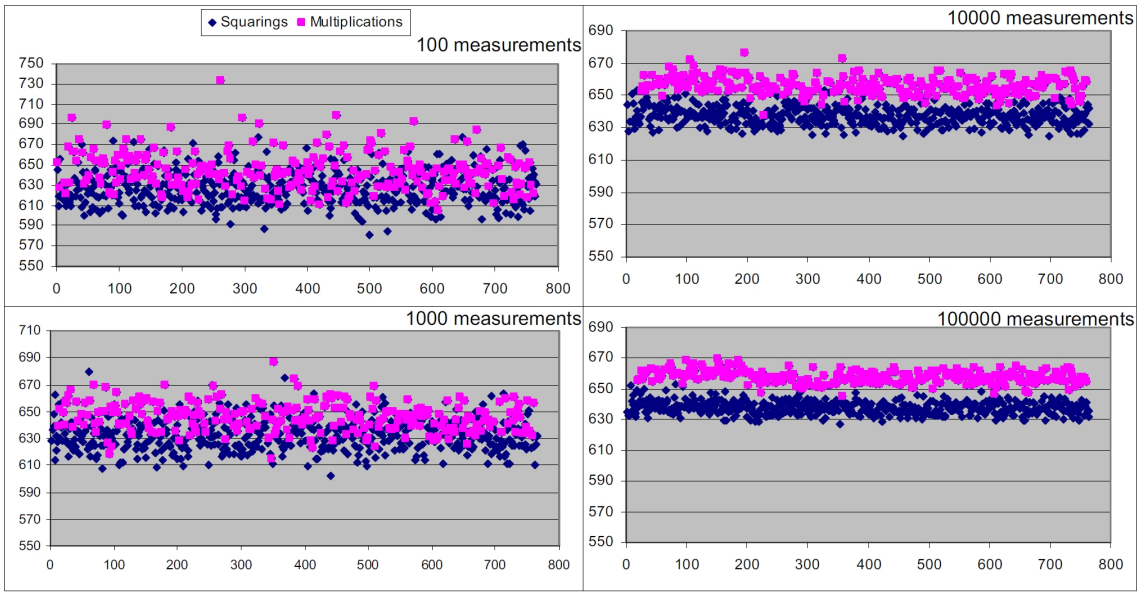


Figura 1.27. Conectando os *misses* induzidos no BTB e a diferença de tempo do S&M [Jean-Pierre et al. 2006].

espião. Iremos supor que inicialmente a CPU prevê o salto do decriptador como não tomado.

O adversário inicia seu programa antes do *software* criptográfico e continuamente executa saltos de modo que eles ocupem as mesmas entradas no BTB. Quando ocorrer um *misspredict*ion de um salto que deveria ser tomado, a CPU irá expulsar uma das entradas no BTB utilizadas pelo programa espião e inserir o endereço de salto do decriptador. Conseqüentemente, ao retomar a execução, o programa espião também sofrerá um *misspredict*ion. Dessa maneira, o adversário é capaz de determinar quando o BTB foi modificado pelo decriptador e determinar os bits da chave.

As medições de tempo foram realizadas no início da multiplicação de Montgomery, fornecendo o tempo de apenas uma das operações realizadas (*squaring* ou multiplicação). Sendo N a quantidade de amostras por *bit* da chave, quanto maior seu valor mais discrepantes serão as diferenças entre as duas operações, como mostra a Figura 1.27 com amostragens 100 a 100000. No caso de $N = 10000$, também é possível ver na Figura 1.28 os bits $d_i \in \{d_{89}, d_{104}\}$, demonstrando como a partir desse ponto é trivial determinar a chave secreta.

Melhorando o Trace-Driven Attack

Nos quatro ataques sobre o BPU mostrados até agora foi utilizada análise diferencial das medições de tempo, sendo necessárias quantidades enormes de amostragens a fim de determinar o correto valor da chave. Um segundo estudo do autor [Aciiçmez et al. 2007] demonstrou ser possível diminuir bruscamente a quantidade de amostragens necessárias no ataque *Trace-Driven*.

O ataque é realizado como um *Trace-Driven* usual, onde o programa espião ex-

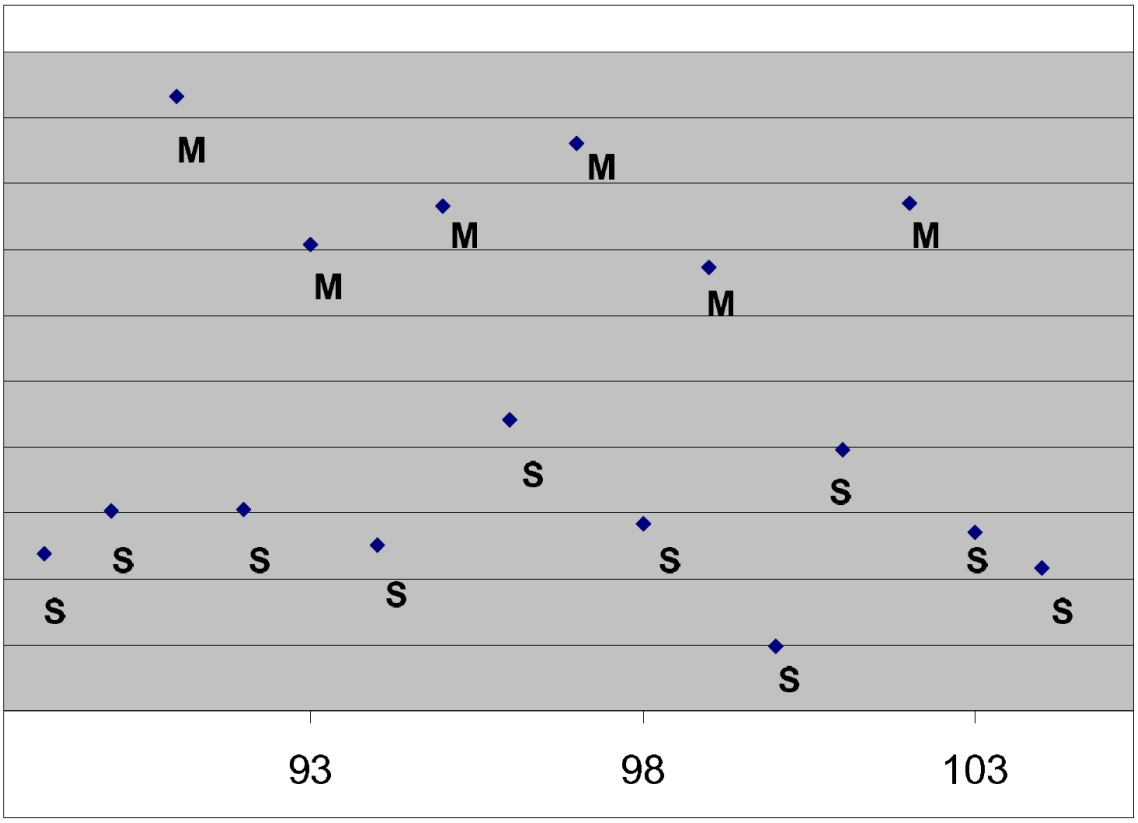


Figura 1.28. Diferenças entre S&M entre iterações 89 e 104 para 10 mil medições [Jean-Pierre et al. 2006].

ecuta uma seqüência fixa de t saltos para expulsar os endereços alvos do BTB. Os autores perceberam que, de acordo com a arquitetura do processador, existe um t ótimo que pode ser experimentalmente determinado. Aumentando a duração de ciclos dos saltos do processo espião, as diferenças de tempo tornam-se muitos mais significativas e facilmente perceptíveis com uma amostragem muito menor, economizando de mil a 10 mil amostragens em relação ao *Trace-Drive Attack* original. Isso permite que a chave seja determinada com apenas *uma* medição por bit. A Figura 1.29 mostra como a precisão dos valores dos bits obtidos são influenciados pela quantidade de ciclos gastos pelo saltos do programa espião de modo que quanto maior a duração do salto, maior é a precisão dos valores dos bits.

Porém não pode ser esquecido que os aplicativos estão sendo executados em um sistema SMT, logo a troca e contexto dos processos pode eventualmente afetar as medições de tempo. Sob uma ótica estatística, se o advsersário executar algumas amostragens, algumas delas serão menos afetadas do que outras pela concorrência dos aplicativos.

A Figura 1.29 mostra quatro de dez amostragens independentes. Tomando a melhor amostragem, o adversário foi capaz de revelar corretamente 508 dos 512 bits da chave secreta.

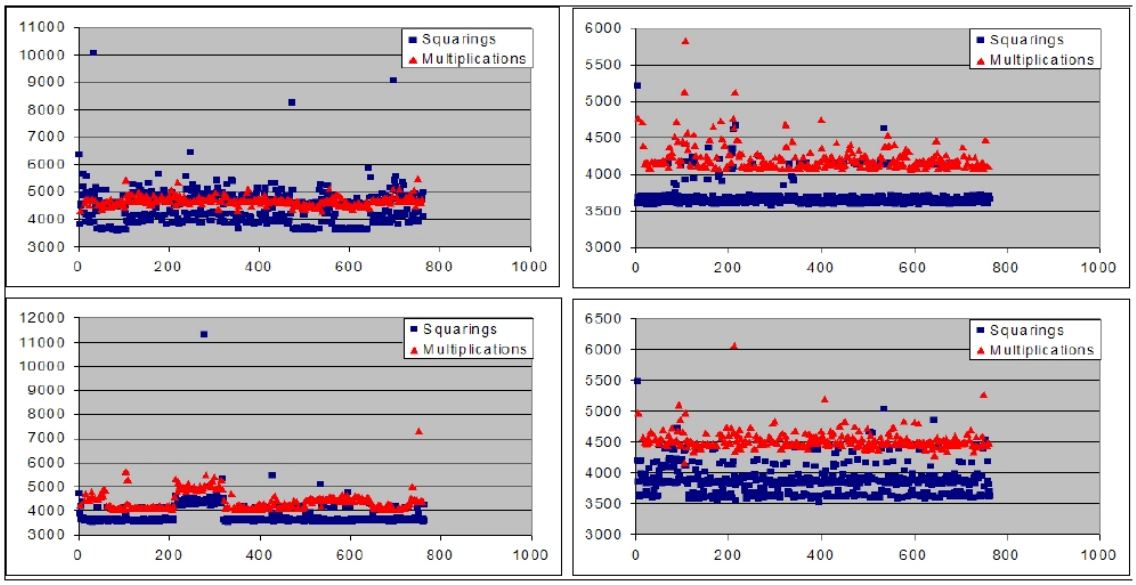


Figura 1.29. *Simple branch prediction* [Aciiçmez et al. 2007].

1.5. Considerações finais

Algoritmos criptográficos são conhecidos pela elevada quantidade de processamento em sua execução,. Assim, um dos grandes desafios em defesas contra ACS puramente via *software* é conseguir prover a garantia do não-vazamento de informações sem comprometer a eficiência dos aplicativos. Deve-se ficar atento para que as próprias medidas não possibilitem um vazamento de informação. Além disso, a implementação do algoritmo de defesa deve passar por uma análise criteriosa, principalmente no que se refere a otimizações realizadas pelo compilador. No caso de código reduntante, ele poderia ser simplesmte removido e esse fato passar despercebido pelo desenvolvedor.

Existe hoje uma grande proliferação de dispositivos embarcados ao nosso redor sem que nos demos conta. Eles estão presentes em automóveis, televisores, celular, cartões de bancos, PDAs, etc. e cada vez mais são capazes de comunicarem-se entre si. A principal questão à qual devemos ficar atentos não é o tanto fato de eles muitas vezes trocarem de dados sigilosos, mas sim o nosso desconhecimento sobre essas transações. Com a acessibilidade do hardware e baixo custo de equipamentos necessários para efetuar certos tipos ataque, é fundamental que medidas elaboradas de segurança criptográfica sejam utilizadas a fim de garantir a segurança e privacidade de nossos dados.

Possivelmente, devido ao fato das publicações utilizadas em nossa pesquisa serem recentes, não foram encontradas medidas eficazes de proteção contra os ataques de análise de preditor de salto, análise de falhas aplicadas na assinatura de RSA e análise eletromagnética do PDA Java. O ataque ao preditor de saltos mostrou-se o mais nocivo porque, ao contrário de todos os outros apresentados, não seria necessário acesso físico ao dispositivo de hardware para executá-lo. Entretanto não sabemos se ele seria aplicável às plataformas com múltiplas unidades de processamento [Akhter and Roberts 2006], pois não há garantias de que o processo invasor sempre seja alocado no mesmo núcleo de processamento que o processo criptográfico.

Agradecimentos

Gostaríamos de agradecer a Karina Magalhães, Fabio Piva e Diego Aranha pela ajuda e comentários na revisão do texto. Os erros e omissões restantes são de nossa total responsabilidade.

Referências

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.

Agrawal, D., Archambeault, B., Rao, J. R., and Rohatgi, P. (2003). The em side-channel(s). In *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 29–45, London, UK. Springer-Verlag.

Akhter, S. and Roberts, J. (2006). *Multi-core programming : increasing performance through software multi-threading*. Intel Press.

Blömer, J., Otto, M., and Seifert, J.-P. (2004). Sign change fault attacks on elliptic curve cryptosystems. In *Fault Diagnosis and Tolerance in Cryptography 2006 (FDTC 06), volume 4236 of Lecture Notes in Computer Science*, pages 36–52. Prentice Hall.

Boneh, D., Demillo, R. A., and Lipton, R. J. (2001a). On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14:101–119.

Boneh, D., Demillo, R. A., and Lipton, R. J. (2001b). On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14:101–119.

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.

Ciet, M. and Joye, M. (2005). Elliptic curve cryptosystems in the presence of permanent and transient faults. *Des. Codes Cryptography*, 36(1):33–43.

Daemen, J. and Rijmen, V. (2002). *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

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

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on information Theory*.

El Gamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA. Springer-Verlag New York, Inc.

Gebotys, C. H. and White, B. A. (2006). Methodology for attack on a java-based pda. In *CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, pages 94–99, New York, NY, USA. ACM.

Gebotys, C. H. and White, B. A. (2008). Em analysis of a wireless java-based pda. *ACM Trans. Embed. Comput. Syst.*, 7(4):1–28.

Gong, L. and Ellison, G. (2003). *Inside Java(TM) 2 Platform Security: Architecture, API Design, and Implementation*. Pearson Education.

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

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

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.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209.

Milenkovic, M., Milenkovic, A., Milenkovic, A., and Kulick, J. (2004). Microbenchmarks for determining branch predictor organization. *Software Practice and Experience*, 34:465–487.

Miller, V. S. (1986). Use of elliptic curves in cryptography. In *CRYPTO '85: Advances in Cryptology*, pages 417–426, London, UK. Springer-Verlag.

Oren, Y. and Shamir, A. (2007). Remote password extraction from rfid tags. *IEEE Trans. Comput.*, 56(9):1292–1296.

Rivest, R., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *portal.acm.org*.

Russell, S. J. and Norvig (2003). *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall.

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

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

Stinson, D. R. (2002). *Cryptography: Theory and Practice, Second Edition*. Chapman & Hall/CRC.

Zabala, E. (2009). Rijndael Cipher: 128-bit Version (Data-Block and Key) Encrypton. Available at http://www.cs.bc.edu/~straubin/cs381-05/blockciphers/rijndael_ingles2004.swf.