Algoritmos de Multiplicação de Montgomery em Hardware e Impacto no Algoritmo RSA

[1]**Marçal Luiz Bissoli,** [1]**Kalinka R.L.J. Castelo Branco,** [2]**Edward David Moreno**

[1]Univem – Centro Universitário Eurípides de Marília, Marilia, S.P
[2]*UEA (Universidade do Estado do Amazonas) & Empresa BenQ, Manaus, AM, Brasil*

marcalbissoli@gmail.com, kalinka@univem.edu.br, edwdavid@gmail.com

Resumo: *Neste artigo descreve-se o método de Montgomery para multiplicação modular, bem como são examinados dois dos seus algoritmos, a saber: O Algoritmo Rápido de Montgomery (**Fast Montgomery Algorithm)** e o Algoritmo Mais Rápido de Montgomery (**Faster Montgomery algorithm).** São ainda apresentadas dois diagramas de máquinas de Estados correspondentes à descrição em VHDL desses algoritmos e a respectiva implementação em FPGAs. No final são apresentadas comparações entre os desempenhos dos algoritmos Rápido e Mais Rápido de Montgomery, focando seu impacto no algoritmo RSA para diferentes tamanhos de chaves.*

1. Introdução

O algoritmo criptográfico RSA é baseado no cálculo de $\mathbf{C = M^e \bmod n}$ para cifrar uma mensagem e $\mathbf{T = C^e \bmod n}$ para decifrá-la, onde T é o texto claro e C, o texto criptografado. Deve ser observado que alguns cálculos básicos, conhecidos como exponenciação modular, são usados tanto para a cifragem quanto para a decifragem [Rivest, 1998].

A segurança do sistema advém do método do quais os parâmetros para exponenciação modular são derivados. O expoente (**e**) deve ser um número primo. O módulo (**n**) deve ser o produto de dois números primos usualmente chamados de **p** e **q**. O expoente privado (**d**) é derivado de **p**, **q** e **e**.

Deduzir os fatores primos de um número muito grande é muito difícil e se torna muito mais difícil à medida que se usam números cada vez maiores. Assim, se **p** e **q** são suficientemente grandes, é impossível fatorar **n** e obter a chave privada em um período razoável de tempo, mesmo com os computadores mais rápidos.

Atualmente, **n** é um número escolhido entre números de 512 a 2048 bits [Moreno, 2005], [Bayley, 1999]. Números com 2048 bits poderiam utilizar durante anos centenas de milhares de computadores para deduzir os números **p** e **q** das chaves públicas [Mctaggart, 2006], [Eberle, 2004].

Como se vê, a matemática sob o algoritmo RSA é resumida em duas operações, multiplicação modular e exponenciação modular. Logo, para se criar uma implementação eficiente do RSA deve-se projetar eficientemente a multiplicação modular de dois números. Entretanto, a multiplicação modular tem um grande inconveniente, a divisão deve ser empregada para que se obtenha o valor do resto. A divisão é uma operação complexa que consome recursos e tempo do computador [Bissoli, 2007].

Dessa forma, embora o RSA ofereça excelente segurança criptográfica, por causa da complexidade dos cálculos matemáticos envolvidos no algoritmo, ele cifra e decifra mensagens de forma muito lenta, quando comparado aos algoritmos de chave privada. Dessa forma, a busca por métodos que aumentem a velocidade de cifragem/decifragem quando se utiliza o criptosistema RSA é muito grande.

O método recorrente de se obter uma potência modular ($x^n \bmod m = x^{n-1} \bmod m . x \bmod m$) é impraticável para aplicações mais modernas. Métodos alternativos têm sido propostos de forma a obter o resto mais rapidamente nas operações de divisão [Dally, 2003/2002], [Khaldoon, 2002], [Gutub, 2000], [Todorov, 2000], [Prince, 2002].

A solução mais popular e útil tem sido o emprego do algoritmo de Montgomery para multiplicação modular [Blum, 1999], [Shantz, 2000], [Koc, 1996]. Esse algoritmo reduz a complexidade da multiplicação modular, especialmente se os números envolvidos são muito grandes como é o caso da criptografia com chaves públicas RSA.

Assim, neste artigo apresenta-se uma forma de realizar a multiplicação e exponenciação modular de forma rápida, usando três diferentes versões do algoritmo de Montgomery. Esses algoritmos são explicados, e foram descritos em VHDL e mapeados em FPGAs. Neste trabalho foi possível realizar implementações do RSA para chaves de 16, 32 e 64 bits. O interessante é discutir os resultados e apresentar os possíveis pontos críticos dessas implementações.

O artigo está organizado em 5 seções. A seção 2 apresenta uma breve descrição da multiplicação de Montgomery, enquanto a seção 3 apresenta três versões diferentes do algoritmo, discutindo pontos de melhorias e otimização, e na seção 4 se enfatiza em aspectos de implementação e desempenho em FPGAs. Finalmente, a seção 5 apresenta as conclusões e sugestões para trabalhos futuros.

2. A Multiplicação de Montgomery

A multiplicação modular de *Montgomery* baseia-se em operações realizadas no domínio de Montgomery (ver figura1). Esta operação calcula:

$$MM(X,Y) = XYR^{-1} \bmod(m),$$

onde m é um inteiro no intervalo $2^{n-1} \leq m < 2^n$ tal que MDC(m,R) = 1,

que ocorre sempre que m for ímpar já que R = 2^n é obrigatoriamente par.

O algoritmo transforma um inteiro no intervalo [0, m-1] em outro inteiro do mesmo intervalo ao qual denominaremos m-resto do inteiro referido anteriormente. Então, diz-se que o m-resto está no domínio de Montgomery.

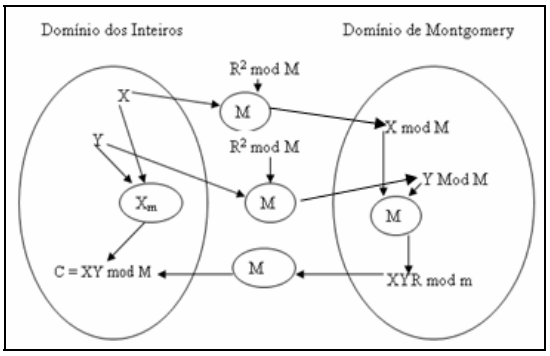

Figura 1 – Multiplicação de Montgomery

A seguir, mostra-se a multiplicação modular Z de dois inteiros X e Y utilizando o método de *Montgomery*. Define-se em seguida a multiplicação de *Montgomery* (MM, ver figura 1). Para tanto, são introduzidas as seguintes definições e notações:

- m é o módulo da multiplicação modular.
- X é o multiplicador da multiplicação modular.
- Y é o multiplicando da multiplicação modular.
- Z é o resultado da multiplicação modular.
- R é uma constante igual a 2^n.
- R^{-1} é o inverso multiplicativo de R mod m ($R.R^{-1} \bmod m = 1$)

As imagens de X e Y, X' e Y'(no domínio de Montgomery), respectivamente, são obtidas por:

$$X' = MM(X, R^2) = X.R^2 R^{-1} \bmod m = X.R \bmod m$$

$$Y' = MM(Y, R^2) = YR^2 . R^{-1} \bmod m = Y.R \bmod m.$$

A imagem Z' de Z é calculada efetuando-se a multiplicação de *Montgomery* nas imagens X' e Y':

- Z' = MM(X',Y')= MM(X.R) mod m, YR mod m) = X.Y.R mod m A multiplicação modular Z é obtida por MM(Z',1) assim,
- $Z = MM(Z',1) = X.Y.\ R.R^{-1} \bmod m = X.Y \bmod m = Z$

A grande vantagem da multiplicação de Montgomery em comparação com outras multiplicações modulares é que com simples operações matemáticas (adição e multiplicação) e pouco dispêndio de processamento, ela fornece rapidamente os resultados requeridos. A divisão e também a inversão são realizadas por meio de ações simples como um deslocamento.

3. Três Versões do Algoritmo de Montgomery

O método clássico para computar a multiplicação modular é efetuar a multiplicação e então subtrair o módulo diversas vezes até que se obtenha um resultado menor do que o módulo. Este método é ineficiente e tem baixa velocidade computacional [Chiaramonte, 2006], [Muzzi, 2005]. A idéia de Montgomery é reduzir os tamanhos dos resultados intermediários para uma quantidade fixa de n+1 bits. Isto é obtido intercalando-se as multiplicações e adições (passos 3 e 4 da Figura 2) com divisões por 2 (Passo 5 da Figura 2), reduzindo-se, dessa forma, o tamanho do comprimento em bits de cada resultado intermediário em um bit.

A figura 2 apresenta um primeiro algoritmo de multiplicação seguindo a indicação dada no domínio de Montgomery, para o qual se necessita da seguinte notação:

- Dados: X,Y,M sendo $0 \leq X,Y<M$
- Resultado: $P = X*Y*2^{-n} \bmod M$
- n: número de bits em X
- x_i : i-ésimo bit de X
- p_o : bit menos significativo de P

1. P:=0;
2. Para i = 0 até n-1 faça
3. $P := P + x_i * Y;$
4. $P := P + P_0 * M;$
5. $P := P/2;$
6. Se $P \geq M$ então $P := P-M;$

Figura 2. Multiplicação de Montgomery

A maior parte das soluções de multiplicação modular é baseada no algoritmo de Montgomery [Koc, 1996], [Savas, 2000], Shantz, 2001], Blum, 1999], [Dally, 2003].

A idéia básica de Montgomery é a seguinte: Somar um múltiplo de M no resultado intermediário não modifica o resultado final, porque o resultado é computado com módulo M, onde M é um número ímpar. Assim, em cada adição no laço (loop) interno, o bit menos

significativo do resultado intermediário é inspecionado. Se ele é 1, isto significa que o resultado intermediário é ímpar, então se adiciona M para torná-lo par. O número par pode ser dividido por 2 sem deixar resto. Esta divisão por 2 reduz novamente o resultado intermediário para n+1 bits. Após **n** passos obtêm-se na realidade uma divisão por 2^n.

A multiplicação de Montgomery exige dois passos no mesmo processo de multiplicação, consequentemente, dobrando o tempo de computação [Bissoli, 2007].

O primeiro passo computa $P = X*Y*2^{-n} \bmod M$ e o segundo computa $P*2^{2n}*2^{-n} \bmod M = X*Y \bmod M$, que é o resultado desejado. Em compensação, é muito fácil de implementar, já que vai se operar somente sobre o bit menos significativo e não necessita de nenhuma comparação. O algoritmo de Montgomery pode ser implementado utilizando-se *Carry Save Adders* com uma representação redundante dos resultados intermediários porque somam três operandos para a obtenção de apenas dois valores.

1. S = 0; C := 0;
2. Para i = 0 até n-1 faça
3. S,C := S + C + $X_i * Y$;
4. S,C := S + C + $S_0 * M$;
5. S := S/2; C := C/2;
6. P := S + C;
7. Se P ≥ M então P := P – M;

Figura 3. Multiplicação Rápida de Montgomery

Uma modificação do algoritmo de Montgomery usando *Carry Save adders* é apresentada na Figura 3, onde se utiliza a seguinte notação:

- Dados: X,Y,M sendo $0 \leq X,Y < M$
- Resultado: $P = X*Y*2^{-n} \bmod M$
- n: número de bits em X
- x_i : i-ésimo bit de X
- s_o : bit menos significativo de S

As adições dos passos 6 e 7 são adições convencionais e por essa razão, não tão rápidas como as adições dos passos 3 e 4, mas como são executadas apenas uma vez elas são insignificantes em termos de dispêndio de tempo em relação àquelas do *loop*, que são executadas n vezes cada.

A Figura 5 mostra uma arquitetura para implementação em *hardware* do algoritmo 2 de Montgomery. A parte mais significante na ocupação de área são os dois *Carry Save Adders* necessários para se computar os valores intermediários de S e no *loop* do algoritmo 2 (rápido de Montgomery). Cada Carry save Adder é composto de **n** somadores completos.

A ocupação de *Hardware* para a implementação do loop do algoritmo 2 (chamado por nós de algoritmo rápido de Montgomery) pode ser reduzida mediante a pré-computação de quatro possíveis valores a serem somados ao resultado intermediário no interior do *loop*. Antes da divisão de **C** e **S** por 2, incrementa-se um valor **I** a **C** e **S** que podem ser obtidos de quatro formas diferentes, a saber:

1. Se a soma dos valores antigos de **C** e **S** é um número par e o bit atual de x_i de X é zero, então I = 0. Isso significa que **C** e **S** não serão incrementados.
2. Se a soma dos valores antigos de **C** e **S** é um número ímpar e o bit atual x_i de X é zero, então I = M.
3. Sendo o bit atual xi de X igual a 1, se a soma dos antigos valores de **C** e **S** é par e o valor de xi * Y também é par ou a soma dos antigos valores de **C** e **S** é ímpar e o valor de xi * Y também é ímpar, então I = Y.
4. Sendo o bit atual xi de X igual a 1, se a soma dos antigos valores de **C** e **S** é par e o valor de xi * Y é ímpar ou a soma dos antigos valores de **C** e **S** é ímpar e o valor de xi * Y é par, então I = Y + M.
5. Y + M pode ser pré-computado antes do loop. Isso economiza uma soma que pode ser trocada por uma escolha da parcela correta a ser somada aos antigos valores de S e C.

O algoritmo 3, chamado por nós de o mais rápido – do original *Faster Montgomery Multiplication*), aproveita essa idéia e é uma modificação do algoritmo 2 de Montgomery. Esse algoritmo é mais bem explicado na figura 5, e na figura 6 que apresenta uma arquitetura para este algoritmo, onde é necessário ter a seguinte notação:

Dados: X,Y,M sendo $0 \leq X,Y < M$
Resultado: $P = X*Y*2^{-n} \bmod M$
n: número de bits em X
x_i : i-ésimo bit de X
s_o : bit menos significativo de S
c_o : bit menos significativo de C
y_o : bit menos significativo de Y
R: Pré-computação de Y + M

1. S := 0; C := 0;
2. Para i = 0 até n-1 faça
3. Se ($s_o = C_o$) e not x_o então I := 0;
4. Se ($s_o \neq C_o$) e not x_o então I := M;
5. Se not ($s_o \oplus c_o \oplus y_o$) e not x_o então I := Y;
6. Se ($s_o \oplus c_o \oplus y_o$) e not x_o então I := R;
7. S,C := S + C + I;
8. S := S/2; C := C/2;
9. P := S + C;
10. Se P ≥ M então P := P – M;

Figura 4 – Multiplicação Mais Rápida de Montgomery

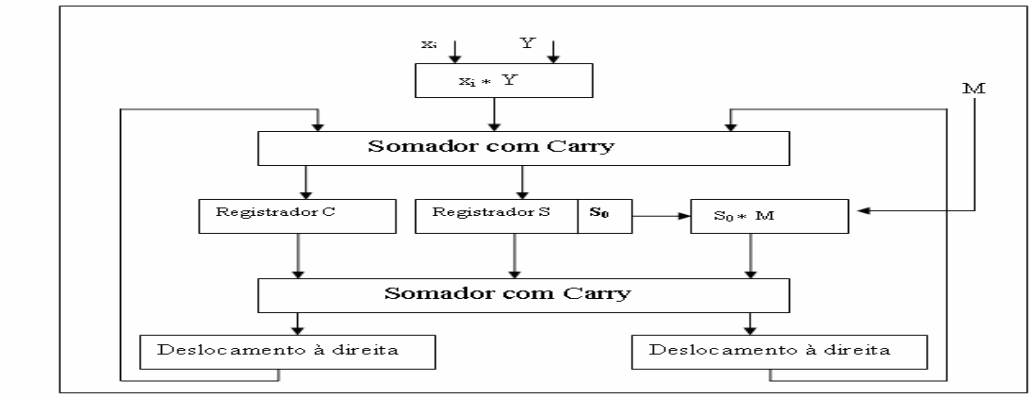

Figura 5 – Implementação em hardware do *loop* do algoritmo 2.

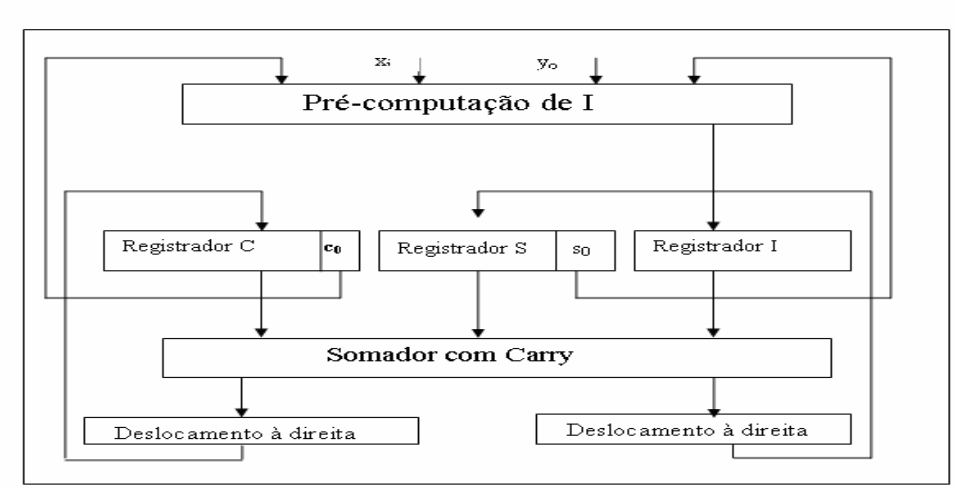

Figura 6 – Implementação do *loop* do algoritmo 3

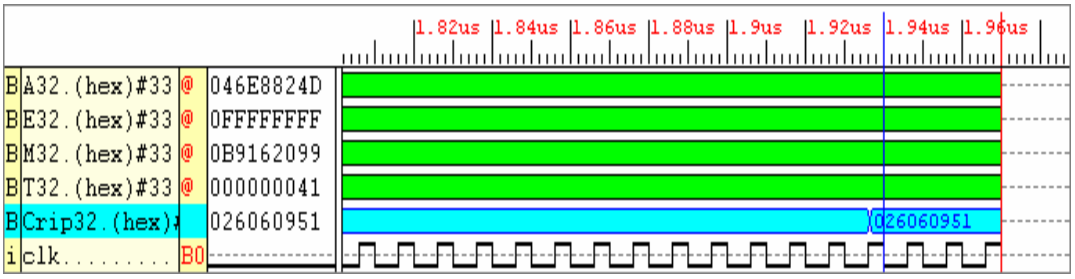

Figura 7 – Simulação de Criptografia para chave de 64 bits

4. Implementações em Hardware (FPGAs)

Nesta seção são examinadas duas implementações do algoritmo de Montgomery, descrito em VHDL. A primeira refere-se ao algoritmo rápido de Montgomery (algoritmo 2) e a segunda, ao algoritmo mais rápido de Montgomery (algoritmo 3). O maior espaço de uso de hardware desses algoritmos deve-se ao uso dos somadores (ver figuras 5 e 6).

Nessas figuras é possível observar que os registradores **S** e **C** são necessários em ambas as implementações, sendo desnecessário o registrador **I,** na implementação do algoritmo 3, mas que foi incluído por tornar mais simples a implementação. Os registradores **Y** e **M** do algoritmo 2 foram substituídos no algoritmo 3 pela pré-computação de **I**.

Com relação à velocidade do algoritmo, no caso do algoritmo 2 a computação é efetuada por dois somadores, consecutivamente, enquanto no algoritmo 3 a passagem por um dos somadores é substituída por uma comparação e uma adição, o que permite intuir que este algoritmo talvez seja bem mais rápido do que o primeiro.

A figura 7 mostra resultados da simulação do algoritmo mais rápido de Montgomery, obtido usando a ferramenta ISE da Xilinx.
A máquina de estados finitos que controla a nossa implementação do algoritmo criptográfico RSA, que utiliza a multiplicação modular baseada no algoritmo Rápido de Montgomery (*Fast Montgomery Algorithm*). Esse algoritmo foi descrito em linguagem VHDL e prototipado em FPGAs [Bissoli, 2007]. O funcionamento dessa máquina de estados pode-ser visto da seguinte forma:

- No estado 000 são feitas as inicializações.
- O estado 001 chama o estado 011 e decrementa **i** (contador dos bits de E) até localizar o primeiro bit significativo de E.
- No estado 010 recebe o valor do somador e decide se chama os estados 011, 100 ou 110, de conformidade com o algoritmo de Montgomery.
- Os estados 011, 100 ou 110, atribuem os valores às variáveis X e Y, de acordo com o algoritmo de Montgomery.
- O Estado 110 finaliza o programa e atribui à Porta de Saída **C** o valor cifrado pelo RSA.
- Deve-se notar a importância da variável Temp que faz o programa "decidir", no estado 010, se o próximo estado a ser "chamado" é o estado 011 ou 100.
- Observar também que o sinal P:= Pout no estado 010 refere-se à chamada de uma soma efetuada por um somador com *carry*, efetuada por um componente VHDL.

Na Figura 8 é apresentada a máquina de estados de nosso algoritmo RSA, em linguagem VHDL, que cifra um caracter seguindo o algoritmo Mais Rápido de Montgomery (*Faster Montgomery Algorithm*). Com relação à figura 8 devem ser feitas as seguintes considerações:

Seja **T** o texto claro; **M**, o módulo; **n** o número de bits de **T** ou de **E** (o programa foi feito para **T** e **E** terem o mesmo número de bits) e **C** o texto cifrado. Sejam **X, Y, P, Temp, flag** e **Pout** variáveis utilizadas no programa.

O funcionamento se explica através dos seguintes estados:

- No estado 000 são feitas as inicializações das portas lógicas.
- No estado 001 é feito um *loop*, sem a utilização do comando de *loop* do VHDL, para que o programa inicie a criptografia somente após ser detectado o primeiro bit significativo de E.
- No estado 010 é feito todo o processamento utilizando-se um único somador.
- O estado 011 efetua a soma de S e C, e efetua a subtração do módulo, se necessário, e "chama" os estados 100 ou 101, de acordo com o bit atual. Se o bit atual for 1, o próximo estado é 101, senão o próximo estado é 100.
- O estado 110 finaliza o programa e atribui o valor cifrado à porta de saída Crip.

Esses algoritmos foram descritos em VHDL e prototipados em um FPGA, usando da ferramenta XST da versão 8.2 do *Xilinx* e utilizando-se uma placa FPGA XC4VFX12 modelo 12FF668.
Os resultados de ocupação de recursos espaciais do FPGA e da velocidade alcançada são mostrados nas tabelas 1 e 2. Foram realizadas implementações para chaves criptográficas de tamanho variável, tendo sido possível mapear o algoritmo para chaves de 16, 32 e 64 bits.
Dos resultados das Tabela 2 é possível concluir que dos algoritmos utilizados o programa RSA que utiliza o método Mais rápido de Montgomery (algoritmo 3) é aquele que precisa de menos recursos do FPGA, o que significa que um circuito integrado teria menos área, e portanto menor tamanho do CI. A velocidade, dada pelo tempo de atraso do circuito e pela freqüência de operação da solução em hardware, também se mostrou bem mais rápida do que a versão rápida do algoritmo.
Esse desempenho superior do Algoritmo mais Rápido de Montgomery deve-se ao fato de que neste algoritmo é utilizado um único somador, enquanto que no algoritmo Rápido de Montgomery são utilizados dois somadores para se efetuar a potenciação modular.
A figura 9 mostra um comparativo da velocidade do algoritmo mais rápido implementado em software (usando linguagem C em um PC Pentium 4, 512 MB de memória), e em hardware (FPGAs) para chaves de tamanho 16, 32 e 64 bits.
A figura 10 mostra um comparativo da velocidade de cifragem obtida em hardware das duas versões do algoritmo de Montgomery, o rápido e o mais rápido. Os dados da velocidade em Mbytes/s foram calculados usando os tempos obtidos na simulação com a ferramenta da *Xilinx Foundations*, e mostrados nas tabelas 1 e 2.
O gargalo do processamento é exatamente a potenciação modular, pois os demais estados do algoritmo RSA consistem praticamente de atribuição de valores às variáveis. Assim, verifica-se na figura 8 que o algoritmo Faster (mais rápido) é aproximadamente 10% mais veloz do que o algoritmo Fast (rápido) de Montgomery em programas idênticos em linguagem VHDL.

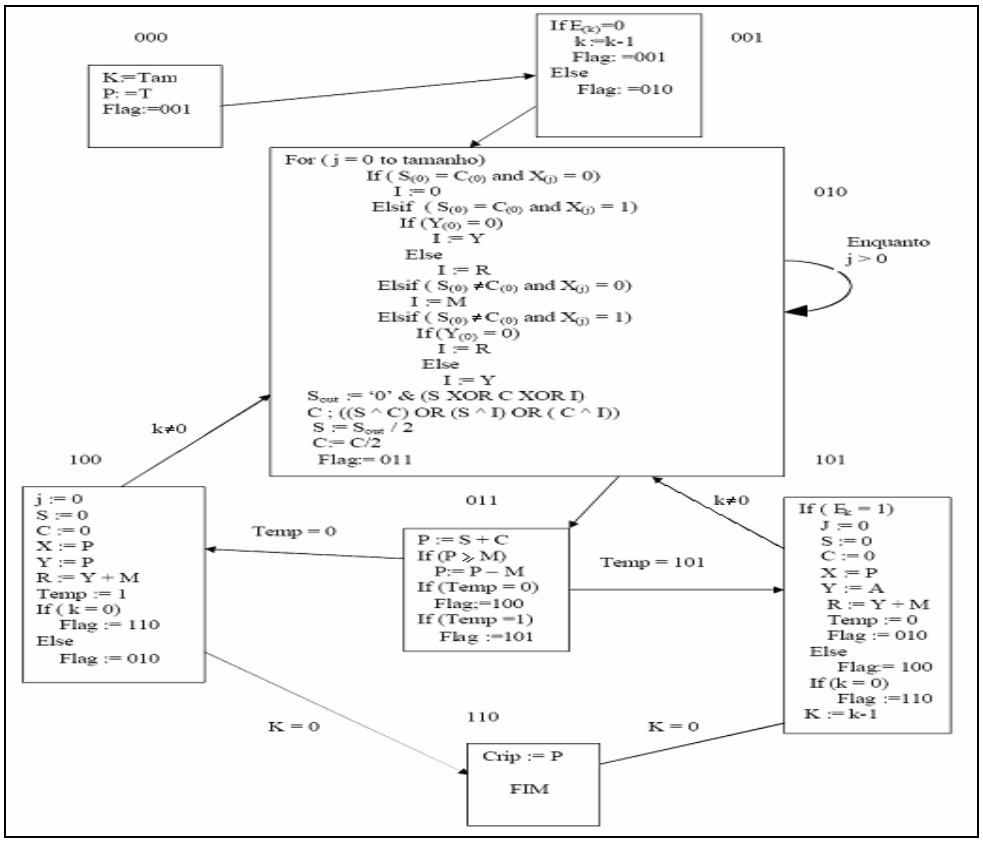

Figura 8 – Máquina de Estados do RSA usando o Algoritmo Mais rápido de Montgomery

Tabela 1. Estatísticas de Recursos Espaciais e Temporais em FPGAs – Algoritmo Rápido de Montgomery

Chaves	Slices	Flip-Flops	LUTs(4)	*TMP (ns).	Freq.Max [MHz]
16bits	386 = 7%	52 = 0%	727 = 6%	55.324	18.378
32 bits	1342= 24%	157 =1%	2582 = 23%	118.252	8.530
64 bits	4992 =91%	335 = 3%	9743 = 89%	272.347	3.688

Tabela 2. Estatísticas de Recursos Espaciais e Temporais em FPGAs – Algoritmo Mais Rápido de Montgomery

Chaves	Slices	Flip-Flop	LUTs(4)	TPM*(ns).	Freq.Max [MHz]
16 *bits*	249= 4%	70 = 0%	481 = 4%	9.359	106.211
32 bits	801 =14%	178 =1%	1557 =14%	18.944	52.943
64 bits	2979 = 54%	415 =3%	5823 =53%	44.334	22.915

***Tempo Máximo de Propagação**

Outro ponto importante de destacar é aquele observado na figura 11, que mostra as velocidades de outras três implementações da multiplicação modular. Neste caso, compara-se a velocidade obtida com o algoritmo de Brakely, o algoritmo de Gutub, e o algoritmo de Montgomery. Para mais detalhes, ver [Bissoli, 2007]. Nessas três implementações (ver figura 11) é possível observar que o atraso máximo do circuito em hardware aumenta conforme se aumenta o tamanho da chave (para os algoritmos Brakley e rápido de Montgomery), não obstante, o algoritmo de Gutub não se mostra escalável. O algoritmo rápido de Montgomery possui velocidade rápida e permanece quase constante e invariável com o aumento da chave.

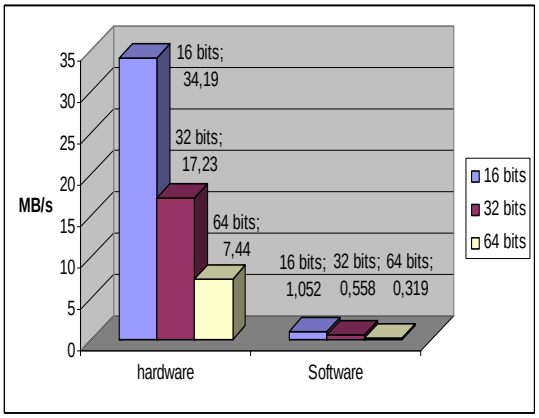

Figura 9. RSA em hardware e software

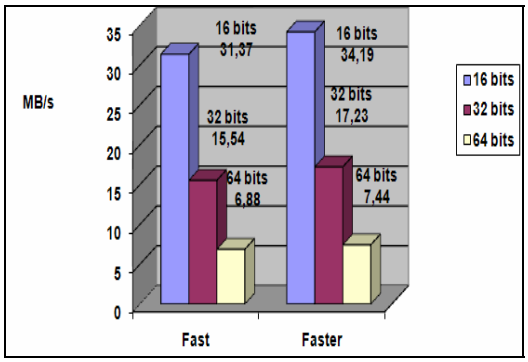

Figura 10. Velocidade do RSA usando os Algoritmos Rápido e Mais Rápido de Montgomery

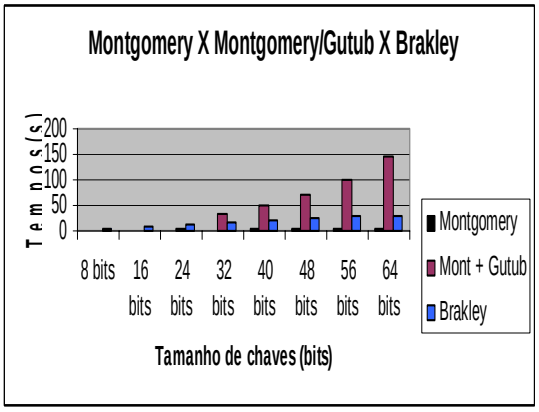

Figura 11 - Comparativo de Velocidade de criptografia com vários algoritmos

5. Conclusões

Este trabalho objetivou o estudo de algoritmos computacionais que pudessem acelerar os tempos de criptografia com o algoritmo RSA, em especial o algoritmo de Montgomery. Estudaram-se diversos algoritmos multiplicativos e exponenciais propostos por (KOÇ, 1994) e (Dally, 2003), com exemplos práticos ilustrativos. Estudou-se também, alternativamente, o método de Gutub para multiplicação modular binária, também aplicado ao RSA. Estes algoritmos foram implementados em linguagem de programação C e em linguagem de descrição de *Hardware* VHDL, com objetivo de compará-los com implementações em software e em Hardware.

Foram analisadas três versões do algoritmo de multiplicação de Montgomery, os quais foram descritos em VHDL e mapeados em um FPGA. Os dados apresentam que o algoritmo mais rápido de Montgomery é bem melhor,k pois ocupa menos área no FPGA e possui maior velocidade de cifragem.

Contudo, foi possível se verificar que, como se suspeitava, o algoritmo RSA tem desempenho em torno de 10 vezes superior em termos de velocidade de criptografia, quando se utiliza o algoritmo de Montgomery, em vez do algoritmo multiplicativo de Brakley, tanto em *software* quanto em *Hardware*.

Cifrar com chaves muito grandes é uma dificuldade que ainda não se conseguiu superar, pelo menos a baixo custo, utilizando-se de computadores comuns.

Como exemplo, a ST Microeletrônica, com filiais em 23 países do mundo, inclusive o Brasil, oferece, por exemplo, o *smart card* ST19wk08, que contém usa um processador modular aritmético de 1088 bits para criptografia assimétrica RSA, com chaves de 256 a 2048 bits (obtido em anúncio que a empresa faz pela Internet). Segundo a empresa, o hardware pode cifrar assinaturas RSA com 1024 bits na velocidade de 85 ms, utilizando o ACR (algoritmo Chinês do Resto) ou em 282 ms, sem a utilização desse algoritmo.

Segundo (HANS, 2004) pode-se cifrar até 2048 bits utilizando-se de uma Máquina de Dupla Emissão (*Dual-Issue Machine*) de dois processadores, um aritmético e o outro, controlador, que consiste numa arquitetura VLIW(*Very Long Instruction Word*) que executa um grande conjunto de operações concorrentemente), onde cada palavra de instrução contém uma instrução aritmética e outra de controle.

Referências como [Amanon, 2005], [Bailey et Al., 1999], [Dally, 2003], [Data & Rakesnake, 2002], [Dormale, 2004], [Gutub, 2000] e muitas outras se referem às dificuldades para se cifrar com chaves de tamanho muito grande e utilizam o algoritmo de Montgomery para melhorar os tempos de criptografia com o RSA. Os exemplos acima, além de reiterar a existência das dificuldades que existem para se obter segurança com a utilização de chaves assimétricas para criptografia, propõem novas arquiteturas para resolver o problema.

A seguir uma lista de idéias visando dar continuidade ao trabalho aqui realizado:

- Criar e propor novas arquiteturas que consigam ter uma boa utilização de recursos das FPGAs, diminuindo a ocupação das placas sem diminuir a velocidade (minimizando o tempo de propagação no circuito), para o algoritmo de Montgomery.
- Pesquisar um algoritmo computacional que melhore a velocidade do algoritmo de Montgomery.
- Tentar diminuir o tempo de propagação no circuito para o algoritmo de Gutub/Método de quadratura e multiplicação, já que ele utiliza menos ciclos para realizar a criptografia.
- Realizar otimizações nas descrições em linguagem VHDL realizadas, visando obter mais velocidade e menos recursos espaciais (físicos) das FPGAs.
- Tentar criar e propor arquiteturas especiais que permitam implementar esses algoritmos em FPGAs para um número maior de chaves. Uma dessas arquiteturas poderia ser a VLIW.

6. Referências

Amanon, D.N., Dissertação de Mestrado, The University of Applied Sciences Offenburg, Germany, Supervisores Prof. Dr. Angelika Erhardt e Prof. Dr. Christof Paar.

Bailey DV, Cammack W, Guajardo J, Paar C; Cryptography in Modern Communication Systems; Texas Instruments DSPS FEST, Houston, TX, Agosto, 1999.

BISSOLI, MARÇAL LUIZ. Impacto da Multiplicação e Exponenciação Modular em Hardware no Algoritmo RSA. Dissertação de Mestrado, Ciência da Computação, UNIVEM, Marilia, S.P., Brasil, 2007.

Blum T., Paar C" Modular Exponentiation on Reconfigurable Hardware" (1999), A Master Thesis, The Worcester Polythechnic Institute.

Chiaramonte R. B.; SICO: Um Sistema de Comunicação de Dados com Suporte Dinâmico à Segurança; Dissertação de Mestrado, UNIVEM, Marília, 2006.

Dally Alan, Marname L., Popovici E., Fast Modular Inversion in The Montgomery Domain on Reconfigurable Logic – ISSC, julho de 2003.

Dally A.; Marname W; Efficient Architectures for implementing Montgomery Modular mulltiplication and RSA Modular Exponentiation on Reconfigurable Logic; ACM, 2002.

Data, S.A.V.A.; Rattlesnake, J.H.A.; RSA Encryption Algorithm in a Nutshell; Neworder Bol St, 2002.

Diffie, W and M.E. Hellman, Exhaustive cryptanalysis of the NBS Data Encryption Standard; Computer 10, 1977.

Dormale, G.M., Bulens, P., Quisquater, J-J., An Improved Montgomery Modular Inversion Targeted for Efficient Implementation on FPGA; International Conference on Field-Programmable Technology, 2004.

Eberle, H., Gura N., Shantz S.C, Gupta V., Rarick L, Sundaran S, A Public-Key Cryptographic Process for RSA and ECC, 15º Conferencia Internacional IEEE para Sistemas de Aplicações Específicas, Arquiteturas e Processadores, 2004.

Gaubatz, Gunnar A. MSc Thesis Submitted to the Faculty of the Worcester Polytechnic Institute, orientadores: Dr. Berk Sunar e Dr. Fred J. Looft, abril de 2002.

Gutub, A.. Modulo Multiplication Hardware Design; Oregon State University, EGE 575, 2000.

Khaldoon M., Prototyping Of Scalable Montgomery Multiplier Using Field Programmable Gate Arrays (FPGAs); M.S. Thesis, Department of Electrical & Computer Engineering, Oregon State University, July 23, 2002.

Kalisy Jr, B.S., The Montgomery Inverse and its Applications; IEEE, 44(8), pg.1064-1065, Agosto de 1995.

Knuth, E. The Art of Computer Programming - Seminumerical Algorithms; Addison-Wesley, 1997.

Koç Ç. K.; Acar T; Kalisky Jr, B.S.. – Analyzing and Comparing Montgomery Multiplication Algorithms; IEEE Micro, v.16 n.3, p.26-33, June 1996

Koç, Ç, K.; Montgomery Reduction With Even Modulus; IEEE Proceedings: Computers and Digital Techniques – Setembro de 1994.

Muzzi, F. A. G., O Padrão de Segurança PKCS#11 em FPGA's – RSA um Estudo de Caso; Dissertação de Mestrado, UNIVEM, Marília, 2006.

Mctaggart M., Introduction to cryptography, Part 3: Asymmetric cryptography, 01.03.2001, p. da IBM, Internet, abril de 2006

Moreno E.D. et Al. Criptografia em Software e hardware; Livro pela Editora Novatec, pg. 288 – 2005

Prince, B.J., Scalable Montgomery Multiplication Algorithm; Oregon State University, Department of Electrical & Computer Engineering, 2002.

Rivest, R.; Shamir, A.; Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; Communications of ACM, 1978.

Savas E,; Koç, Ç.K. The Montgomery Modular Inverse – Revisited; IEEE, Special issue on computer arithmetic, Vol.49, nº 7, julho de 2000, pg 763-766.

S. C. Shantz S. C. Shantz, "From Euclid's GCD to Montgomery multiplication to the great divide," Tech. Rep. TR-2001-95, Sun Microsystems Laboratories, Santa Clara, Calif, USA, June 2001.

Todorov G.; Tenca A.F.; ASIC Design, Implementations and Analysis of a Scalable High-Radix Montgomery Multiplier – M.S.Thesis - Dezembro – 2000.