Implementação para Multiplicação por Escalar em Curvas Elípticas sobre $\mathbb{Z}_p$

Pedro Lara, **Fábio Borges**,
Coordenação de Sistemas e Redes, CSR, LNCC,
25651-075, Petrópolis, RJ
E-mail: pcslara@lncc.br, borges@lncc.br

Resumo: *Desde a introdução da criptografia assimétrica por Whitfield Diffie e Martin Hellman em 1976, o Problema do Logaritmo Discreto (PLD) foi utilizado de algumas formas diferentes. Uma variação do PLD está sobre o grupo formado pelos pontos de uma curva elíptica. A grande vantagem de criptossistemas baseados em curvas elípticas é a redução do tamanho da chave criptográfica. Este artigo descreverá a implementação da operação de multiplicação por escalar em curvas elípticas sobre o corpo primo $\mathbb{Z}_p$.*

Palavras-chave: *Criptografia, Curvas Elípticas, Segurança da Informação*

1 Introdução

Antes de 1976 se alguém quisesse trocar informações confidenciais era primordial combinar uma chave em um canal de comunicação que fosse considerado seguro. Com a publicação do artigo [3] por W. Diffie e M. Hellman este fato não seria necessário. Suponha que dois usuários, Alice e Bob, queiram estabelecer uma chave secreta compartilhada. Alice seleciona aleatoriamente um inteiro secreto a tal que $a \in \mathbb{Z}_p$ e envia para Bob $P_A = g^a \mod p$, sendo p um primo e g um gerador do grupo cíclico $\mathbb{Z}_p^*$. Analogamente, Bob escolhe um inteiro secreto $b \in \mathbb{Z}_p$ e envia à Alice $P_B = g^b$. Agora ambos usam sua chave secreta para obter uma chave secreta compartilhada $K = (P_A)^b = (P_B)^a = g^{ab}$, observe que a segurança deste protocolo está baseada na dificuldade computacional de calcular a dado g e p. Quando estas variáveis são relativamente grandes este problema se torna inviável, e é conhecido como Problema do Logaritmo Discreto (PLD). Em 1985, V. Miller [7] e N. Koblitz [5], propuseram independentemente a aplicação de curvas elípticas em criptografia assimétrica. A segurança deste método está, mais uma vez, baseada no PLD. Neste caso, isto significa que não existe um algoritmo eficiente para: dados dois pontos P, Q pertencentes a uma curva elíptica Ω encontrar um inteiro K tal que $Q = KP$. A principal vantagem deste método criptográfico é que o mesmo exige uma chave de comprimento consideravelmente menor que a chave usada em alguns clássicos da criptografia assimétrica, tais como o RSA, no entanto com segurança equivalente, ficando acessível a sistemas com restrições computacionais como *smart cards*. Neste artigo exporemos a experiência da implementação da principal e mais custosa operação aritmética na criptografia usando curvas elípticas.

2 Curvas Elípticas Sobre $\mathbb{Z}_p$

Seja o corpo $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, sendo p um primo qualquer. Todas as operações admitidas a seguir devem ser calculadas em função do resto da divisão por p, ou seja, todos os parâmetros e variáveis pertencem ao conjunto $\mathbb{Z}_p$.

Definição 2.1 *Uma curva elíptica sobre um corpo $\mathbb{Z}_p$ (assumiremos sempre que p é ≥ 3) é o lugar geométrico dos pontos $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ que satisfazem a equação*

$$y^2 + axy + by = x^3 + cx^2 + dx + e. \tag{1}$$

Podemos simplificar a equação (1) deixando na forma

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

com $a, b \in \mathbb{Z}_p$. Iremos definir uma fórmula para soma entre dois pontos pertencentes a uma curva elíptica [6], que será muito útil mais a frente. Se quisermos somar $P = (x_1, y_1)$ com $Q = (x_2, y_2)$ e seja $R = (x_3, y_3)$ o resultado desta soma, e se $P \neq -Q$, temos

$$x_3 = \lambda^2 - x_1 - x_2 \mod p,$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \mod p$$

onde

$$\lambda = \begin{cases} \frac{(y_2 - y_1)}{(x_2 - x_1)} & \mod p, \text{se } x_1 \neq x_2 \\ \frac{(3x_1^2 + a)}{(2y_1)} & \mod p, \text{se} x_1 = x_2 \text{ e } y_1 \neq 0. \end{cases} \tag{3}$$

3 Linguagens, Ferramentas e Algoritmos Utilizadas

As operações aritméticas básicas em curvas elípticas foram implementadas em `Java`. Esta linguagem dá o aparato para o desenvolvimento orientado a objeto. Este era um dos principais requisitos para a implementação exposta neste trabalho. Assim podemos ter uma API flexível a mudanças que provêm à oportunidade de criar e implementar componentes totalmente reutilizáveis usando o paradigma de orientação a objetos. O fato de ser uma linguagem multiplataforma motivou a escolha da linguagem `Java`. A grande popularidade que o `Java` alcançou na última década também teve um peso na escolha desta linguagem. Na implementação, foi necessário utilizar inteiros de tamanho arbitrário. Em curvas elípticas, se utiliza números de aproximadamente 160 bits, ou mais, para uma segurança aceitável. Na linguagem `Java` é nativo o uso da classe `BigInteger` [2] que permite trabalhar com inteiros grandes.

3.1 Algoritmo Binário para Multiplicação por Escalar

Nesta parte mostraremos uma técnica de multiplicação por escalar baseada no método de exponenciação binário. O algoritmo 1 mostra o funcionamento do algoritmo para multiplicação por escalar binário, na qual processa os bits de K da esquerda para direita (analogamente poderíamos ter a versão da direita para esquerda).

Algoritmo 1: Multiplicação por escalar (método binário: versão esquerda para direita)

Entrada: Um inteiro $K = \sum_{i=0}^{j} 2^i k_i$ onde $k_i \in \{0, 1\}$ (representação em base binária), um ponto $P \in \Omega$ e a curva Ω
Saída: O ponto $K \cdot P \in \Omega$

```
1 início
2 |  Q ← ∞;
3 |  para i = j até 0 faça
4 |  |  Q ← 2 · Q;
5 |  |  se k_i = 1 então
6 |  |  |  Q ← Q + P;
7 |  retorna Q ∈ Ω;
8 fim
```

A densidade média da representação binária de K é $\frac{1}{2}$, logo o tempo de execução esperado do algoritmo 1 é de aproximadamente $(j + 1)/2$ adições de pontos (representaremos por α o

tempo de execução para 1 adição) e $j+1$ duplicações de pontos (que será representado por β) é denotado [4] por

$$\left(\frac{j+1}{2}\right)\alpha + (j+1)\beta.$$

Para os testes de desempenho utilizamos as curvas recomendadas em [1] pelo *National Institute of Standards and Technology* (NIST). Neste caso, multiplicamos a ordem do ponto pelo próprio ponto. As curvas indicam o número de bits do primo que define o corpo $\mathbb{Z}_p$. Por exemplo, a curva P-256, está definida sobre um corpo $\mathbb{Z}_p$ cujo o primo p possui 256 *bits*. O processador utilizado foi um Intel Pentium 4 ® com freqüência 2.40GHz. Os gráficos 1 e 2 mostram os resultados para os algoritmos binários.

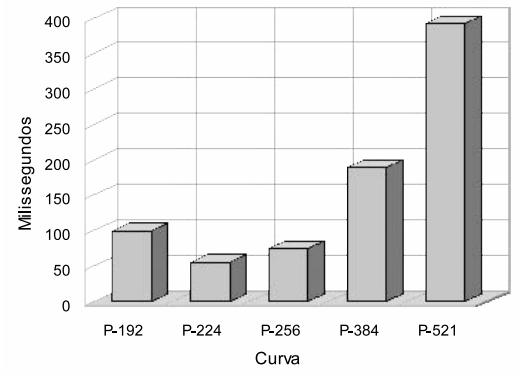

Figura 1: Tempo de execução para o algoritmo binário (direita para esquerda).

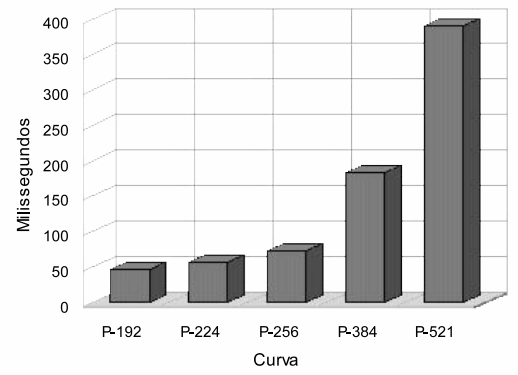

Figura 2: Tempo de execução para o algoritmo binário (esquerda para direita).

De maneira geral, o algoritmo binário que processa os *bits* mais significativos primeiro tem um desempenho melhor que o seu dual. Nos testes apresentados acima, o algoritmo binário versão direita para esquerda só é mais eficiente para a curva P-224. No entanto, é drasticamente ineficaz em relação à curva P-192 nas condições dos testes.

3.2 Algoritmo Baseado em NAF com Dimensão w

O valor de $\mathrm{NAF}_w(K)$ (Non-Adjacent Form) para $w \geq 2$ é o uma representação $K = \sum_{i=0}^{j} k_i 2^i = \langle k_j k_{j-1} \dots k_0 \rangle$ onde cada $|k_i| < 2^{w-1}$.

Exemplo 3.1 *Considere* $K = 52419574521$.
$NAF_2(K) = \langle 10\bar{1}000101\bar{0}1010010\bar{0}1\bar{0}1\bar{0}1000\bar{1}0\bar{1}0000\bar{1}001\rangle$
$NAF_3(K) = \langle 3000100\bar{3}00100003\bar{0}00\bar{3}00\bar{1}00300001\bar{0}01\rangle$
$NAF_4(K) = \langle 3000000700\bar{7}0000\bar{3}000\bar{3}000005\bar{0}0000007\bar{7}\rangle$
Onde $\bar{1} = -1, \bar{3} = -3, \bar{5} = -5, \dots$

Abaixo algumas propriedades da expansão $\mathrm{NAF}_w(K)$

- Cada inteiro K possui única representação $\mathrm{NAF}_w(K)$.
- O comprimento da representação NAF de K é no máximo o tamanho da representação binária mais 1.
- A densidade de coeficientes não nulos na representação $\mathrm{NAF}_w(K)$ é $j/(w+1)$, onde $j = \log_2 K$.
- No máximo um de w coeficientes de $\mathrm{NAF}_w(K)$ (seguidos) é diferente de zero.

Algoritmo 2: Cálculo de $\mathrm{NAF}_w(K)$

Entrada: Um inteiro K
Saída: $\mathrm{NAF}_w(K)$

```
1  início
2      i ← 0;
3      enquanto K ≥ 1 faça
4          se K é ímpar então
5              k_i ← K (mods 2^w);
6              K ← K − k_i;
7          senão
8              k_i ← 0;
9          K ← K ≫ 1;
10         i ← i + 1;
11     retorna ⟨k_j, k_{j−1}, ..., k_0⟩;
12 fim
```

O cálculo de $\mathrm{NAF}_w(K)$ pode ser computado via o algoritmo 2.

Na interação, se K é ímpar então o resto $r \equiv K \pmod{2^w}$ é calculado de forma que $(K-r)/2$ seja escolhido de forma que os próximos $(w-1)$ coeficientes sejam iguais a 0. Neste caso, o operador mods retorna o único resto pertencente ao intervalo $[-2^{w-1}, 2^{w-1}-1]$. Este conceito também é conhecido como *resto principal*. Olhando o algoritmo 1 vemos que ele só soma P e Q quando o coeficiente é 1. Se tivermos uma representação de K com uma densidade maior de coeficientes nulos, podemos acelerar o algoritmo binário (1). O algoritmo 3 utiliza a representação não-adjacente para multiplicar um ponto em uma curva elíptica por um inteiro, este possui tempo esperado de aproximadamente:

$$((2^{w-2}-1) + j/(w+1))\alpha + j\beta,$$

onde $j = \log_2 K$, α é o tempo de execução para uma adição e β é o tempo esperado para uma duplicação de um ponto.

Algoritmo 3: Multiplicação por escalar usando NAF com dimensão w.

Entrada: Um inteiro K e um ponto $P \in \Omega$
Saída: $KP \in \Omega$

```
1  início
2      Calcule a representação NAF de K, Σ_{i=0}^{j} k_i 2^i = NAF_w(K) (algoritmo 2);
3      Pré-Computação: Calcule P_i = iP para i ∈ {1, 3, 5, 7, ..., 2^{w−1} − 1};
4      Q ← ∞;
5      para i = j até 0 faça
6          Q ← 2Q;
7          se k_i ≠ 0 então
8              se k_i > 0 então
9                  Q ← Q + P_{k_i};
10             senão
11                 Q ← Q − P_{−k_i};
12     retorna Q;
13 fim
```

A tabela 1 mostram os resultados em milissegundos da multiplicação por escalar (incluindo a pré-computação quando houver). A tabela 2 mostra o custo, em milissegundos, da pré-

computação referente a linha 3 no algoritmo 3.

Curva	NAF_2	NAF_3	NAF_4	NAF_5	NAF_6	NAF_7	NAF_8	NAF_9	NAF_{10}
P-192	38	30	33	33	44	60	97	168	352
P-224	43	43	41	43	51	71	118	203	423
P-256	58	54	55	57	66	89	144	246	510
P-384	144	138	138	141	156	194	287	461	910
P-521	300	312	288	292	317	376	524	796	1501

Tabela 1: Desempenho da multiplicação por escalar.

Curva	Dimensão w do NAF para pré-computação							
	NAF_3	NAF_4	NAF_5	NAF_6	NAF_7	NAF_8	NAF_9	NAF_{10}
P-192	18	19	18	18	43	71	159	330
P-224	1	1	4	11	29	73	168	388
P-256	0	2	4	15	35	85	200	465
P-384	2	2	7	23	58	143	339	788
P-521	1	4	13	35	91	245	533	1238

Tabela 2: Tempo de execução para pré-computação.

3.3 Desempenho e Comparações

De forma geral, o método baseado em NAF tem se mostrado mais eficiente do que o método binário. Este fato se deve a representação com maior densidade de coeficientes nulos que o NAF proporciona. Comparamos a nossa implementação com a biblioteca `Java` código aberto *Bouncy Castle Crypto Package* versão 1.41. Este é um dos pacotes criptográficos em `Java` mais conhecidos. Neste caso, para a comparação, usamos o método `multiply` que é responsável pela multiplicação por escalar da classe `ECPoint` (Bouncy Castle). Utilizamos os mesmos parâmetros recomendados pelo NIST em ambas as implementações.

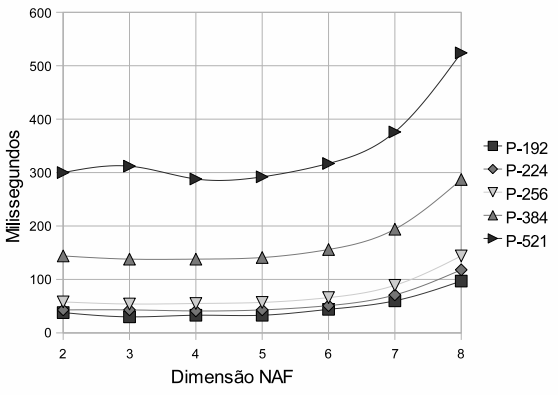

Figura 3: Tempo de execução usando NAF agregado a pré-computação.

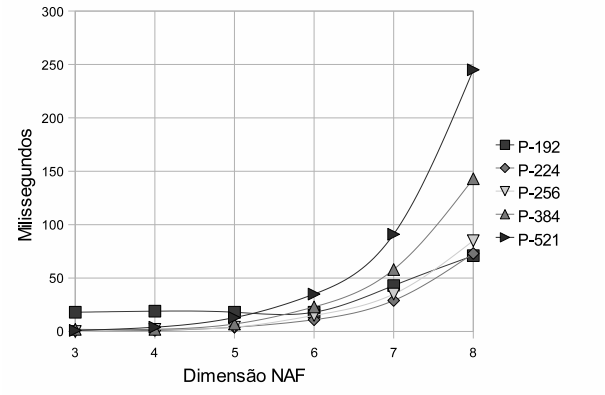

Figura 4: Tempo de execução para a pré-computação referente ao algoritmo 3.

A figura 5 compara os tempos de execução das implementações para as curvas e parâmetros estabelecidas pelo NIST. Já o gráfico 6 compara os resultados retirando o tempo de pré-computação da nossa implementação.

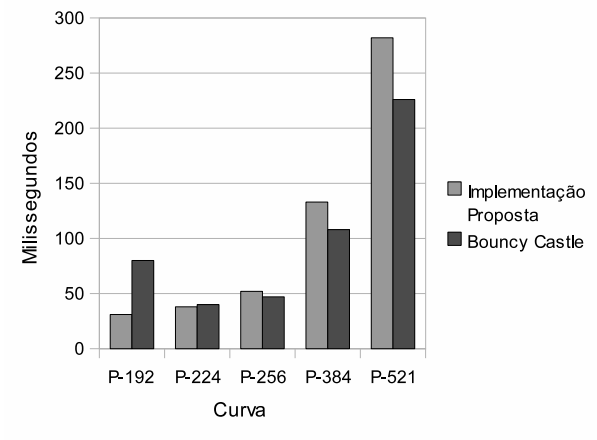

Figura 5: Comparação entre a implementação proposta e a biblioteca Bouncy Castle.

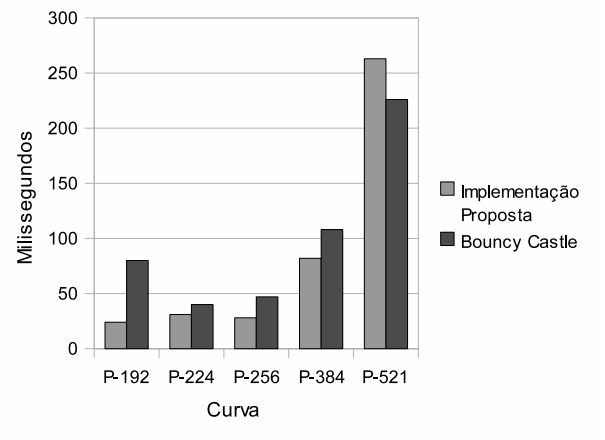

Figura 6: Comparação sem contar o tempo de pré-computação da implementação proposta.

Para esta comparação, coletamos os melhores resultados apresentados pela implementação mostrada neste trabalho. Sendo assim, o tamanho da dimensão do NAF que apresentaram melhores resultados é $3 \leq w \leq 5$, como pode ser observado na tabela 1. Sem a pré-computação, a dimensão do NAF que mostrou melhores desempenhos é $w = 9$ (veja as tabelas 1 e 2). A nossa implementação se mostrou mais eficiente para as curvas cujo corpo $\mathbb{Z}_p$ são menores, neste caso, P-192 e P-224. Já para as curvas P-224, P-384 e P-521 nossa implementação apresentou-se mais lenta que a biblioteca *Bouncy Castle* como foi ilustrado no gráfico 5. Se retirarmos a pré-computação, temos que nossa implementação só não fica mais rápida na curva P-521 (veja figura 6).

4 Conclusões e Trabalhos Futuros

Este artigo descreveu dois métodos gerais de multiplicação por escalar em curvas elípticas. Mostramos os desempenhos em *software* dos respectivos algoritmos citados. E, finalmente, comparamos nossa implementação com a implementação da multiplicação por escalar da biblioteca criptográfica *Bouncy Castle Crypto Package*. O próximo objetivo é melhorar o desempenho da implementação e migrar para dispositivos móveis que dão suporte a plataforma J2ME (*Java 2 Micro Edition*).

Referências

[1] *Digital signature standard (DSS)*, National Institute of Standards and Technology, Washington, 2000, URL: `http://csrc.nist.gov/publications/fips/`. Note: Federal Information Processing Standard 186-2.

[2] *Java TM 2 Platform, Standard Edition, v 1.4.2 API Specification*, 2008, Available at `http://java.sun.com/j2se/1.4.2/docs/api/`.

[3] Whitfield Diffie and Martin E. Hellman, *New directions in cryptography*, IEEE Trans. Information Theory **IT-22** (1976), no. 6, 644–654. MR MR0437208 (55 #10141)

[4] Darrel Hankerson, Alfred Menezes, and Scott Vanstone, *Guide to elliptic curve cryptography*, Springer Professional Computing, Springer-Verlag, New York, 2004. MR MR2054891 (2005c:94049)

[5] Neal Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation **48** (1987), no. 177, 203–209.

[6] Neal Koblitz, Alfred Menezes, and Scott Vanstone, *The state of elliptic curve cryptography*, Des. Codes Cryptography **19** (2000), no. 2-3, 173–193.

[7] Victor S. Miller, *Use of elliptic curves in cryptography*, Advances in cryptology—CRYPTO '85 (Santa Barbara, Calif., 1985), Lecture Notes in Comput. Sci., vol. 218, Springer, Berlin, 1986, pp. 417–426. MR MR851432 (88b:68040)