ISSN 1984-820X
Anais do CNMAC
v.3
PUBLICAÇÃO DA SBMAC
CNMAC2010
XXXIII CONGRESSO NACIONAL DE MATEMÁTICA APLICADA E COMPUTACIONAL

Paralelização para o Algoritmo de Exponenciação Modular

Pedro Lara, **Fábio Borges**, **Renato Portugal**
Laboratório Nacional de Computação Científica,
25651-075, Petrópolis, RJ
E-mail: pcslara@lncc.br, borges@lncc.br, portugal@lncc.br

Resumo: *Algoritmos para a exponenciação modular tem sido utilizados de maneira central em diversos criptossistemas assimétricos. Sendo, em muitos casos, o algoritmo com o maior custo computacional, por exemplo, o criptossistema RSA. Este trabalho descreve métodos paralelos para a computação eficiente da exponenciação modular. Serão avaliados resultados teóricos e práticos acerca das técnicas propostas.*

Palavras-chave: *Exponenciação Modular, Paralelização, Criptografia*

1 Introdução

O protocolo pioneiro proposto por Whitfield Diffie e Martin E. Hellman [4] em 1975 possibititou estabelecer uma chave secreta entre dois usuários em um canal de comunicação inseguro, que é considerada a primeira prática de criptografia de chave pública. Seja p um primo e g um gerador do grupo cíclico $\mathbb{Z}_p^*$. Vamos supor que Alice e Bob queiram combinar uma chave secreta. Inicialmente, Alice escolhe aleatoriamente um inteiro secreto a tal que $a \in \mathbb{Z}_p$ e envia para Bob $k_a = g^a$. Analogamente, Bob seleciona um inteiro secreto $b \in \mathbb{Z}_p$ e envia à Alice $k_b = g^b$. Agora ambos usam seu inteiro secreto para obter uma chave secreta compartilhada $k_{ab} = (k_a)^b = (k_b)^a = g^{ab}$. Este protocolo usa exaustivamente o algoritmo de exponenciação modular. O RSA (Ron **R**ivest, Adi **S**hamir e Len **A**dleman) foi publicado em 1978 [10] e é atualmente o principal criptossistema de chave pública usado em aplicações comerciais. A segurança deste método está baseada na ausência de um algoritmo eficiente para o Problema da Fatoração de Inteiros (PFI). O criptossistema RSA consiste em três partes - geração de chaves, encriptação e decriptação. As duas últimas utilizam diretamente a exponenciação modular. Logo, o desempenho deste criptossistema assimétrico está estreitamente ligado com o desempenho da exponenciação modular [8]. A implementação em *hardware* do RSA é discutida em [1].

Brickell, Gordon, McCurley and Wilson (BGMW) em [2] usaram a pré-computação de algumas potências para acelerar a exponenciação. Em [3] os mesmos autores propuseram duas paralelizações usando pré-computação. Outras paralelizações relacionadas ao BGMW são discutidas em [7]. Deste modo, estas técnicas são particularmente interessantes em métodos que utilizam base fixa, por exemplo, protocolo Diffie-Hellman. O uso do RSA com múltiplos primos [11] (*multi-prima RSA*) juntamente com o Teorema do Resto Chinês fornece uma paralelização direta e relativamente eficiente. N. Nedjah e L. M. Mourelle em [9] discutem e comparam a paralelização de alguns dos principais algoritmos de exponenciação modular. De maneira geral, os métodos de paralelização propostos anteriormente atuam apenas para problemas específicos, por exemplo, quando se conhece *a priori* a base g. Diferentemente, este trabalho descreve importantes melhorias referente a técnica geral de exponenciação modular proposta em [6]. O método paralelo que será apresentado se mostrou mais escalável e eficiente computacionalmente do que em [6].

2 Paralelização do Algoritmo de Exponenciação Modular

Nesta seção será feita uma revisão das técnicas apresentadas em [6]. Quando levamos em conta a representação em base binária do expoente $e = (b_j b_{j-1} \dots b_1 b_0)_2$, podemos identificar a seguinte identidade

$$g^e = g^{2^{r+1}e_2} \cdot g^{e_1}, \tag{1}$$

onde $e_1 = (b_r \dots b_1 b_0)_2$, $e_2 = (b_j \dots b_{r+1})_2$ e $r = \lceil (j-1)/2 \rceil$. Na verdade, a equação (1) segue do fato que

$$e = 2^{r+1}e_2 + e_1. \tag{2}$$

Isto posto, podemos computar as parcelas $g^{2^{r+1}e_2}$ e g^{e_1} de (1) separadamente, sem dependência mútua. A equação (1) resume a ideia central da paralelização do algoritmo binário de exponenciação. O algoritmo 1 sistematiza estas ideias [6]. Quando $j+1$ (número de *bits* de e)

Algoritmo 1: Paralelização do algoritmo de exponenciação modular.

Entrada: Inteiro $g \in \mathbb{Z}_p$ e $e = \sum_{i=0}^{j} 2^i b_i$ onde $b_i \in \{0,1\}$ (representação em base binária).
Saída: $g^e \mod p$.

```
1  início
2  | e1 ← (br ... b1b0)2;
3  | e2 ← (bj ... br+1)2;
4  | início
5  | | [Região Paralela 1]
6  | | a1 ← g^e1 mod p;
7  | fim
8  | início
9  | | [Região Paralela 2]
10 | | a2 ← g^(2^(r+1));
11 | | a2 ← a2^e2 mod p;
12 | fim
13 | retorna a1 · a2 mod p;
14 fim
```

for par, a “separação” do expoente e será igualmente dividida. Cada um dos expoentes e_1 e e_2 fica com $\frac{j+1}{2}$ *bits* em sua representação. Senão, e_1 deverá ter um *bit* a mais em relação a e_2. Como a computação na Região Paralela 2 (veja algoritmo 1) é mais custosa que na Região Paralela 1, é mais sensato, quando $j+1$ for impar, que e_1 tenha um *bit* a mais que e_2. Esta escolha pode permitir um equilíbrio maior entre o tempo de execução das regiões paralelas, o que é bastante desejado. O algoritmo 1 atua com duas linhas paralelas de execução, no entanto, essa ideia pode ser mais geral - poderíamos ter $e_1, e_2, \dots, e_n$ onde n é o número linhas paralelas de execução e proceder de forma análoga ao que foi mostrado no algoritmo 1 (veja o algoritmo 2). No algoritmo 2, o tamanho do expoente e_i para $i = 1, \dots, n$ também deve ser levado em conta. Nesse caso, o resto da divisão do número de *bits* de e por n deve ser analisado.

3 Análise de Complexidade

No algoritmo 1, é fácil perceber que a Região Paralela 2 exige uma computação maior que a Região Paralela 1. Neste caso, quando computamos $a_2 = g^{2^{r+1}}$ na verdade estamos calculando $r+1$ elevações ao quadrado e 1 multiplicação [6]. E, finalmente, quando fazemos $a_2 = a_2^{e_2}$ fazemos $r+1$ elevações ao quadrado e $\frac{r+1}{2}$ multiplicações em média. Assim, a Região Paralela 2 do algoritmo 1 consome, aproximadamente, $2(r+1)$ elevações ao quadrado e $\frac{r+1}{2}$ multiplicações. Se somarmos a multiplicação computada na linha 13 do algoritmo 1, temos um custo computacional de

$$2(r+1)E + \left(\frac{r+1}{2} + 1\right)M, \tag{3}$$

Algoritmo 2: Paralelização com n linhas de execução.

Entrada: Inteiro $g \in \mathbb{Z}_p$ e $e = \sum_{i=0}^{j} 2^i b_i$ onde $b_i \in \{0,1\}$ (representação em base binária).
Saída: $g^e \mod p$.

```
1  início
2  |  e_1 ← (b_{r_1} ... b_1 b_0)_2;
3  |  e_2 ← (b_{r_2} ... b_{r_1+1})_2;
4  |  ⋮
5  |  e_n ← (b_j ... b_{r_{n-1}+1})_2;
6  |  início
7  |  |  [Região Paralela 1]
8  |  |  a_1 ← g^{e_1} mod p;
9  |  fim
10 |  início
11 |  |  [Região Paralela 2]
12 |  |  a_2 ← g^{2^{r_1}};
13 |  |  a_2 ← a_2^{e_2} mod p;
14 |  fim
15 |  ⋮
16 |  início
17 |  |  [Região Paralela n]
18 |  |  a_n ← g^{2^{r_{n-1}}};
19 |  |  a_n ← a_n^{e_n} mod p;
20 |  fim
21 |  retorna a_1 · a_2 · ... · a_n mod p;
22 fim
```

onde M é o tempo requerido para uma multiplicação e E é o tempo computacional para uma elevação ao quadrado. Essa análise vale quando $j+1$ é par, que é o pior caso. Já em função de j, a expressão (3) fica

$$(j+1)E + \left(\frac{j+1}{4} + 1\right) M. \tag{4}$$

Observe, que não incluímos a Região Paralela 1 no tempo computacional. Isso segue do fato que esta região possui um custo computacional médio de $(r+1)E + \frac{r+1}{2}M$, ou seja, faz $(r+1)E$ unidades de tempo a menos que a Região Paralela 2. Se compararmos com a complexidade do algoritmo binário sequencial, que é de $(j+1)E + \left(\frac{j+1}{2}\right)M$, com a complexidade do algoritmo paralelo (veja expressão (4)), temos uma diferença de aproximadamente $\frac{j+1}{4}M$ unidades de tempo.

Não é difícil generalizar estas ideias para n linhas paralelas. Para isso basta tomar $r = \frac{j-1}{n}$ e lembrar que, ao final do algoritmo (linha 21 do algoritmo 2), $n-1$ multiplicações serão executadas. Na verdade, o número de elevações ao quadrado continua sendo $(j+1)$, no entanto, o número de multiplicações fica $\frac{j+1}{2n} + n - 1$. Generalizando, temos um custo computacional de:

$$(j+1)E + \left(\frac{j+1}{2n} + n - 1\right) M. \tag{5}$$

Assim, em todos os casos sempre fazemos $(j+1)E$. A diferença fica com o número de multiplicações. Desta forma, para fazer uma análise um pouco mais detalhada precisamos explorar a função $\eta(j,n)$ que retorna no número de multiplicações definida por

$$\eta(j,n) = \frac{j+1}{2n} + n - 1. \tag{6}$$

A fim de minimizar o número de multiplicações e fazendo j constante, considere a derivada parcial

$$\frac{\partial \eta(j,n)}{\partial n} = -\frac{j+1}{2n^2} + 1. \tag{7}$$

Igualando (7) a zero e resolvendo a equação na variável n, temos a seguinte relação entre o número de linhas paralelas e o número de *bits* do expoente

$$n = \sqrt{\frac{j+1}{2}}. \tag{8}$$

Como a concavidade de $\eta(j,n)$ é positiva, segue que a equação (8) fornece um mínimo global. Este é o número ótimo de linhas paralelas n em função do número de *bits* $j+1$, que determina a segurança do criptossistema.

4 Paralelização Massiva

O algoritmo 2 não usa totalmente os recursos paralelos. A grande ideia do próximo algoritmo é paralelizar a linha 21 do algoritmo 2 de forma que cada processador multiplique em paralelo o resultado parcial. Assim, na primeira iteração, usamos $\frac{n}{2}$ processadores sendo que cada processador $i = \{1, \ldots, \frac{n}{2}\}$ computa $a_i = a_{2i-1} \cdot a_{2_i}$. Na próxima iteração será necessário $\frac{n}{4}$ processadores, desta forma precisamos de $\lceil \log_2 n \rceil$ iterações. O algoritmo seguinte descreve a execução paralela para a linha 21 do algoritmo 2.

Algoritmo 3: Paralelização da linha 21 do algoritmo 2.

Entrada: Inteiros $a_1, a_2, \ldots, a_n \in \mathbb{Z}_p$
Saída: O produtório $\prod_{i=1}^{n} a_i \in \mod p$.

```
1 início
2   enquanto n ≠ 1 faça
3     para cada processador i = 1 até n/2 faça em paralelo
4       A_i ← a_{2i-1} · a_{2i} mod p;
5     n ← n/2;
6     para i = 1 até n faça
7       a_i ← A_i;
8   retorna A_1;
9 fim
```

4.1 Análise de Complexidade

A diferença do algoritmo 2 e da modificação apresentada no algoritmo 3 está no número de multiplicações modulares. Neste caso, a linha 21 do algoritmo 2 executa $n-1$ multiplicações modulares. Com a modificação apresentada nesta seção a linha 21 passa a computar $\lceil \log_2 n \rceil$ multiplicações, assim o custo computacional para este algoritmo fica

$$(j+1)E + \left(\frac{j+1}{2n} + \lceil \log_2 n \rceil\right) M. \tag{9}$$

Neste caso, o número de multiplicações modulares é

$$\mu(j,n) = \frac{j+1}{2n} + \lceil \log_2 n \rceil. \tag{10}$$

Desta forma, resolvendo $\frac{\partial \mu(j,n)}{\partial n} = 0$ temos

$$n = \frac{\ln 2}{2}(j+1). \tag{11}$$

Na expressão (5), que nos dava o custo computacional para o algoritmo 2, temos um total de multiplicações iguais a $\eta(j,n) = \frac{j+1}{2n} + n - 1$. Se compararmos com a equação (10), reduzimos um fator linear $(n-1)$ a um fator logarítmico ($\lceil \log_2 n \rceil$) no número de multiplicações modular.

ISSN 1984-820X
Anais do CNMAC
v.3
PUBLICAÇÃO DA SBMAC
CNMAC2010
XXXIII CONGRESSO NACIONAL
DE MATEMÁTICA APLICADA
E COMPUTACIONAL

5 Resultados

A implementação foi desenvolvida em linguagem `C` usando como ferramenta básica de paralelização a biblioteca OpenMPI que implementa o padrão MPI (Message Passing Interface) [12]. A API (Application Programming Interface) do MPI oferece, de maneira eficaz, a paralelização usando memória física distribuída. Para aritmética de precisão múltipla utilizamos a biblioteca GMP (GNU Multiple Precision) [5]. As escolhas acima visaram um bom desempenho de tempo de execução. O *hardware* utilizado foram 8 nós `Sun Blade x6250` cada nó com 2 processadores `Intel Xeon E5440 Quad Core 3GHz` e 16GB de memória física interligados por um barramento *InfiniBand*. Para todos os testes de desempenho utilizamos um expoente cuja a esparsidade da representação binária seja $\frac{1}{2}$. Neste caso, o expoente para os testes ficaram da forma $(101010\ldots1010)_2$. Para os testes, usamos até 64 processadores. As figuras de 1 a 3 exibem as medianas do tempo de execução e os *speedups* de 40 coletas sucessivas para até 64 processadores para 512, 1024 e 2048 bits no expoente referentes ao algoritmo 2. A escala do tempo de execução está em microssegundos.

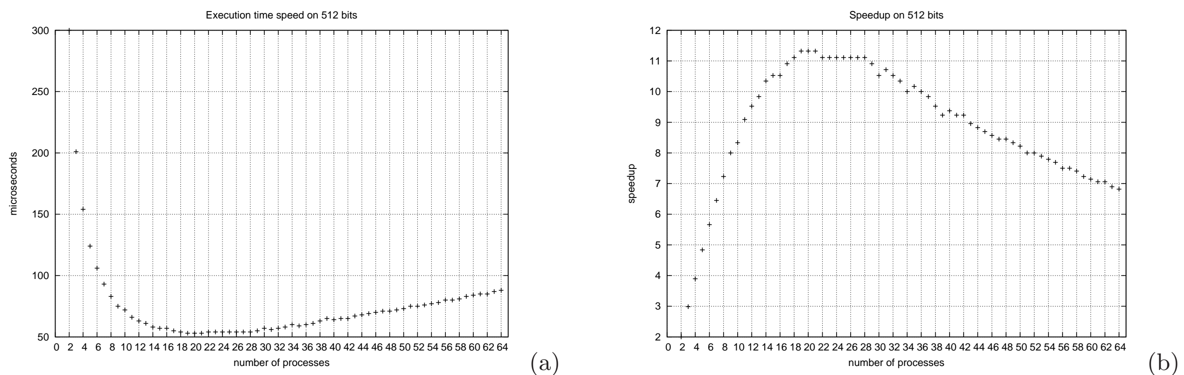

Figura 1: Tempo de execução (a) e *speedup* (b) para até 64 processadores usando 512 bits

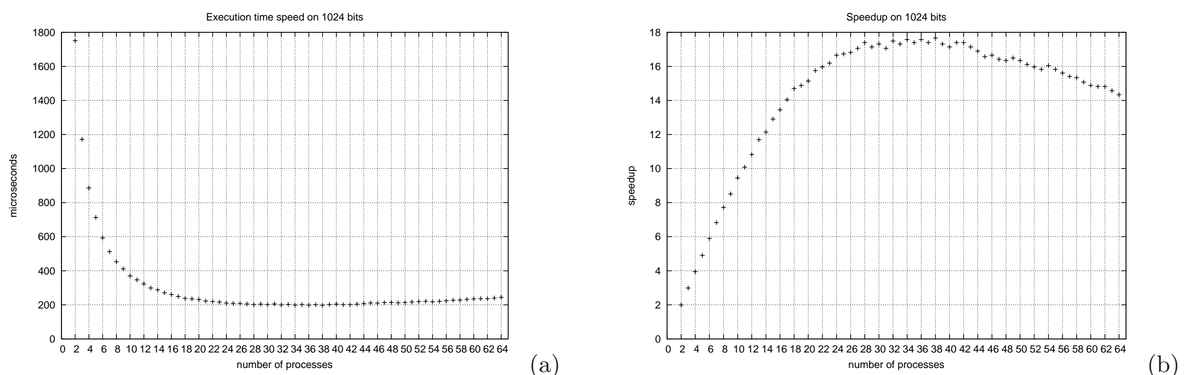

Figura 2: Tempo de execução (a) e *speedup* (b) para até 64 processadores usando 1024 bits

Para expoentes de 512 bits, por exemplo, o *speedup* começa a se degenerar a partir de 20 processadores. Considerando que da análise teórica temos um número ótimo de processadores igual $n = \sqrt{\frac{j+1}{2}}$, onde $(j+1)$ representa o número de bits, neste caso, para 512 bits, temos $n = \sqrt{\frac{512}{2}} = 16$, que é relativamente próximo do obtido (ver figura 1).

A figura 4 compara os algoritmos 2 e 3 para entradas de 1024 bits. Neste testes pegamos os melhores valores obtidos para até 64 processadores. Para até 20 processadores quase não existe diferença entre os algoritmos. O *overhead* gerado pela comunicação entre os processos no algoritmo 3 faz com que ele não seja mais rápido que o algoritmo 2 para até 20 processadores. No gráfico da figura 4 é possível observar que a implementação do algoritmo 3 se mantém escalável

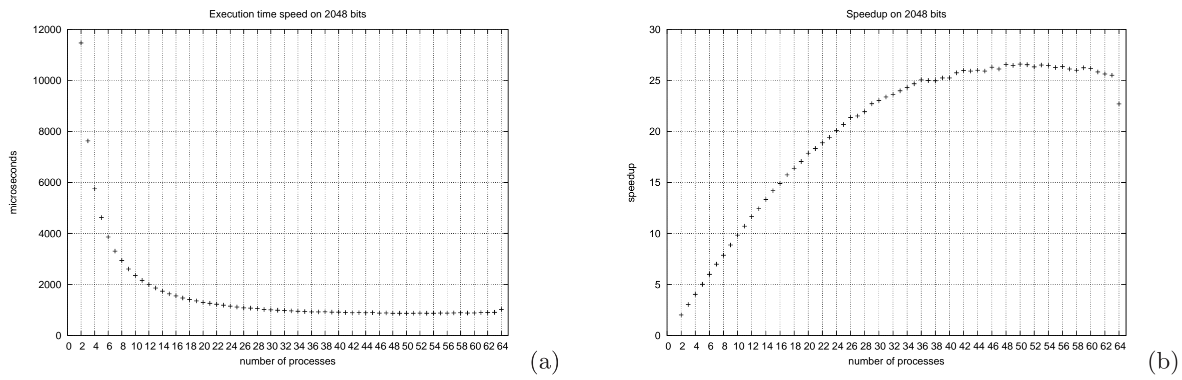

Figura 3: Tempo de execução (a) e *speedup* (b) para até 64 processadores usando 2048 bits

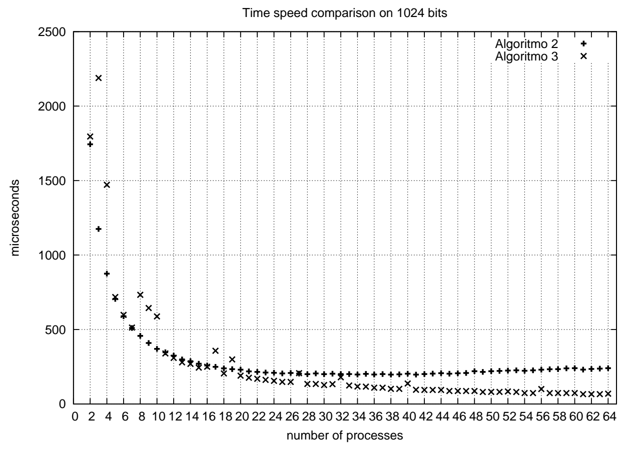

Figura 4: Comparativo entre os algoritmo de paralelização apresentados.

e com o tempo de execução inferior ao gráfico do algoritmo 2 a partir de 20 processadores. O gráfico da figura 4 permite distinguir claramente aspectos importantes dos dois métodos: o algoritmo 2 atua bem com poucos processadores e necessita de pouca comunicação entre os processos, a modificação usando o algoritmo 3, por sua vez, exige muita comunicação entre os processos e é substancialmente mais escalável que o algoritmo 2. No gráfico da figura 4, o algoritmo 3 não apresenta uma concavidade, neste caso o mínimo e máximo estão localizados nos extremos. Da análise teórica, que resulta na equação (8), temos um mínimo de aproximadamente 23 processadores, ficando coerente com os resultados experimentais do algoritmo 2 apresentado na figura 4.

6 Conclusões

Este trabalho descreveu propostas para a paralelização do algoritmo da exponenciação modular, usado na cifragem e decifragem do método de criptografia RSA. Inicialmente foi apresentado um algoritmo paralelo baseado na partição do expoente. Posteriormente foi observado que o algoritmo apresentado não usava todos os recursos paralelos disponíveis. Desta forma, foi apresentadas modificações no algoritmo 2 para conceber um algoritmo mais eficiente. Foi obtido uma redução significativa do número de multiplicações modulares no desempenho final do algoritmo de exponenciação. Anteriormente tínhamos um fator linear $(n-1)$ e com a modificação temos um fator logarítmico $(\log_2(n))$ no número de multiplicações necessárias, onde n é o número de processadores. Assim a nova paralelização se mantém escalável para um número substancialmente maior de processadores. Este fato motiva, como trabalhos futuros, explorar o poder das unidades gráficas de processamento, GPU, para acelerar o processo de exponenciação modular.

Em aplicações reais, isto pode ser bastante interessante para servidores com altas cargas de requisições de segurança, por exemplo, *https servers*. Também, é de grande interesse, estudar a implementação de algoritmos para a multiplicação por escalar em curvas elípticas, visto que técnicas criptográficas baseadas em curvas elípticas apresentam, de maneira geral, uma chave substancialmente menor que os algoritmos baseados no problema do logaritmo discreto em $\mathbb{Z}_p$. É de grande relevância ressaltar que a classe de algoritmos para a multiplicação por escalar em curvas elípticas possui relação bijetiva com os algoritmos de exponenciação modular.

7 Agradecimentos

Gostaríamos de agradecer o suporte financeiro oferecido pelo CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico).

Referências

[1] Ernest F. Brickell, *A survey of hardware implementation of RSA*, CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology (London, UK), Springer-Verlag, 1990, pp. 368–370.

[2] Ernest F. Brickell, Daniel M. Gordon, Kevin S. Mccurley, and David B. Wilson, *Fast exponentiation with precomputation*, Advances in Cryptology – Proceedings of CRYPTO'92, vol. 658, Springer-Verlag, 1992, pp. 200–207.

[3] ______, *Fast exponentiation with precomputation: Algorithms and lower bounds*, 1995, Preprint http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.606.

[4] 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)

[5] Torbjörn Granlund, *GNU multiple precision arithmetic library 4.1.2*, December 2002, http://swox.com/gmp/.

[6] Pedro C. S. Lara, Fábio Borges, and Renato Portugal, *Paralelização eficiente para o algoritmo binário de exponenciação modular*, Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (Campinas - SP), 2009.

[7] Chae Hoon Lim and Pil Joong Lee, *More flexible exponentiation with precomputation*, Advances in Cryptology – CRYPTO'94, 1994, pp. 95–107.

[8] Nadia Nedjah and Luiza Macedo Mourelle, *Efficient parallel modular exponentiation algorithm*, ADVIS '02: Proceedings of the Second International Conference on Advances in Information Systems (London, UK), Springer-Verlag, 2002, pp. 405–414.

[9] ______, *Parallel computation of modular exponentiation for fast cryptography*, International Journal of High Performance Systems Architecture **1** (2007), no. 1, 44–49.

[10] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Commun. ACM **21** (1978), no. 2, 120–126.

[11] RSA Labs, *PKCS#1 v2.0 Amendment 1: Multi-Prime RSA*, 2000.

[12] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra, *Mpi-the complete reference, volume 1: The mpi core*, MIT Press, Cambridge, MA, USA, 1998.