UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ARQUITETURAS DE CRIPTOGRAFIA DE CHAVE PÚBLICA: ANÁLISE DE DESEMPENHO E ROBUSTEZ

DISSERTAÇÃO DE MESTRADO

Guilherme Perin

Santa Maria, RS, Brasil
2011

ARQUITETURAS DE CRIPTOGRAFIA DE CHAVE PÚBLICA: ANÁLISE DE DESEMPENHO E ROBUSTEZ

por

Guilherme Perin

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-Graduação em Informática, Área de Concentração em Microeletrônica e Processamento de Sinais, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Informática.

Orientador: Prof. Dr. João Baptista dos Santos Martins

Santa Maria, RS, Brasil

2011

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

ARQUITETURAS DE CRIPTOGRAFIA DE CHAVE PÚBLICA: ANÁLISE DE DESEMPENHO E ROBUSTEZ

elaborada por
Guilherme Perin

como requisito parcial para obtenção do grau de
Mestre em Informática

COMISSÃO EXAMINADORA:

João Baptista dos Santos Martins, Dr.
(Presidente/Orientador)

Ney Laert Vilar Calazans, Dr. (PUCRS)

José Eduardo Baggio, Dr. (UFSM)

Santa Maria, 15 de Abril de 2011

Dedico este trabalho à Cris Dietrich.

AGRADECIMENTOS

Gostaria de agradecer à minha esposa Cris Dietrich, por todo o carinho, paciência e apoio prestados durante a realização deste trabalho de mestrado. Sem ela, este trabalho não teria existido.

Agradeço à toda a minha família, principalmente aos meus pais, Melânia e Ivanir Perin, aos meus sogros, Irma e João Dietrich, por todo o incentivo, apoio, amizade e convivência durante os 24 meses de trabalho.

Agradeço ao meu orientador de mestrado, João Baptista dos Santos Martins, pela orientação e oportunidades.

Aos colegas e professores do Gmicro, quero agradecer por toda a convivência, aprendizado e bons momentos divididos durante o mestrado.

Quero agradecer também aos amigos de Santa Maria, pelos ótimos momentos convividos.

Agradeço especialmente ao pessoal do LIRMM, pela receptividade, oportunidades e todo o aprendizado proporcionado, especialmente aos professores Philippe Maurine, Lionel Torres e Pascal Benoit.

Também sou muito grato ao Programa de Pós-Graduação em Informática da UFSM, à CAPES e ao CNPq.

“Quando os ventos de mudança sopram, algumas pessoas levantam barreiras, outras constroem moinhos de vento.”

Érico Veríssimo, O Tempo e o Vento

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria, RS, Brasil

ARQUITETURAS DE CRIPTOGRAFIA DE CHAVE PÚBLICA: ANÁLISE DE DESEMPENHO E ROBUSTEZ

AUTOR: GUILHERME PERIN
ORIENTADOR: JOÃO BAPTISTA DOS SANTOS MARTINS

Local da Defesa e Data: Santa Maria, 15 de Abril de 2011.

Com a expansão da área de comunicação de dados e o consequente aumento do fluxo de informações, a segurança tem se tornado uma grande preocupação. Apesar dos métodos criptográficos modernos serem matematicamente seguros, sua implementação em *hardware* tende a apresentar fugas de informações confidenciais por canais laterais, tais como consumo de potência e emissões eletromagnéticas. Embora questões de desempenho sejam cruciais para um projeto de *hardware*, aspectos de robustez contra ataques baseados em fugas de informações por canais laterais tem ganhado maior atenção nos últimos anos.

Neste trabalho, explora-se arquiteturas em *hardware* voltadas para o algoritmo de chave pública RSA, originalmente proposto em 1977 por Rivest, Shamir e Adleman. Este algoritmo possui como principal operação a exponenciação modular, e esta é calculada através de sucessivas multiplicações modulares. Sendo que o RSA envolve números inteiros da ordem de 1024 *bits* ou mais, a operação de divisão inerente em multiplicações modulares torna-se o principal problema. O algoritmo de Montgomery, proposto em 1985, é um método bastante utilizado na implementação da multiplicação modular em *hardware*, pois além de evitar divisões, trabalha em um contexto de precisão múltipla com termos representados por bases numéricas, geralmente, potências de dois.

Dentro deste contexto, propõe-se inicialmente uma arquitetura sistólica, baseada nas propriedades de aritmética de precisão múltipla do Algoritmo de Montgomery. Em seguida, apresenta-se uma melhoria para a arquitetura sistólica, através de uma arquitetura que realiza a multiplicação modular de Montgomery voltada à multiplexação dos processos aritméticos. A arquitetura multiplexada é empregada nos métodos de exponenciação modular *left-to-right square-and-multiply* e *square-and-multiply always* e é submetida a ataques por canais laterais SPA (*Simple Power Analysis*) e SEMA (*Simple Electromagnetic Analysis*) e aspectos de robustez da arquitetura multiplexada são analisados para diversos tamanhos de palavras (base numérica do algoritmo de Montgomery). Como proposta de melhoria aos ataques por canais laterais simples, os traços de consumo de potência e emissão eletromagnética são demodulados em amplitude de modo a eliminar a influência das harmônicas do sinal de *clock* sobre os traços coletados. Por fim, interpretações e conclusões dos resultados são apresentados, assim como propostas de contra-medidas para a arquitetura multiplexada com relação aos ataques por canais laterais realizados.

Palavras-chave: Criptografia de Chave Pública, RSA, Exponenciação Modular, Multiplicação Modular de Montgomery, Ataques por Canais Laterais.

ABSTRACT

Master's Dissertation
Post-Graduation Program in Informatics
Federal University of Santa Maria, RS, Brazil

PUBLIC-KEY CRYPTOGRAPHY ARCHITECTURES: PERFORMANCE AND ROBUSTNESS EVALUATION

AUTHOR: GUILHERME PERIN
ADVISOR: JOÃO BAPTISTA DOS SANTOS MARTINS

Place and Date: Santa Maria, April 15th, 2011.

Given the evolution of the data communication field, and the resulting increase of the information flow in data, networks security became a major concern. Modern cryptographic methods are mathematically reliable. However their implementation in *hardware* leaks confidential information through side-channels like power consumption and electromagnetic emissions. Although performance issues are crucial for a hardware design, aspects of robustness against attacks based on side-channel informations have gained much attention in recent years.

This work focuses on hardware architectures based on the RSA public-key algorithm, originally proposed in 1977 by Rivest, Shamir and Adleman. This algorithm has the modular exponentiation as its main operation and it is performed through successive modular multiplications. Because the RSA involves integers of 1024 bits or more, the inherent division of modular multiplications became the main concern. The Montgomery algorithm, proposed in 1985, is a largely used method for hardware designs of modular multiplications, because it avoids divisions and all operations are performed in a multiple-precision context with all terms represented in a numerical base, generally, a power of two.

This dissertation proposes a systolic architecture able to perform the Montgomery modular multiplication with multiple-precision arithmetic. Following, an improvement to the systolic architecture is presented, through an architecture that computes the Montgomery multiplication by multiplexing the multi-precision arithmetic processes. The multiplexed architecture is employed in the left-to-right square-and-multiply and square-and-multiply always modular exponentiation methods and is subjected to SPA (*Simple Power Analysis*) and SEMA (*Simple Electromagnetic Analysis*) side-channel attacks and robustness aspects are analysed. Different word sizes (numerical bases) are applied as well as different input operands. As an improvement to SPA and SEMA attacks, the power consumption and electromagnetic traces are demodulated in amplitude to eliminate the clock harmonics influence in the acquired traces. Finally, interpretations, conclusions and countermeasure propositions to the multiplexed architecture against the implemented side-channel attacks are presented.

Keywords: Public-Key Cryptography, RSA, Modular Exponentiation, Montgomery Modular Multiplication, Side-Channel Attacks

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O crescimento da área de comunicações de dados e a consequente expansão do fluxo de informações tornaram os dispositivos eletrônicos mais vulneráveis em termos de segurança. Comércio eletrônico, sistemas de *home banking* e conversações em tempo real fazem parte de um cenário que necessita cada vez mais de segurança da informação. A criptografia de chave pública é uma tecnologia largamente utilizada no provimento de segurança em comunicações eletrônicas.

Os dispositivos eletrônicos que necessitam de segurança da informação normalmente executam algum algoritmo de criptografia em *software*. Esses sistemas de cifração de dados deve fornecer uma resposta rápida de modo a não comprometer o desempenho do sistema. Para isso, também utiliza-se circuitos integrados como aceleradores para esse processo, uma vez que a implementação em *hardware* de uma aplicação específica tende a manter um desempenho constante e confiável. Por trás dessas arquiteturas de *hardware* estão algoritmos aritméticos eficientes que tem por objetivos prover melhor desempenho e aplicação dos métodos de criptografia.

Mesmo fornecendo dados criptografados, dispositivos criptográficos implementados em *hardware* são sistemas físicos e podem eventualmente apresentar vazamento de informações confidenciais por canais ocultos, tais como consumo de potência, emissões eletromagnéticas e tempo de execução. Pesquisas recentes mostram que tanto o consumo de potência quanto as emissões eletromagnéticas de um dispositivo criptográfico podem ter uma forte relação com os dados sendo processados. Como consequência, um adversário pode observar as informações vazadas por estes canais ocultos e revelar informações importantes, incluindo a chave de criptografia. Esta ação é conhecido como ataque por canais laterais.

Portanto, implementações em *hardware* de algoritmos de criptografia devem atender critérios de desempenho e segurança.

1.1 Objetivos

Este trabalho propõe duas arquiteturas de criptografia de chave pública (Diffie,1976) tendo como método de cifração o algoritmo RSA, que recebe esta denominação em função das iniciais dos nomes de seus autores Rivest, Shamir e Adleman (Rivest,1978). Com estas arquiteturas, é realizada uma análise de desempenho e robustez. As arquiteturas foram desenvolvidas com base no algoritmo de Montgomery para multiplicação modular (Montgomery,1985) e métodos de exponenciação modular são empregados nos processos de cifração e decifração.

A análise de desempenho é medida através de dois critérios:

- Recursos lógicos necessários quando as arquiteturas são sintetizadas em FPGAs (*Field Programmable Gate Array*).
- Tempo de execução de uma multiplicação modular de Montgomery para as arquiteturas propostas e tempo de execução de uma exponenciação modular em função do tamanho do expoente. Neste último caso, pode-se apurar o tempo de cifração e decifração do algoritmo RSA.

A análise de robustez é realizada através da aplicação de ataques por canais laterais sobre as arquiteturas propostas. Traços de consumo de potência e emissão eletromagnética são adquiridos das arquiteturas quando estas realizam processos de exponenciação modular. Duas categorias de ataques são aplicados:

- Ataques SPA (Simple Power Analysis (Kocher,1999)) e SEMA (Simple Electromagnetic Analysis (Gandolfi,2001)(Quisquater,2001)). De modo a validar estes ataques, estes são primeiramente aplicados sobre uma arquitetura RSA padrão (Opencores,2010).
- Ataques SPA e SEMA baseados na demodulação em amplitude dos traços de consumo de potência e emissão eletromagnética, denotados por SDPA (*Simple and Demodulated Power Analysis*) e SDEMA (*Simple and Demodulated Electromagnetic Analysis*). Neste caso, a influência do sinal de *clock* é suprimido dos traços para melhor recuperar a informação confidencial.

Por fim, este trabalho também tem como objetivos apresentar contra-medidas à nível de algoritmo ou arquitetura frente aos ataques realizados.

1.2 Organização do Trabalho

No Capítulo 2 são apresentados os conceitos básicos de criptografia e criptoanálise. Em seguida, são apresentadas fundamentações teóricas para o entendimento dos principais métodos de exponenciação modular e do Algoritmo de Montgomery para multiplicação modular. Em seguida, são descritos os principais ataques por canais laterais e suas aplicações sobre o algoritmo RSA.

O Capítulo 3 apresenta uma revisão sobre trabalhos relacionados. Arquiteturas baseadas em matrizes sistólicas são abordadas com ênfase em questões de desempenho.

As arquiteturas propostas são descritas em detalhes no Capítulo 4. Primeiramente, descreve-se a arquitetura sistólica e o fluxo de operações aritméticas dos elementos de processamento. Em seguida, a arquitetura multiplexada é proposta como uma nova abordagem para questões de desempenho e flexibilidade.

O Capítulo 5 apresenta uma análise de robustez da arquitetura multiplexada na qual, primeiramente, ataques SPA (*Simple Power Analysis*) e SEMA (*Simple Electromagnetic Analysis*) são realizados. Esta arquitetura é empregada em dois métodos de exponenciação modular: *left-to-right square-and-multiply* (Menezes,2001) e *square-and-multiply always* (Coron,1999). Esta análise simples de potência e emissões eletromagnéticas destaca tanto os pontos de fugas de informação da arquitetura quanto as partes que apresentam maior robustez. De modo a validar os ataques SPA e SEMA, estes são empregados sobre uma arquitetura RSA padrão implementada com o algoritmo de exponenciação modular *right-to-left square-and-multiply*. Como melhoria dos ataques simples, apresenta-se também neste capítulo ataques baseados em sinais demodulados em amplitude. Este capítulo também apresenta interpretações dos resultados e possíveis contra-medidas algorítmicas e arquiteturais.

Finalmente, o Capítulo 6 apresenta conclusões do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma revisão teórica de algoritmos de criptografia e tópicos matemáticos abordados neste trabalho. Inicialmente, apresenta-se fundamentos de criptologia, abordando-se os tópicos de criptografia e criptoanálise. Dentro desses assuntos apresenta-se uma breve descrição de criptografia de chave privada e pública, com detalhes no algoritmo RSA (Rivest, 1978). Na sequência, discute-se aspectos relacionados à criptoanálise, focando-se na segurança em *hardware* frente a ataques invasivos, semi-invasivos e não-invasivos. Em termos de algoritmos e métodos matemáticos, descreve-se os algoritmos de exponenciação modular mais comumente empregados em criptografia de chave pública. A seguir, o algoritmo de Montgomery é apresentado em detalhes, junto com aspectos relacionados à aritmética de precisão múltipla e aritmética modular. Por fim, este capítulo apresenta uma abordagem detalhada de ataques por canais laterais (*Side-Channel Attacks*), enfatizando-se aspectos de segurança dos algoritmos de exponenciação modular, a principal operação do RSA.

2.1 Criptologia

A criptologia é a prática e o estudo da ocultação da informação e da solução de problemas de criptografia. Divide-se, geralmente, em dois sub-tópicos:

- Criptografia, a qual consiste no projeto de primitivas e protocolos de modo a suprir os requisitos de segurança da informação, através de processos de proteção de dados;
- Criptoanálise, a qual consiste em ultrapassar os critérios de segurança oferecidos pelos sistemas criptográficos.

Ambas atuam de modo complementar, sendo que a criptografia é um conjunto de técnicas com o objetivo de esconder uma mensagem e a criptoanálise é formada por estudos de propriedades e processos que permitam quebrar sistemas de criptografia.

2.1.1 Criptografia

A criptografia abrange alguns dos critérios da segurança da informação, tais como:

- Confidencialidade: é um serviço usado de modo a manter intacto o conteúdo da informação. Existem diversas abordagens para fornecer confidencialidade, variando desde proteções físicas até transformações matemáticas de modo a tornar os dados ininteligíveis.

- Integridade dos dados: é um serviço que visa detectar a alteração não autorizada de dados. Para assegurar a integridade dos dados, é preciso ter a capacidade de detectar a manipulação dos dados por terceiros não autorizados. Manipulação de dados inclui exclusão, inserção e substituição.

- Autenticação: é um serviço relacionado à identificação de usuários em sistemas que necessitem de segurança da informação. Esta função aplica-se à ambas as entidades envolvidas na comunicação e à própria informação. Duas partes que entram em uma comunicação devem identificar-se uma à outra. A informação entregue à um canal de comunicação deve ser autenticada com a origem, data da origem, conteúdo dos dados, hora enviada, etc. Por estas razões, este aspecto da criptografia é usualmente subdividido em duas grandes partes: autenticação das entidades e autenticação da origem dos dados. Esta última implicitamente fornece integridade dos dados, pois se a mensagem é modificada, significa que a fonte foi alterada.

- Não-repúdio: é um serviço que evita a negação de ações e autorizações por parte de uma entidade. Assim, o não-repúdio da informação permite verificar que as entidades de origem e destino são as partes que desejam estabelecer uma comunicação. O não-repúdio, portanto, garante que os dados sejam entregues ao destinatário.

As técnicas de criptografia subdividem-se em três classes principais: algoritmos simétricos (ou de chave privada), algoritmos assimétricos (ou de chave pública) e protocolos de criptografia. Algoritmos simétricos fazem parte de um contexto no qual duas partes de um sistema de comunicação, possuindo o mesmo sistema de cifração e decifração, compartilham a mesma chave. No caso de algoritmos de chave pública, cada usuário possui a sua chave secreta, usada na decifração, e compartilham uma chave pública, usada na cifração. E, de um modo geral, protocolos de criptografia tratam da aplicação dos algoritmos simétricos e assimétricos.

2.1.1.1 Criptografia Simétrica

A criptografia simétrica baseia-se no conceito de que a mesma chave é usada para cifrar e decifrar uma mensagem. Este sistema implica que duas entidades compartilham a mesma chave através de um canal seguro. Um canal seguro garante a confidencialidade, autenticidade e integridade da informação transmitida sobre este canal.

A criptografia simétrica compreende duas sub-classes de funções de cifração: algoritmos de cifra de bloco (*block cipher*) e algoritmos de cifra de fluxo (*stream cipher*). A cifração de bloco compreende um cifrador de chave simétrica operando em grupos de *bits* de tamanho fixo, chamados blocos. Como exemplos de algoritmos de cifração por blocos, tem-se o *Data Encryption Standard* (DES) (DES,1999) e o *Advanced Encryption Standard* (AES) (AES,2001). Estes algoritmos repetem o mesmo conjunto de operações várias vezes, sendo cada um destes chamado *round*.

Os algoritmos de cifração de fluxo são essencialmente geradores pseudo-aleatórios que permitem obter uma chave de criptografia como uma sequência longa de *bits*, dita sequência de cifração. A cifração de fluxo tende a ser muito mais rápida que a cifração por bloco, pois os algoritmos de cifra de fluxo operam tipicamente em pequenas unidades de textos claros, usualmente *bits*. O texto claro é a mensagem a ser transmitida. Além disso, a cifração de um texto claro em particular com a cifração por bloco resulta no mesmo texto cifrado quando a mesma chave é usada. Por outro lado, na cifração por fluxo a transformação deste mesmo texto claro varia para cada processo de cifração. O exemplo mais clássico de algoritmo de criptografia de cifração de fluxo é o RC4, proposto por Ronald Rivest em 1987.

2.1.1.2 Criptografia Assimétrica

A criptografia assimétrica (ou de chave pública) foi inicialmente proposta em 1976 por Whitfield Diffie e Martin E. Hellman em (Diffie,1976), como um método adequado de comunicação entre duas entidades através de um canal inseguro, ou seja, que pode ser observado por qualquer adversário.

Esse sistema parte do princípio de que uma das entidades gera duas chaves de criptografia, uma chave pública e uma chave privada, e envia a chave pública sobre um canal de comunicação inseguro. A outra entidade envolvida na comunicação utiliza esta chave pública para cifrar sua mensagem. Por fim, a mensagem cifrada só pode ser decifrada através da chave privada, que não é enviada sobre o canal de comunicação.

Desse modo, a criptografia assimétrica resolve o principal problema da criptografia simé-

trica: como compartilhar a chave secreta sobre um canal inseguro de comunicação. Como exemplo para esse processo, tem-se esquema de troca de chaves de Diffie-Hellman. Apesar de suprir o inconveniente da troca de chaves, métodos de criptografia de chave pública empregam números de grande magnitude nos processos de cifração e decifração, sendo, portanto, um processo de comunicação mais lento que no caso da criptografia simétrica.

Existem três famílias de algoritmos de chave pública de maior relevância prática. Estes algoritmos são classificados de acordo com seu problema computacional:

1. **Esquema de Fatoração de Inteiros:** vários esquemas de chave pública são baseados no fato de que fatorar números inteiros grandes é uma tarefa muito difícil. O mais conhecido algoritmo desta família é o RSA.

2. **Esquema do Logaritmo Discreto:** neste esquema, enquadram-se algoritmos baseados no problema do logaritmo discreto para campos finitos. Como exemplos, têm-se o esquema de troca de chaves de Diffie-Hellman, Elgamal e o DSA (*Digital Signature Algorithm*)

3. **Esquema de Curvas Elípticas:** as generalizações do esquema do logaritmo discreto são os esquemas de chave pública com curvas elípticas. Os exemplos mais comuns incluem ECDH (*Elliptic Curve Diffie-Hellman key exchange* (Diffie,1976)), ECDSA (*Elliptic Curve Digital Signature Algorithm*) e o ECC (*Elliptic Curve Cryptography* (Koblitz,1987)).

A seguir apresenta-se o algoritmo de chave pública RSA em detalhes, enfatizando-se aspectos práticos de cálculos de parâmetros, cifração e decifração.

2.1.1.3 O Algoritmo RSA

O algoritmo RSA foi publicado em 1977 por Ron Rivest, Adi Shamir e Len Adleman em (Rivest,1978). Sua aplicação varia entre sistemas que requerem cifração de pequenas porções de dados (transporte de chaves) e assinaturas digitais. A segurança deste método criptográfico está na dificuldade de fatoração de números inteiros grandes.

A geração das chaves para o algoritmo RSA se dá da seguinte maneira:

1. Gera-se dois números primos aleatórios grandes p e q.

2. Calcula-se $n = p \times q$ e $\phi(n) = (p-1)(q-1)$. O termo $\phi(n)$ é denominado Função Totiente de Euler (Menezes,2001). Esta função define o número de inteiros positivos menores do que n, os quais são primos entre si com n.

3. Considerando-se que a função *gcd(a,b)* refere-se ao máximo divisor comum de a e b, defini-se um número inteiro e, de modo que, $1 < e < \phi(n)$ e gcd$(e, \phi(n))= 1$ ou seja, e e $\phi(n)$ devem ser primos entre si.

4. Calcula-se d, de modo que $d.e \equiv 1(mod\phi(n))$ e $1 < d < \phi(n)$.

O par (e,n) é conhecido como chave pública e o par (d,n) como chave privada. Após a geração das chaves criptográficas do RSA, resta apenas aplicá-las nos processos de cifração e decifração de uma mensagem. Supondo que m seja a mensagem a ser cifrada, o processo de cifração é realizado fazendo-se uso do par chave pública (e,n), através da seguinte equação:

$$c = m^e mod(n) \tag{2.1}$$

Para recuperar a mensagem m, a partir da mensagem cifrada c, utiliza-se o par chave privada (d,n), através da seguinte equação:

$$m = c^d mod(n) \tag{2.2}$$

Assim, o algoritmo RSA tem como principal operação aritmética a *exponenciação modular*, como mostram as equações (2.1) e (2.2). Apesar da publicação dos termos e e n, o algoritmo RSA é considerado matematicamente seguro pois, para encontrar a chave privada d é necessário fatorar o módulo n em dois números primos p e q. Geralmente utilizam-se chaves criptográficas de 1024 *bits*, podendo-se chegar até 4096 *bits*. Fatorar números inteiros desta magnitude é computacionalmente inviável com a tecnologia atual.

O maior desafio em termos de implementações em *hardware* do RSA concentra-se na dificuldade de elaborar uma arquitetura rápida que realize o processo de exponenciação modular com números inteiros grandes. A solução é o emprego de algoritmos matemáticos que auxiliem em processos aritméticos mais complexos. Portanto, a seguir apresenta-se os métodos de exponenciação modular mais comumente empregados nos processos de cifração e decifração do RSA.

2.1.1.4 Métodos de Exponenciação Modular

Uma das operações aritméticas mais importantes de criptografia de chave pública é a exponenciação. Realizar uma exponenciação modular significa obter o resto da divisão de um inteiro positivo B (chamado *base*), elevado à i-ésima potência E (chamado *expoente*), por um inteiro

positivo N (chamado *módulo*). Portanto, dados B, N e E, calcula-se C da seguinte forma:

$$C = B^E \ mod \ N \tag{2.3}$$

Para o algoritmo RSA, no caso da cifração, o expoente representa parte da chave pública E e a base representa a mensagem a ser cifrada. Por outro lado, a decifração emprega como expoente parte da chave privada D e a base representa a mensagem C a ser decifrada. Para ambos os processos, o módulo N resulta do produto entre dois números primos $N = p \times q$.

Os termos envolvidos no cálculo da exponenciação modular para o algoritmo RSA são representados com valores entre 1024 e 4096 *bits*. Portanto, métodos binários de exponenciação modular são comumente adotados. Nesse caso, o expoente E é dado na base 2 (binária) e interpretado *bit*-à-*bit*. Assim, o cálculo é quebrado em sucessivas multiplicações modulares $A.B \ mod \ N$ e quadrados modulares $A.A \ mod \ N$, sendo A e B resultados intermediários da exponenciação modular.

Em implementações em *hardware* do RSA, os métodos de exponenciações modulares *left-to-right square-and-multiply* e *right-to-left square-and-multiply* (Menezes,2001) são frequentemente empregados. Os dois métodos são mostrados na Figura 1.

Algoritmo 1 - Left-to-Right Square-and-Multiply

Entrada: mensagem M, expoente $E = \sum_{i=0}^{n-1} e_i.2^i$ e modulo N.
Saída: $C = A^E \bmod N$

```
A = 1;
B = M;
for i = n-1 to 0 do
    A = A.A mod N
    if e_i = 1 then
        A = A.B mod N
    end if;
end for;
C = A
```

(a)

Algoritmo 2 - Right-to-Left Square-and-Multiply

Entrada: mensagem M, expoente $E = \sum_{i=0}^{n-1} e_i.2^i$ e modulo N.
Saída: $C = A^E \bmod N$

```
A = 1;
B = M;
for i = 0 to n-1 do
    if e_i = 1 then
        A = A.B mod N
    end if;
    B = B.B mod N
end for;
C = A
```

(b)

Figura 1: Algoritmos de Exponenciação Modular (a) Left-to-Right Square-and-Multiply and (b) Right-to-Left Square-and-Multiply.

Para os Algoritmos 1 e 2 apresentados na Figura 1, a cada interpretação de um *bit* do expoente e_i, uma operação de quadrado modular é executada. Caso o *bit* e_i seja 1, uma operação de multiplicação modular deve ser executada em série (*left-to-right*) ou pode ser executada em paralelo (*right-to-left*) à operação de quadrado modular.

Assim, uma arquitetura que visa o cálculo de uma exponenciação modular baseia-se no projeto de uma arquitetura para executar multiplicações modulares. Da mesma forma, os sucessivos processos de multiplicação modular envolvem operações matemáticas com números inteiros grandes, mais precisamente, da mesma magnitude do módulo N. Considerando que calcular ($A.B \ mod \ N$) requer uma divisão por N, o cálculo da multiplicação modular torna-se, portanto, o grande gargalo da implementação do RSA em *hardware*.

Desta forma, apresenta-se a seguir o algoritmo de Montgomery e suas propriedades matemáticas que tornam adequada a implementação da multiplicação modular em *hardware*.

2.1.1.5 Multiplicação Modular e o Algoritmo de Montgomery

A multiplicação modular é considerada uma operação aritmética complexa em função das operações de multiplicação e divisão que lhe são inerentes. O algoritmo de Montgomery (Montgomery,1985) é um método para o cálculo da multiplicação modular ($A.B \ mod \ N$) sem a necessidade de empregar uma divisão por N. Assim, o algoritmo de Montgomery é bastante adequado para implementações em *hardware* da multiplicação modular, pois permite que números inteiros grandes sejam representados em uma precisão numérica dada por uma base, geralmente uma potência de dois. Assim, todas as divisões deste algoritmo são realizadas através de simples operações de deslocamento de *bits* à direita.

A Figura 2 apresenta o algoritmo de Montgomery com a notação apresentada em (Menezes,2001).

Algoritmo 3: Multiplicação Modular de Montgomery para o cálculo de ABR^{-1} mod N.

$N = \sum_{i=0}^{m-1}(2^k)^i n_i, \ n_i \in \{0, 1, ..., 2^k-1\}$
$A = \sum_{i=0}^{m-1}(2^k)^i a_i, \ a_i \in \{0, 1, ..., 2^k-1\}$
$B = \sum_{i=0}^{m-1}(2^k)^i b_i, \ b_i \in \{0, 1, ..., 2^k-1\}$
$R = (2^k)^m, \ A, B < 2N; \ N < R = (2^k)^m$
$N' = -N^{-1} \bmod 2^k$

1. $S_0 = 0$;
2. for $i = 0$ **to** $m-1$ **do**
3. $q_i = ((S_0 + a_i b_i)N') \bmod 2^k$
4. $S_{i+1} = (S_i + q_i N + a_i B)/2^k$
5. end for
6. if $S \geqslant N$ **then**
7. S = S - N
8. end if

Figura 2: Algoritmo de Montgomery.

O Algoritmo de Montgomery apresenta como resultado $S = A.B.R^{-1}\ mod\ N$, na qual A e B são os operandos de entrada e R é comumente chamada de constante de Montgomery, dada através de uma potência de dois.

O dígito q_i é escolhido de modo que $S_i + a_iB + q_iN$ seja divisível por 2^k. Uma vez que o algoritmo garante que os k *bits* menos significativos de S_{i+1} sejam iguais à zero, essa divisão resume-se à uma operação de deslocamento à direita de k *bits*. A operação $mod\ 2^k$ no cálculo do dígito q_i é facilmente computada por um truncamento dos k *bits* menos significativos. O termo N' é calculado de modo que $-N^{-1}.N' \equiv 1\ mod\ 2^k$, para $i < k$. Neste trabalho, assume-se que N' é um valor pré-calculado.

Se os operandos de entrada A e B são menores do que N, o resultado S e os resultados intermediários S_{i+1} serão limitados por $2N$. Devido à essa limitação, uma subtração final deve ser efetuada caso $S \geq N$. Porém, como mostrado em (Walter,1999), na aplicação da multiplicação modular de Montgomery em uma exponenciação modular, essa subtração final pode ser evitada se $A, B < N$ e $N < 2^{km}$. Para tanto, os operandos de entrada da exponenciação A e B (conforme Figura 1) devem estar no domínio de Montgomery:

$$\begin{aligned} A &= mont(1, R^2) = 1.R^2.R^{-1}\ mod\ N = R\ mod\ N \\ B &= mont(M, R^2) = M.R^2.R^{-1}\ mod\ N = M.R\ mod\ N \end{aligned} \tag{2.4}$$

Como consequência, todos os resultados das multiplicações modulares no decorrer de uma exponenciação modular carregam a constante de Montgomery R. De modo a suprimir esta constante, ao final da exponenciação modular, uma última multiplicação modular de Montgomery é computada tendo como parâmetros de entrada o último resultado A no domínio de Montgomery ($AR\ mod\ N$) e 1:

$$A = mont(A, 1) = A.R.1.R^{-1}\ mod\ N = A\ mod\ N \tag{2.5}$$

Assim, o resultado final fica limitado a N. A Figura 3 apresenta a exponenciação modular de Montgomery:

Assim, a associação do método de Montgomery com a exponenciação modular supre os inconvenientes oferecidos pela manipulação de números inteiros grandes. Todas as operações da multiplicação modular de Montgomery são dadas em um contexto de precisão múltipla, isto é, os operandos de entrada são representados por uma base 2^k, o que facilita a manipulação de números inteiros da ordem de 1024 *bits* ou mais. Para o caso do RSA, o tamanho dos operandos

Algoritmo 4 - Left-to-Right Square-and-Multiply: Exponenciação Modular de Montgomery

Entrada: mensagem M, expoente $E = \sum_{i=0}^{n-1} e_i.2^i$, modulo N, $R = 2^n$.

Saída: $C = A^E \bmod N$

```
A = mont(1,R^2)
B = mont(M,R^2)
for i = n-1 to 0 do
    A = mont(A,A)
    if e_i = 1 then
        A = mont(A,B)
    end if;
end for;
C = mont(A,1)
```

Figura 3: Algoritmo *Left-to-Right Square-and-Multiply*: Exponenciação Modular de Montgomery.

é proporcional à segurança oferecida, o que justifica a utilização do algoritmo de Montgomery nos processos de cifrações e decifrações.

Porém, a implementação em *hardware* do RSA, seja como um circuito integrado de aplicação específica (ASIC) ou mesmo em FPGA, resulta em um sistema físico. Isso significa que, mesmo criptografando dados com uma chave suficientemente longa, o circuito integrado pode vazar informações confidenciais através de alguns canais ocultos (ou laterais). As sucessivas multiplicações modulares, que podem ser tanto multiplicação (*A.B mod N*) quanto quadrado modular (*A.A mod N*), possuem uma ordem de execução que depende diretamente dos *bits* da chave privada (ou do expoente). Logo, informações de consumo de potência ou emissões eletromagnéticas vazadas durante as multiplicações ou quadrados modulares são fortemente relacionados aos *bits* da chave privada.

Portanto, apresenta-se a seguir fundamentos de criptoanálise e uma descrição básica dos principais métodos de ataque por canais laterais e como estes exploram as informações vazadas em uma exponenciação modular.

2.1.2 Criptoanálise

A criptoanálise compreende um conjunto de técnicas e procedimentos que visa obter o significado de uma informação cifrada. O esquema clássico utilizado por um criptoanalista é o modelo de *caixa preta*. O criptoanalista, portanto, tem acesso ao texto cifrado, ou texto claro e texto cifrado correspondente. Isso significa que este adversário não possui nenhuma informação sobre resultados intermediários decorrentes dos cálculos de um algoritmo criptográfico.

Pode-se assumir que diferentes pressupostos sejam dados à um criptoanalista:

- Texto cifrado conhecido, apenas;
- Texto claro conhecido, ou seja, o adversário possui um conjunto de textos cifrados para o qual ele conhece os textos claros correspondentes;
- Texto claro escolhido (ou texto cifrado escolhido), nesse caso o adversário pode obter os textos cifrados (ou claros) correspondentes à escolha do texto claro (ou cifrado).
- Escolha adaptativa do texto claro, ou seja, o adversário pode escolher textos claros subsequentes baseando-se na informação obtida de cifrações prévias;
- Ataque com chaves relacionadas, ou seja, o adversário não conhece as chaves, porém conhece a diferença entre elas.

O ataque por força bruta insere-se, também, no contexto de criptoanálise. Nesse caso, obtém-se a chave através de uma pesquisa exaustiva para todas as possibilidades da chave.

Nos últimos anos, a criptoanálise tem evoluído de acordo com a evolução das primitivas e protocolos matemáticos criptográficos que têm se tornado mais complexos e resistentes. No mundo real, estes métodos criptográficos têm sido aplicados em diversos sistemas eletrônicos de comunicação de dados. Logo, ataques baseados na implementação de tais sistemas tornaram-se mais robustos, isto é, apresentam maior poder computacional e matemático para quebrar um sistema criptográficos, o que torna a segurança de sistemas em *hardware* uma grande preocupação.

2.1.2.1 Segurança em *Hardware*

Atualmente, a segurança necessária em comunicações de dados são supridas por certas categorias de dispositivos criptográficos. Certamente, o exemplo mais conhecido deste tipo de dispositivo são os *smart-cards*, os quais consistem de um circuito integrado posicionado em um cartão (cartão de banco, de identificação, etc.) que tem por objetivos processar dados do usuário de modo seguro. Alguns *smart-cards* consistem de componentes de armazenamento, isto é, memórias não-voláteis. Outros, são compostos por microprocessadores e memória. Dispositivos de acesso à internet, para *Pay-TV* e *smartphones*, também empregam processos criptográficos dentro de uma rede de dados.

Todos estes dispositivos armazenam informações confidenciais do usuário. Sendo compostos por cripto-processadores, normalmente armazenam dados referentes à chave criptográfica do método implementado, ou mesmo, a própria chave. Assim, tais sistemas estão sujeitos a

ataques realizados por adversários munidos com certas informações referentes ao módulo de criptografia. Tais ataques podem ser:

1. Ataques Invasivos: são ataques em que se necessita o desencapsulamento do circuito integrado sob análise, ou seja, o dispositivo tende a ser inutilizado após o ataque. Técnicas de engenharia reversa podem ser vistas como principal exemplo de ataques invasivos, sendo o objetivo recuperar o *layout* do circuito através de processos químicos e microscópicos digitais de alta resolução. Apesar de poderosa, esta técnica apresenta diversas desvantagens, tais como alto custo de implementação, necessidade de engenheiros altamente qualificados e equipamentos que acompanhem a evolução dos processos de fabricação de circuitos integrados.

2. Ataques Semi-Invasivos: assim como os ataques invasivos, este tipo de ataque também requer o desencapsulamento do circuito integrado sob análise, porém não necessariamente inutiliza o sistema. Técnicas que utilizam *laser* para ataques que induzem falhas no dispositivo são os principais exemplos. Outro tipo de ataque semi-invasivo utiliza uma sonda de modo a espiar a informação em um condutor do circuito integrado. Apesar de eficientes, o custo para tais ataques pode ainda ser considerado elevado.

3. Ataques Não-Invasivos: são técnicas muito poderosas que visam obter informações do dispositivo criptográfico sem qualquer contato físico com o dispositivo. Tecnicamente, dividem-se em ataques que monitoram informações físicas (tempo de execução, consumo de potência, emissões eletromagnéticas) vazadas pelo dispositivo (*Side-Channel Ataques (SCA)*) ou outros que monitoram o cálculo sendo executado (*Fault Attacks (TA)*). Além das vantagens técnicas destes ataques, o custo para os ataques não-invasivos pode ser considerado relativamente baixo.

A seguir, descreve-se os principais ataques por canais laterais sobre implementações do algoritmo RSA.

2.2 Ataques por Canais Laterais

Ataques baseados em canais laterais foram originalmente introduzidos por Paul Kocher em 1996 (Kocher,1996). Em sua publicação, este autor descreve como as informações de tempo de execução de um algoritmo de criptografia podem ser exploradas como canais de fuga de informações confidenciais. Em (Kocher,1999), ele também introduz os primeiros ataques baseados no consumo de potência de um dispositivo criptográfico. Em 2001, os primeiros resultados

de análises de emissões eletromagnéticas de dispositivos criptográficos foram relatados (Gandolfi,2001)(Quisquater,2001).

Desde então, os ataques por canais laterais foram categorizados de acordo com sua classe de análise matemáticas. Ataques SPA e SEMA são ataques que exploram o comportamento assimétrico apresentado em uma única curva durante a manipulação dos dados. Os ataques por análise diferencial (DPA (Kocher,1999) e DEMA (Gandolfi,2001)(Quisquater,2001)) baseiam-se na aquisição de uma quantidade significativa de curvas e por hipóteses e classificações das curvas revela-se os *bits* da chave criptográfica. Já os ataques por análise de correlação (CPA (Brier,2004) e CEMA) possuem certa semelhança com ataques por análise diferencial, porém baseando-se na correlação do consumo de potência ou emissão eletromagnética com os dados sendo processados. A evolução dos ataques é provocada pela evolução das contra-medidas a nível de algoritmo ou de arquitetura. Nesta seção apresenta-se uma descrição destas categorias de ataques por canais laterais sobre implementações do algoritmo RSA.

2.2.1 Análise Simples (SPA - *Simple Power Analysis*, SEMA - *Simple Electromagnetic Analysis*)

Os ataques por análise simples (SPA, SEMA) focam na relação entre o consumo de potência ou a emissão eletromagnética e os dados sendo manipulados pelo dispositivo criptográfico, examinando-se uma única curva. Nenhum cálculo estatístico é empregado nesta análise. Tendo o algoritmo RSA como alvo de ataque, o adversário é capaz de distinguir os *bits* da chave privada apenas observando o comportamento assimétrico do algoritmo de exponenciação modular em uma única curva.

Algoritmos *left-to-right* e *right-to-left square-and-multiply* são os principais exemplos no que tange este comportamento assimétrico, que é caracterizado pelo cálculo (expoente = 1) ou não (expoente = 0) de uma multiplicação em paralelo (*right-to-left*) ou em série (*left-to-right*) com o quadrado modular. Como consequência, tais implementações são vulneráveis à ataques SPA e SEMA. As Figuras 4(a) e 4(b) apresentam exemplos de curvas de consumo de potência e emissão eletromagnética, respectivamente, submetidas à análise simples. Conforme ilustram essas figuras, os *bits* do expoente são facilmente revelados. Ambas as curvas foram adquiridas de processos de decifração de uma arquitetura RSA padrão (Opencores,2010).

De modo a suprimir este comportamento assimétrico da exponenciação modular, diversas soluções têm sido propostas, como os algoritmos *Montgomery Powering Ladder* (Joye,2003), *Square-and-Multiply Always* (Coron,1999), *Squaring-and-Multiply Exponentiation* (Joye,2007) e a contra-medida BRIP (Mamiya,2004). Nestes casos, as sequências de exe-

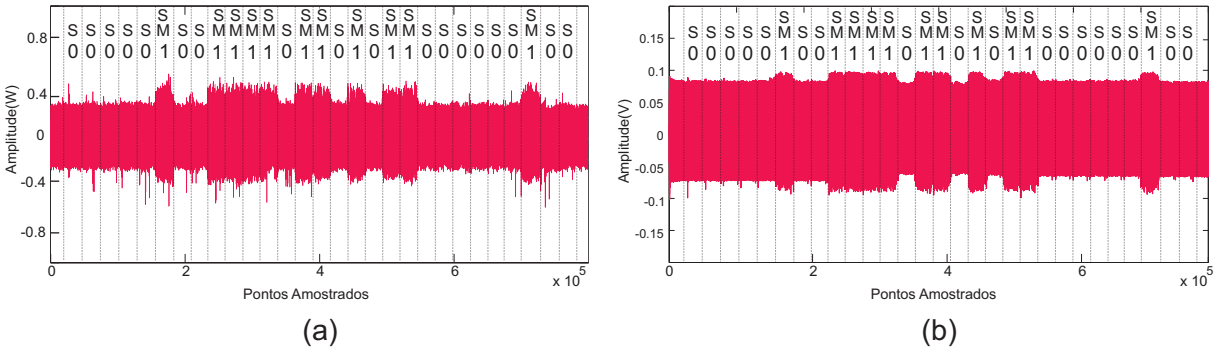


Figura 4: Ataques (a) SPA e (b) SEMA.

cuções de multiplicações e quadrados modulares não dependem dos *bits* da chave privada.

Sendo estas soluções robustas contra análise simples, novos ataques foram propostos. A seguir, descreve-se ataques mais potentes, baseados em análise diferencial e por correlação.

2.2.2 Análise Diferencial (DPA - *Differential Power Analysis*, DEMA - *Differential Electromagnetic Analysis*)

Como um avanço dos ataques SPA e SEMA, em (Kocher,1999) e em (Gandolfi,2001) são propostos ataques por análise diferencial do consumo de potência (DPA - *Differential Power Analysis*) e por emissão eletromagnética (DEMA - *Differential Electromagnetic Analysis*), respectivamente. Ao contrário da análise simples, os ataques DPA e DEMA exigem a aquisição e processamento de muito mais curvas. Outra diferença importante entre ataques simples e por análise diferencial é que as curvas adquiridas são analisadas de modo diferente: em ataques SPA e SEMA, a curva em questão é analisada, principalmente, ao longo do eixo de tempo. O adversário tenta, então, localizar padrões em uma única curva. Para ataques DPA e DEMA, o formato das curvas ao longo do eixo do tempo não tem relevância, pois este ataque analisa como o consumo de potência ou a emissão eletromagnética, em certos instantes de tempo, depende dos dados sendo processados.

2.2.2.1 O Ataque DPA/DEMA padrão

Para a demonstração do ataque DPA/DEMA padrão, utiliza-se como método criptográfico o algoritmo simétrico ou de chave privada DES, conforme ilustrado na Figura 5.

O algoritmo DES é um algoritmo de criptografia que recebe como parâmetro de entrada uma palavra de 64 *bits* (texto claro ou cifrado) e uma sequência de sub-chaves de 48 *bits*. As sub-chaves são geradas à partir da chave privada de 64 *bits*. Para cada *round* do algoritmo, existe

uma função de transformação. O algoritmo fornece como resultado uma palavra de 64 *bits* que representa o texto cifrado (ou resp. texto claro). A palavra de entrada passa por uma tabela de permutação de entrada que realiza uma permutação *bit*-à-*bit*. O valor de saída da tabela de permutação inicial é inserido no primeiro dos 16 *rounds* do algoritmo. Em cada *round*, o valor de entrada (64 *bits*) é dividido em metades esquerda (L_i) e direita (R_i), isto é, cada qual com 32 *bits*, sendo i o índice do *round*. A parte direita (R_i) e a sub-chave correspondente à cada *round* são valores de entrada para a função de *Feistel*, representada na Figura 5 por uma caixa com um "F"e ilustrada no canto superior direito da figura. Na função de *Feistel* (ou função-*f*), o termo R_i passa por uma tabela de expansão, passando de 32 para 48 *bits*. Em seguida, é feita uma operação lógica XOR entre este resultado e a sub-chave. O resultado desta operação XOR é repassado em blocos de 6 *bits* como entrada para 8 S-boxes (do inglês *substitution boxes*), que fornecem, cada uma, 4 *bits* como resultado. Os 32 *bits* de saída das 8 S-boxes passam por uma tabela de permutação *bit*-à-*bit*. Uma operação lógica XOR é feita entre o resultado de saída da função de *Feistel* e a metade esquerda L_i proveniente da tabela de permutação inicial (no caso do algoritmo estar no primeiro *round*) ou da metade esquerda L_i (no caso de o algoritmo algum *rounds* diferente do primeiro), conforme mostrado nas equações abaixo:

$$L_i = R_{i-1}R_i = L_{i-1} \oplus f(R_{i-1}, k_i) \tag{2.6}$$

Assim, resultado desta última operação XOR será a nova metade direita do próximo *round* do algoritmo e a nova metade esquerda será a metade direita do *round* anterior. Ao final dos 16 *rounds* as metades esquerda e direita resultantes passam por uma tabela de permutação final *bit*-à-*bit*. O resultado do algoritmo DES provem desta última tabela.

Nesse caso, o ataque diferencial é assumido como sendo um ataque em que o texto claro é conhecido ou o texto cifrado é conhecido. Um adversário cifra (ou decifra) N texto claros de entrada P (ou N textos cifrados C) com uma chave desconhecida armazenada no dispositivo e monitora o consumo de potência ou a atividade eletromagnética durante a cifração (respectivamente, decifração). A maneira como estas curvas são processadas estatisticamente depende da hipótese feita para alguns *bits* da sub-chave utilizada no primeiro *round* do algoritmo. Esta sub-chave apresenta 48 *bits* e a hipótese pode ser feita sobre quaisquer 6 *bits* dessa sub-chave. Existem, portanto, $2^6 = 64$ valores possíveis. Neste caso, o texto claro é o parâmetro conhecido. O adversário faz, então, todas as hipóteses para os 6 *bits* desta sub-chave e calcula a saída dos 4 *bits* da *S-box 1* (ou da S-box correspondentemente escolhida). Para realizar a análise diferencial, o adversário deve selecionar as curvas de consumo de potência ou emissão eletromagnética em grupos A e B. Caso o *bit* menos significativo da saída da *S-box 1* seja 0, a curva pertence ao

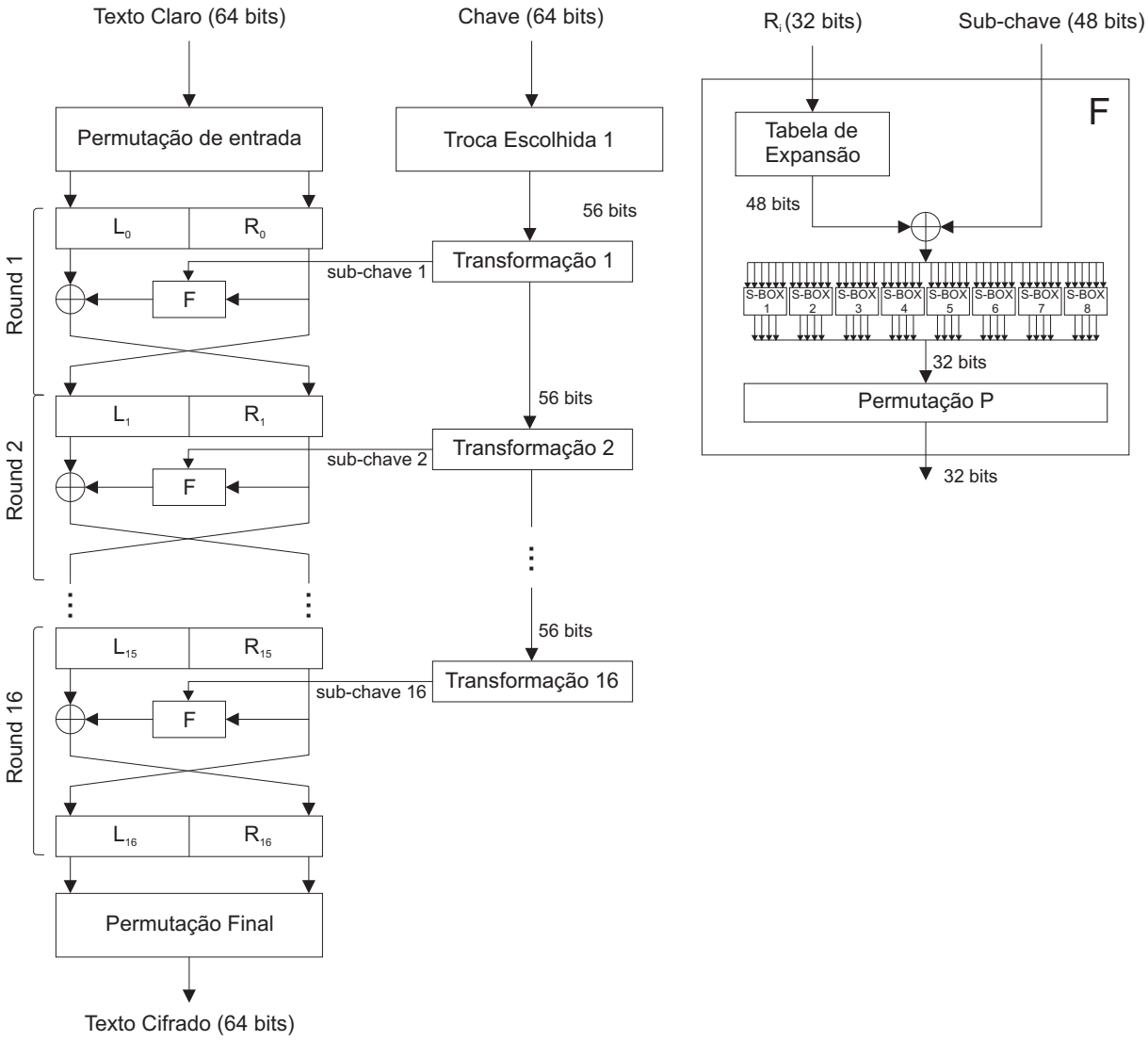


Figura 5: Algoritmo simétrico ou de chave privada DES.

grupo A. Caso contrário, se o *bit* menos significativo for 1, a curva é classificada como sendo do grupo B. O adversário calcula, então, 64 curvas diferenciais baseando-se nas 64 hipóteses para os 6 *bits* da sub-chave. A curva diferencial, para uma hipótese K_s de uma parte da sub-chave, é dada por:

$$\Delta_{K_s}[j] = \frac{\sum_{i=1}^{N} D(P_i, K_s) T_i[j]}{\sum_{i=1}^{N} D(P_i, K_s)} - \frac{\sum_{i=1}^{N} 1 - D(P_i, K_s) T_i[j]}{\sum_{i=1}^{N} 1 - D(P_i, K_s)} \tag{2.7}$$

onde $\Delta_{K_s}[j]$ é a curva diferencial contendo j amostras, N é o número de curvas utilizadas, P_i é o texto claro considerado e $T_i[j]$ é a curva de j amostras. $D(P_i, K_s)$ é a função de seleção, utilizada para classificar as curvas nos grupos A e B. Essa função de seleção D apresenta valores 0 ou 1 (de acordo com o resultado do primeiro *bit* de saída da S-Box 1), associados aos grupos A e B respectivamente. Assim, quando a função de seleção para o texto claro P_i e a hipótese da

sub-chave K_s é 0, a i-ésimo curva T na equação 2.7 é do grupo A. Caso contrário, a curva é do grupo B. Portanto, esta equação realiza uma diferença de médias entre as curvas dos grupos A e B. A Figura 6 ilustra o resultado de uma análise DPA sobre o algoritmo DES.

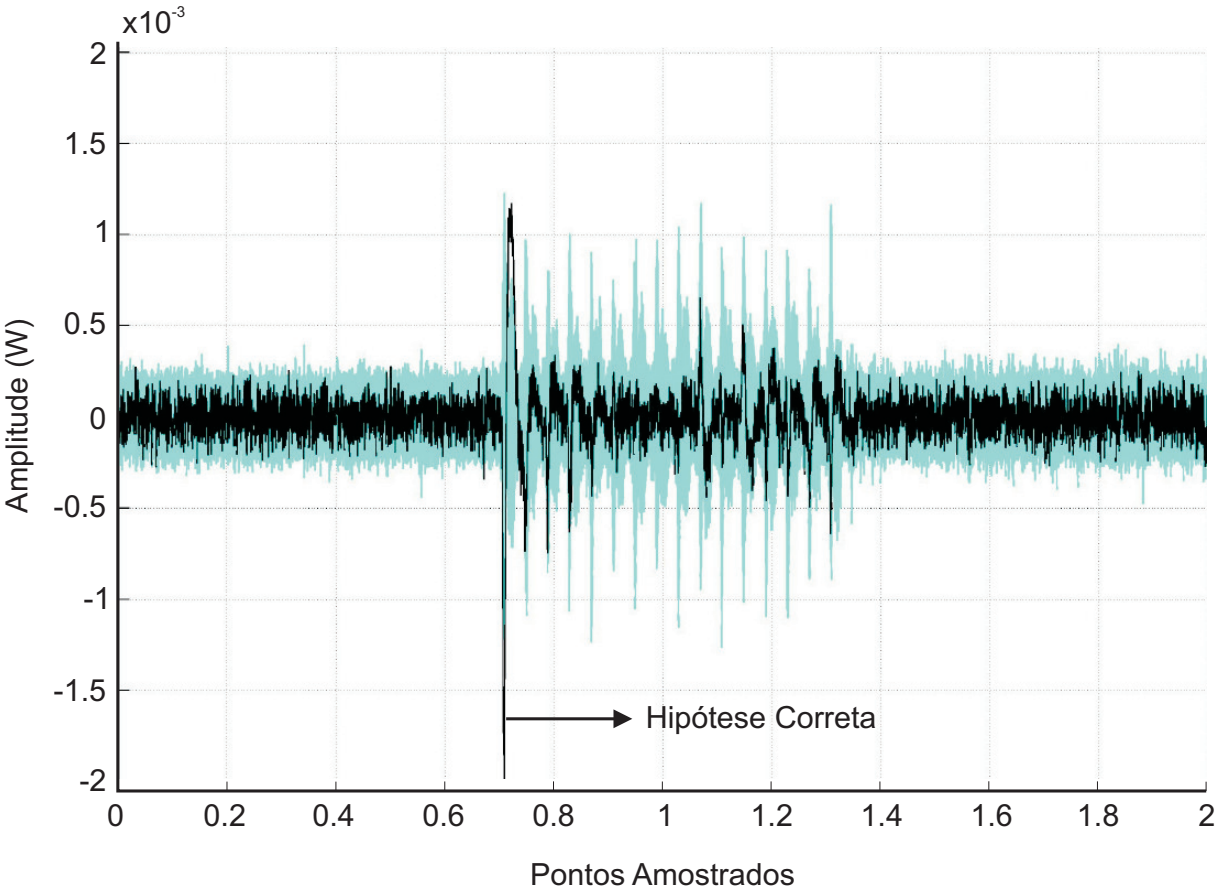


Figura 6: Ataque DPA sobre o algoritmo DES.

Se a hipótese para os *bits* da sub-chave estiver correta, a curva resultante da análise diferencial $\Delta_{K_s}[j]$ apresenta um pico de amplitude mais elevado que as demais curvas.

2.2.2.2 Ataque DPA/DEMA sobre o RSA

Um ataque DPA/DEMA sobre o algoritmo RSA busca revelar os *bits* da chave privada d. Assim, considerando-se essa chave privada como sendo $d = (d_{k-1}, d_{k-2}, ..., d_1, d_0)$, seleciona-se uma porção desta chave a ser inicialmente revelada. Caso esta porção seja de 8 *bits*, tem-se 2^8 possibilidades. Para cada possibilidade, ou hipótese d_H para os 8 *bits* da chave privada, o ataque DPA/DEMA sobre o RSA é subdividido nos seguintes passos:

1. Conhecendo-se o algoritmo de exponenciação modular empregado, adquire-se N curvas de consumo de potência (DPA) ou emissão eletromagnética (DEMA). Cada curva é o monitoramento do processo de decifração do RSA, pois o objetivo é extrair a chave privada.

2. Por hipótese, divide-se estas curvas em dois conjuntos (A e B), de acordo com um critério de seleção estatístico. Este critério pode ser o *Hamming Weight* do resultado da multiplicação modular obtido com o processamento dos 8 primeiros *bits* da chave privada, ou seja, a hipótese. Caso o adversário possua maiores conhecimentos sobre a implementação do RSA, o critério de seleção pode ser o *Hamming Weight* de um registrador que armazene um resultado que dependa do *bit* da chave privada sendo interpretado. Para auxiliar na seleção, a mensagem de entrada (texto cifrado) é conhecida pelo adversário.

3. Para cada conjunto de curvas (A e B), realiza-se uma média:

$$\begin{aligned} A_{av}(t) &= \sum_{i=1}^{N} \frac{TA_i(t)}{N} \\ B_{av}(t) &= \sum_{i=1}^{N} \frac{TB_i(t)}{N} \end{aligned} \tag{2.8}$$

   em que TA_i e TB_i são as curvas classificados nos grupos A e B, respectivamente.

4. A curva diferencial $D_H(t)$, para a hipótese d_H em questão, é obtido pela diferença entre as médias das curvas:

$$D_H(t) = A_{av}(t) - B_{av}(t) \tag{2.9}$$

Se a hipótese d_H da chave estiver correta, significa que a separação das N curvas em conjuntos A e B, de acordo com o critério de seleção, foi feita de maneira correta e picos significativos de amplitude devem aparecer na curva $D_H(t)$ resultante. Caso contrário, se a hipótese d_H difere dos *bits* da chave verdadeira, os picos de maiores amplitude não aparecem na curva resultante. A Figura 7 ilustra o resultado de uma análise DPA sobre o algoritmo RSA (Messerges,1999).

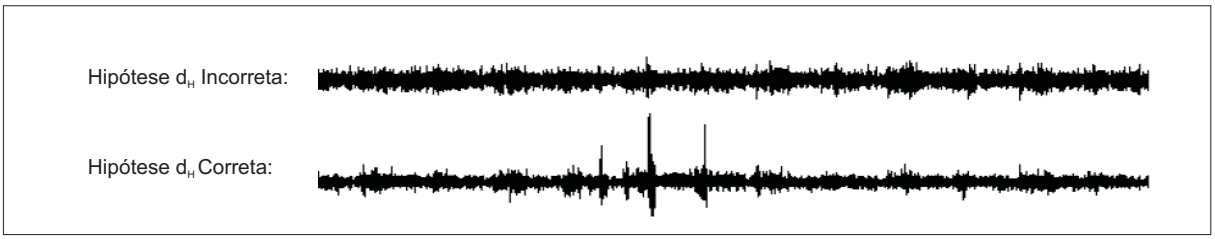


Figura 7: Ataque DPA sobre o algoritmo RSA.

2.2.3 Análise por Correlação (CPA - *Correlation Power Analysis*, CEMA - *Correlation Electromagnetic Analysis*)

A análise por correlação baseia-se no pressuposto de que o vazamento de informações através do consumo de potência ou emissões eletromagnética depende do número de *bits* chaveados de um estado para outro, levando-se em conta um período de tempo predeterminado. Normalmente, arquiteturas de microprocessadores para criptografia são modeladas como uma máquina de estados onde as transições de um estado para outro são impulsionadas por certos eventos, tal como a subida de um sinal de *clock*. Isto torna-se bastante relevante do ponto de vista de lógica elementar em tecnologias CMOS, onde a corrente consumida relaciona-se com energia necessária para alterar o valor de certos *bits* de um estado para outro. Essa energia, por sua vez, relaciona-se com as cargas de capacitâncias e correntes de curto-circuito induzidas pelas transições no *gate* dos transistores. Curiosamente, este comportamento elementar dos transistores é comumente aceito, porém não há um modelo satisfatório amplamente aplicado em análise por correlação do consumo de potência.

Se este modelo de transição é adotado, tem-se uma uma questão básica: qual é o estado de referência para o qual os *bits* são chaveados? Assume-se, então, que este estado de referência é uma palavra de *m bits* da máquina, a qual é desconhecida, mas não necessariamente composta de zeros. Esta palavra será sempre a mesma se a mesma manipulação de dados ocorrer ao mesmo tempo, embora assuma-se a ausência de qualquer efeito de dessincronização. Além disso, assume-se que o chaveamento de um *bit* de 0 para 1 ou de 1 para 0 necessite a mesma quantidade de energia e que todos os *bits* manipulados pela máquina de estados a certo instante sejam perfeitamente balanceados. O número de *bits* chaveados na mudança de um estado R para um estado D é denotado por $H(D \oplus R)$, também chamado de distância de *Hamming* entre D e R.

O cálculo da correlação é feito através da aquisição de N curvas de consumo de potência ou emissão eletromagnética W_i, e da previsão de N modelos de distâncias de *Hamming* $H_{i,R} = H(M_i \oplus R)$, sendo M_i dados aleatórios previstos e R o estado de referência. Um fator de correlação $\rho_{WH}(R)$ é dado da seguinte forma:

$$\rho_{WH}(R) = \frac{N\sum W_i H_{i,R} - \sum W_i \sum H_{i,R}}{\sqrt{N\sum W_i^2 - (\sum W_i^2)}\sqrt{N\sum H_{i,R}^2 - (\sum H_{i,R}^2)}} \tag{2.10}$$

O termo ρ da Equação 2.10 é também conhecido como fator de correlação de Pearson (Brier,2004). Este cálculo estatístico baseia-se, portanto, nas co-variâncias entre as curvas W_i e a distância de *Hamming* $H_{i,R} = H(M_i \oplus R)$, representada por um valor inteiro.

2.2.3.1 Ataque CPA/CEMA sobre o RSA

Um modelo de ataque com análise por correlação sobre o algoritmo RSA é apresentado em (Amiel,2007). Quando calcula-se a exponenciação, supõe-se valores (0 ou 1) para um ou mais *bits* do expoente secreto d, levando-se em conta uma mensagem de entrada m_j. Correlaciona-se, então, a curva de consumo de potência ou emissão eletromagnética (para um período de tempo t) com a distância de *Hamming* $H_{i,R} = H(M_i \oplus R)$, sendo R um estado de referência e M_i um resultado intermediário previsto com base na hipótese para os *bits* de d. Este resultado intermediário pode ser uma palavra do resultado de uma multiplicação modular obtida com o processamento dos *bits* hipotéticos da chave privada. Caso a hipótese esteja correta, a curva de correlação apresenta picos de maior amplitude. Caso contrário, estes picos não se fazem presentes. A Figura 8 apresenta exemplos de resultados apresentados em (Amiel,2007).

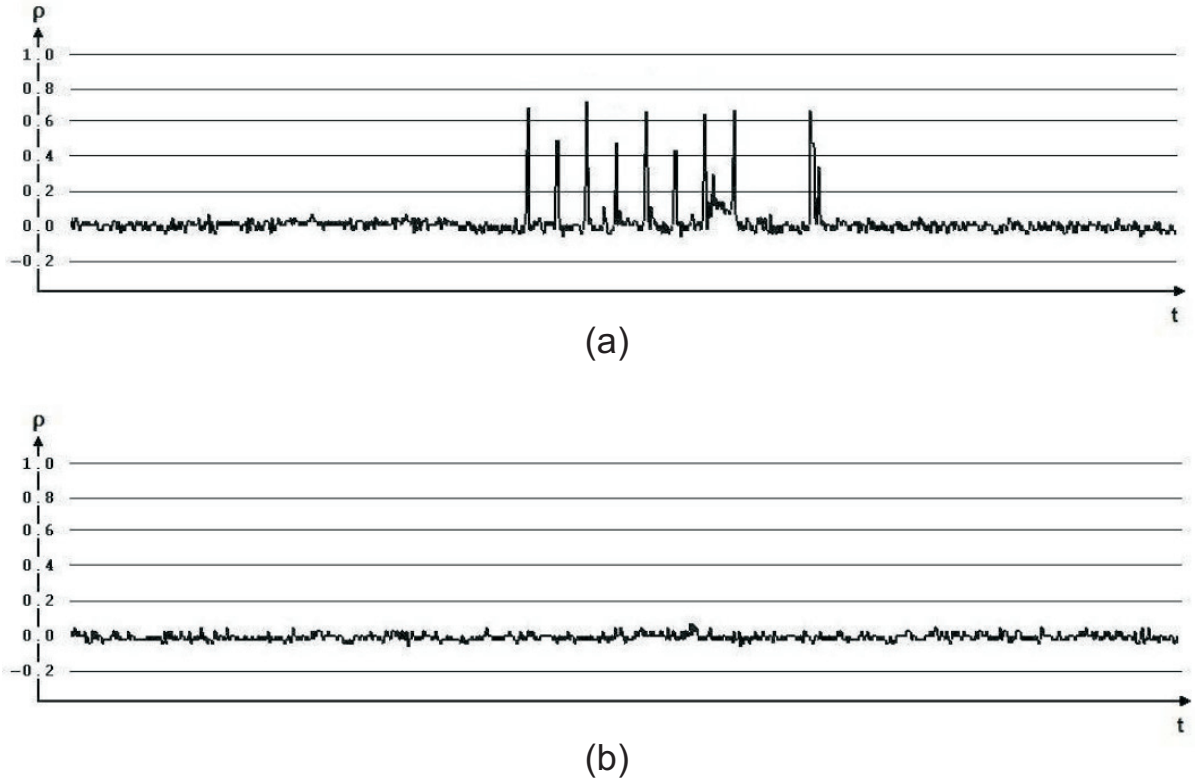


Figura 8: Ataques por Correlação: (a) hipótese correta e (b) hipótese incorreta.

Uma dificuldade dos ataques CPA ou CEMA é a escolha do estado de referência R. Normalmente, define-se que os valores de R sejam iguais à zero. Assim, o modelo de distância de *Hamming* passa a ser o *Hamming Weight* do resultado intermediário previsto. O *Hamming Weight* é o número de *bits* iguais à 1 em uma palavra. Caso contrário, quando os *bits* de R são considerados como sendo diferentes de zero, deve-se pressupor o tamanho do registrador R e definir hipóteses para todos os possíveis valores que podem ser armazenados neste registrador.

A complexidade da análise, neste caso, aumenta.

De fato, um ataque CPA/CEMA necessita menos curvas de consumo de potência ou emissão eletromagnética que no caso de um ataque DPA/DEMA.

A contra-medida mais comumente utilizada contra ataques DPA/DEMA e CPA/CEMA para o RSA consiste em modificar a mensagem de entrada m através de um valor aleatório, seja empregando-se um método aditivo $m_r = m + r_1 n \; mod \; r_2 n$, no qual r_1 e r_2 são valores aleatórios e n é o módulo, ou através de um modo multiplicativo $m_r = r_1^e m$, sendo e a chave pública. No entanto, tal contra-medida pode não ser suficiente, uma vez que o processamento das sequências de quadrado e multiplicação modular ainda dependem dos *bits* da chave privada, no caso da decifração. Assim, de modo a alterar a ordem das execuções de quadrado e multiplicações modulares, realiza-se uma randomização da chave privada através do cálculo $d_r = d + r_1 \phi(N)$, sendo $\phi(N)$ a Função Totiente de Euler e r_1 um valor aleatório de pequena magnitude.

3 TRABALHOS RELACIONADOS

Ao longo das últimas décadas, diferentes arquiteturas foram propostas para implementações do algoritmo RSA em *hardware*. Neste Capítulo, apresenta-se alguns trabalhos relacionados com os propostos nesta dissertação, focando-se em implementações do RSA baseadas no algoritmo de Montgomery para a realização da multiplicação modular.

Visando implementações mais eficientes, diversas modificações foram propostas para o algoritmo de Montgomery (Orup,1995)(Koç,1996)(Tenca,1999). Algumas destas versões apresentam certas desvantagens, tais como a fixação da precisão dos operandos. Por outro lado, o emprego de versões para a multiplicação modular com o contexto de múltipla precisão com bases numéricas grandes tem apresentado significativas vantagens e aumento de desempenho. No entanto, o emprego de bases grandes gera certas restrições ao projeto do *hardware*, uma vez que estas classes de arquiteturas são mais adequadas para implementações em *software*. A utilização de bases menores pode ser uma solução para um projeto em *hardware*, porém, como será discutido neste capítulo, critérios de segurança e velocidade tornam-se as maiores preocupações (Walter,1999).

Inicialmente, apresenta-se propostas de arquiteturas de base numérica pequena e suas características em relação ao desempenho e critérios de segurança. A seguir, arquiteturas de bases numéricas grandes são apresentadas, bem como discutem-se as restrições de implementação em *hardware*.

3.1 Arquiteturas para Multiplicação Modular

3.1.1 Arquitetura de Tenca e Koç

O trabalho de Tenca e Koç (Tenca,1999) apresenta uma arquitetura escalável para o cálculo da multiplicação modular de Montgomery. A arquitetura é dimensionada com base no número de palavras de cada operando envolvido na multiplicação modular. A arquitetura é escalável com relação à precisão dos operandos de entrada; as limitações desta arquitetura estão

na complexidade da parte de controle das execuções e na parte de memórias internas.

Esta arquitetura é baseada no algoritmo MWR2MM (*Multiple-Word Radix-2 Montgomery Algorithm*), o qual é mostrado na Figura 9.

Algorithm 2 The Multiple-Word Radix-2 Montgomery Multiplication Algorithm

Require: odd $M, n = \lfloor \log_2 M \rfloor + 1$, word size w, $e = \lceil \frac{n+1}{w} \rceil$, $X = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $Y = \sum_{j=0}^{e-1} Y^{(j)} \cdot 2^{w \cdot j}$, $M = \sum_{j=0}^{e-1} M^{(j)} \cdot 2^{w \cdot j}$, with $0 \leq X, Y < M$

Ensure: $Z = \sum_{j=0}^{e-1} S^{(j)} \cdot 2^{w \cdot j} = MP(X, Y, M) \equiv X \cdot Y \cdot 2^{-n} \pmod{M}, 0 \leq Z < 2M$

1: $S = 0$ — *initialize all words of S*
2: **for** $i = 0$ to $n - 1$ step 1 **do**
3: $q_i = (x_i \cdot Y_0^{(0)}) \oplus S_0^{(0)}$
4: $(C^{(1)}, S^{(0)}) = x_i \cdot Y^{(0)} + q_i \cdot M^{(0)} + S^{(0)}$
5: **for** $j = 1$ to $e - 1$ step 1 **do**
6: $(C^{(j+1)}, S^{(j)}) = C^{(j)} + x_i \cdot Y^{(j)} + q_i \cdot M^{(j)} + S^{(j)}$
7: $S^{(j-1)} = (S_0^{(j)}, S_{w-1..1}^{(j-1)})$
8: **end for**
9: $S^{(e-1)} = (C_0^{(e)}, S_{w-1..1}^{(e-1)})$
10: **end for**
11: return $Z = S$

Figura 9: Algoritmo MWR2MM.

De acordo com a notação original do algoritmo MWR2MM, o termo *M* representa o módula da multiplicação modular e os termos *X* e *Y* os operandos de entrada. no cálculo *XY mod N*. O termo *w* representa a base numérica do algoritmo de Montgomery e *e* indica o número de palavras de cada operando dentro do contexto de precisão múltipla. O operando *Y* é escaneado palavra-a-palavra, e o operando *X* é escaneado *bit-a-bit*. Os operandos possuem comprimento de *n bits* e cada palavra possui *w bits*. O resultado *S* é composto por $e = \frac{n+1}{w}$ palavras, sendo que sua faixa de magnitude é $[0, 2M-1]$, sendo *M* o termo que designa o modulo da operação modular. Os operandos *Y* e *M* recebem um *bit* extra com valor 0, como *bit* mais significativo. Todos estes operandos são apresentados como vetores unidimensionais $M = (M^{(e-1)}, ..., M^{(1)}, M^{(0)})$, $Y = (Y^{(e-1)}, ..., Y^{(1)}, Y^{(0)})$, $S = (S^{(e-1)}, ..., S^{(1)}, S^{(0)})$ e $X = (X^{(n-1)}, ..., X^{(1)}, X^{(0)})$, nos quais o sobrescrito nos operandos *M*, *Y* e *S* representa o índice de uma palavra de *w bits* e o sobrescrito no operando *X* representa o índice de um *bit* dentro deste operando que contem *n bits*. A variável de *carry* $C^{(j)}$ possui dois *bits* e o valor de $C^{(j)}$ nunca excede $C^{(j)} = 2$. Isto ocorre em função da linha 6 do algoritmo MWR2MM.

A arquitetura de Tenca e Koç é composta por um número fixo de elementos de processamento (PE, do inglês *processing elements*) equivalentes ao número de palavras dos operandos *X*, *Y* e *M* de entrada. A Figura 10 ilustra a arquitetura de Tenca e Koç.

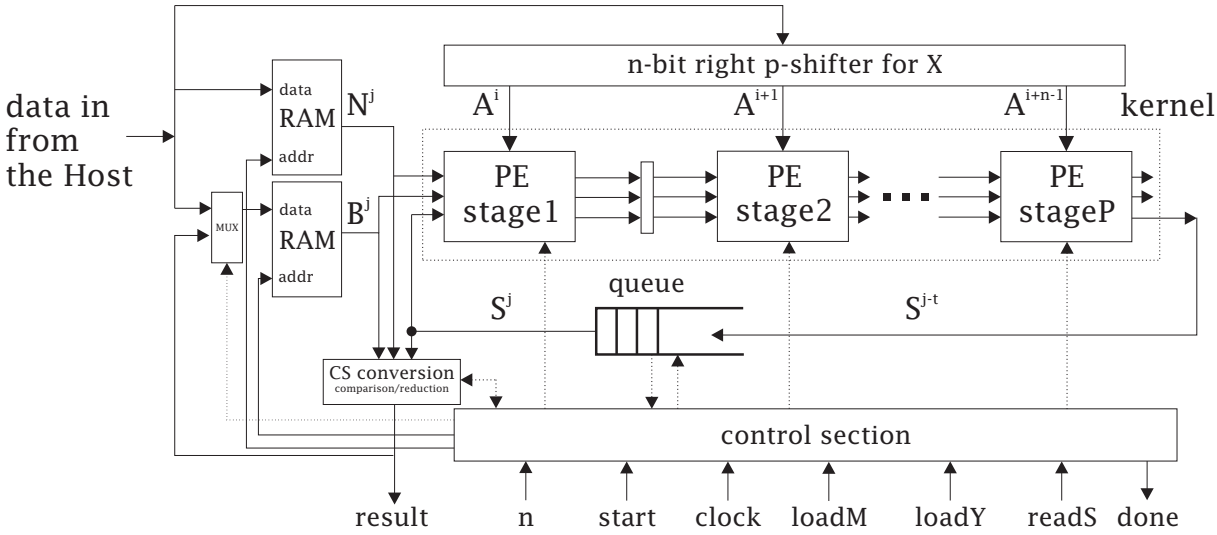


Figura 10: Arquitetura de Tenca e Koç.

Além dos elementos de processamento que compõem a estrutura da arquitetura, esta é também composta por elementos de memória, conversão de dados e unidade de controle. os resultados de síntese para esta arquitetura foram apresentados para uma tecnologia CMOS 0.5 μm. Em torno de 40k *gates* foram estimados nesse projeto e uma multiplicação modular de 1024 *bits* é realizada em 43μs.

3.1.2 Arquitetura de Huang *et al*

Após a introdução da arquitetura de Tenca e Koç, diversos projetos baseados no algoritmo MWR2MM foram apresentados de modo a reduzir o tempo necessário para o cálculo de uma multiplicação modular (Tenca,2001)(Michalski,2006)(Huang,2008).

Em resumo, o trabalho apresentado em (Tenca,2001) propõe uma versão para bases numéricas maiores em relação ao algoritmo MWR2MM, e introduz o algoritmo MWR2^{k}MM, o qual é baseado na codificação *Booth* de multiplicadores. Como vantagens, essa nova abordagem apresenta um número reduzido de passos para o cálculo da multiplicação modular em relação ao algoritmo MWR2MM, porém com maior complexidade da parte de controle e da parte aritmética.

Em (Michalski,2006), introduz-se o algoritmo MWRKMM, o qual deriva do método FIOS (*Finely Integrated Operand Scanning*), descrito em (Koç,1996). Esta arquitetura requer a construção de multiplicadores de modo a acelerar o cálculo da multiplicação modular, porém isto gera certas restrições com relação à área da implementação.

A arquitetura de Huang *et al* (Huang,2008) foca na otimização de arquiteturas de *hardware*

para o algoritmo MWR2MM e propõe uma versão de base 4 MWR4MM de modo a diminuir a latência entre os elementos de processamento. O Algoritmo MWR4MM é ilustrado na Figura 11.

Algorithm 4 The Multiple-Word Radix-4 Montgomery Multiplication Algorithm

Require: odd $M, n = \lfloor \log_2 M \rfloor + 1$, word size w, $e = \lceil \frac{n+1}{w} \rceil$, $X = \sum_{i=0}^{\lceil \frac{n}{2} \rceil - 1} x^{(i)} \cdot 4^i$, $Y = \sum_{j=0}^{e-1} Y^{(j)} \cdot 2^{w \cdot j}$, $M = \sum_{j=0}^{e-1} M^{(j)} \cdot 2^{w \cdot j}$, with $0 \leq X, Y < M$

Ensure: $Z = \sum_{j=0}^{e-1} S^{(j)} \cdot 2^{w \cdot j} = MP(X, Y, M) \equiv X \cdot Y \cdot 2^{-n} \pmod{M}, 0 \leq Z < 2M$

1: $S = 0$ — *initialize all words of S*
2: **for** $i = 0$ to $n - 1$ step 2 **do**
3: $q^{(i)} = Func(S_{1..0}^{(0)}, x^{(i)}, Y_{1..0}^{(0)}, M_{1..0}^{(0)})$ — *$q^{(i)}$ is 2-bit long*
4: $(C^{(1)}, S^{(0)}) = S^{(0)} + x^{(i)} \cdot Y^{(0)} + q^{(i)} \cdot M^{(0)}$ — *C is 3-bit long*
5: **for** $j = 1$ to $e - 1$ step 1 **do**
6: $(C^{(j+1)}, S^{(j)}) = C^{(j)} + S^{(j)} + x^{(i)} \cdot Y^{(j)} + q^{(i)} \cdot M^{(j)}$
7: $S^{(j-1)} = (S_{1..0}^{(j)}, S_{w-1..2}^{(j-1)})$
8: **end for**
9: $S^{(e-1)} = (C_{1..0}^{(e)}, S_{w-1..2}^{(e-1)})$
10: **end for**
11: return $Z = S$

Figura 11: Algoritmo MWR4MM.

O número de ciclos de *clock* necessários para calcular uma multiplicação de Montgomery, com certa precisão *n*, é menor. A Figura 12 apresenta a arquitetura proposta por Huang *et al*.

Em termos de resultados de síntese e desempenho, a arquitetura proposta por Huang apresenta 65 elementos de processamento, com ocupação de 4178 *Slices*, quando sintetizada em FPGAs Virtex-2, considerando operandos de 1024 *bits*. O tempo necessário para uma multiplicação modular de Montgomery de 1024 *bits* é 21.76μs, o que representa uma redução de 50% em relação à arquitetura de Tenca e Koç.

3.1.3 Arquitetura de Blum e Paar

O trabalho de Blum e Paar (Blum,2001) apresenta uma arquitetura sistólica voltada para otimização de desempenho e redução de elementos lógicos em FPGA.

A versão do algoritmo de Montgomery empregada nesta arquitetura foi proposta em (Orup,1995) e aparece ilustrado na Figura 13. Esta versão apresenta uma simplificação do cálculo do quociente q_i, fazendo deste uma simples operação de truncamento. O operando *B* é multiplicado pela base numérica 2^k e o módulo *N* é multiplicado pela seu inverso multiplica-

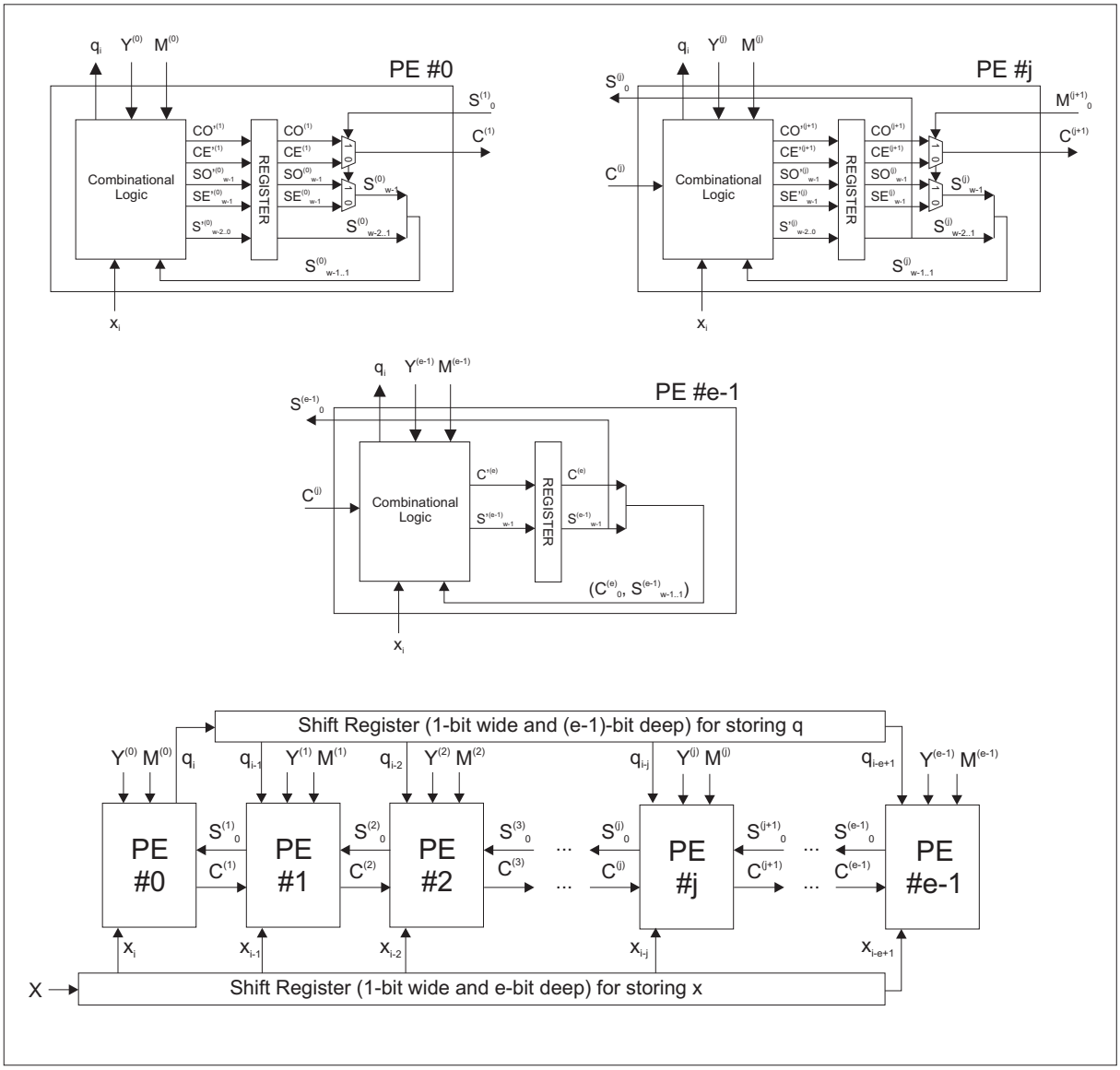


Figura 12: Arquitetura de Huang *et al*.

tivo N' modulo 2^k. Como consequência destas modificações, o algoritmo possui três iterações adicionais.

A arquitetura sistólica proposta por Blum e Paar é apresentada na Figura 14, junto com a estrutura interna dos elementos de processamento.

Esta arquitetura emprega bases numéricas maiores (2^8, 2^{16}). Como resultados, uma multiplicação modular é computada em $2m + 20$ ciclos de *clock*. A grande desvantagem desta implementação está no fato de que todas as multiplicações são pré-calculadas, ou seja, memórias RAM armazenam múltiplos dos operandos B e M. Como consequência, gasta-se muitos ciclos de *clock* antes de cada multiplicação modular para realizar todos os pré-cálculos.

Multiplicação Modular de Montgomery para o cálculo de ABR^{-1} mod N.

$\overline{M} = (M' \bmod 2^k)M$, $\overline{M} = \sum_{i=0}^{m-1}(2^k)^i\overline{m}_i$, $\overline{m}_i \in \{0, 1, ..., 2^k\text{-}1\}$,
$A = \sum_{i=0}^{m+1}(2^k)^i a_i$, $a_i \in \{0, 1, ..., 2^k\text{-}1\}$, $a_{m+2} = 0$
$B = \sum_{i=0}^{m+2}(2^k)^i b_i$, $b_i \in \{0, 1, ..., 2^k\text{-}1\}$
$R = (2^k)^m$, $A, B < 2\overline{M}$; $4\overline{M} < R = (2^k)^m$
$M' = -M^{-1} \bmod 2k$

1. $S_0 = 0$;
2. for *i = 0* **to** *m+2* **do**
3. $q_i = (S_0) \bmod 2^k$
4. $S_{i+1} = (S_i + q_i\overline{M})/2^k + a_iB$
5. end for

Figura 13: Versão do algoritmo de Montgomery proposta em (Orup,1995).

3.1.4 Arquitetura de McIvor *et al*

A arquitetura apresentada em (McIvor,2005) é composta por um arranjo de elementos de processamento que tem por objetivos realizar as operações aritméticas do algoritmo de Montgomery. Esta nova abordagem apresenta diversas vantagens em relação à proposta de Blum e Paar.

O trabalho de McIvor *et al* também baseia-se na versão do algoritmo de Montgomery apresentado em (Orup,1995). A Figura 15 ilustra a estrutura proposta, detalhando a arquitetura interna dos elementos de processamento.

A arquitetura é composta por $m+1$ elementos de processamento idênticos, com um primeiro bloco que realiza o cálculo de $S_i + q_i.N$, utilizando os *bits* menos significativos de S_i e N. O arranjo dos demais elementos de processamento é responsável pelo cálculo de $(S_i + q_i.N) + a_i.B + Carry$.

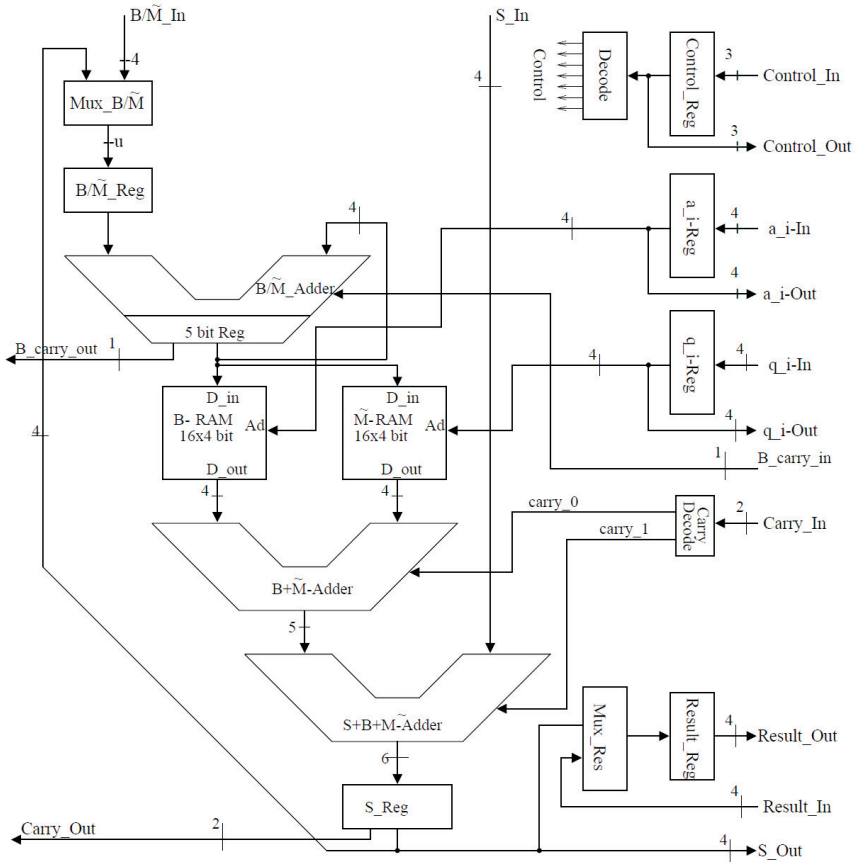
Os resultados desta arquitetura foram apresentados em termos de desempenho em uma aplicação para FPGA da família Virtex-2 (Xilinx,2007). As Tabelas 1 e 2 abaixo resumem os resultados obtidos para esta arquitetura.

Tabela 1: Resultados de desempenho para implementações de 512 bits.

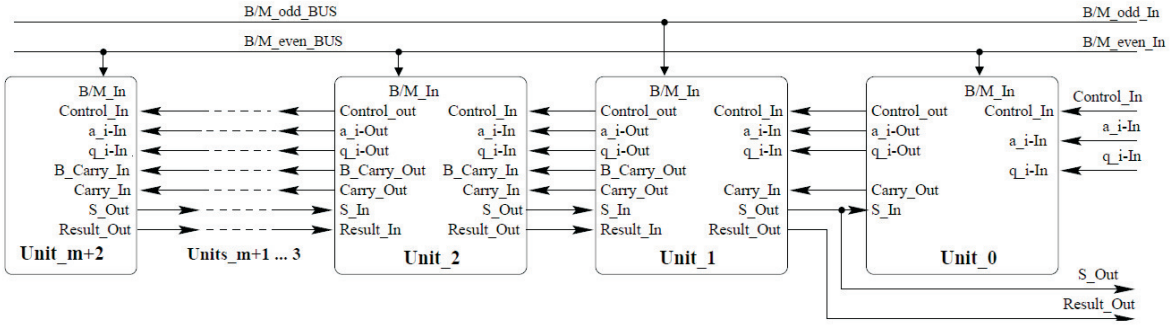
Base	Clock(MHz)	Area (Slices)	Mult 18×18	Clock Cycles	Data Rate(Mb/s)
2^8	119.86	2,952	131	199	616.77
2^{16}	108.70	2,923	67	103	1080.68

Esta arquitetura apresenta um baixo número de ciclos de *clock* para uma multiplicação

$$S_{i+1} = (S_i + q_i \cdot \tilde{M})/2^k + a_i \cdot B$$



(a)



(b)

Figura 14: Arquitetura de Blum e Paar. (a) Elementos de Processamento. (b) Arquitetura Sistólica.

Tabela 2: Resultados de desempenho para implementações de 1024 bits.

Base	Clock(MHz)	Area (Slices)	Mult 18×18	Clock Cycles	Data Rate(Mb/s)
2^8	104.70	5,797	259	391	549.40
2^{16}	101.86	5,709	131	199	1048.29

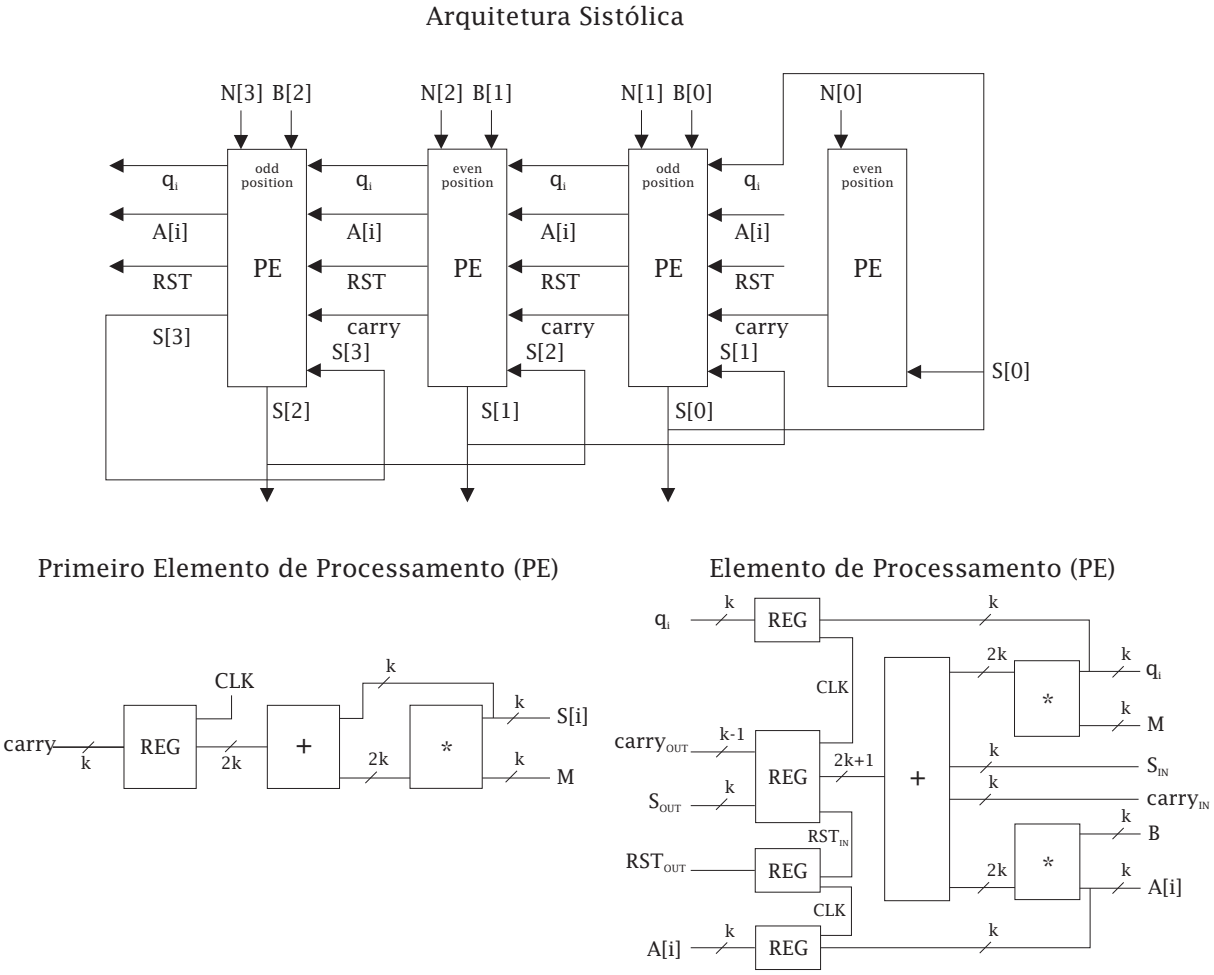


Figura 15: Arquitetura de McIvor *et al*.

modular de Montgomery, apesar da necessidade de empregar um número consideravelmente elevado de multiplicadores.

4 ARQUITETURAS PROPOSTAS E ANÁLISE DE DESEMPENHO

Este capítulo apresenta duas arquiteturas para a multiplicação modular de Montgomery. Primeiramente, uma arquitetura sistólica é descrita em detalhes, caracterizando o modo de funcionamento em matriz unidimensional desta implementação. Na sequência, apresenta-se uma nova proposta, através de uma arquitetura que calcula a multiplicação modular de Montgomery com multiplexação de processos aritméticos. Por fim, propõe-se uma estrutura para a execução da exponenciação modular a qual comporta dois diferentes métodos de exponenciação: *left-to-right square-and-multiply* e *square-and-multiply always*. Por fim, apresenta-se uma análise de desempenho para as arquiteturas propostas e uma comparação com trabalhos relacionados.

4.1 Arquitetura Sistólica

O conceito de arquiteturas sistólicas (do inglês *systolic array*) foi inicialmente introduzido em 1978 (Kung,1978) para designar uma classe especial de arquiteturas paralelas. A grande difusão e aplicação destas arquiteturas se deve ao modo como os dados fluem entre os elementos de processamento. Tipicamente, um arranjo sistólico é capaz de realizar operações simples como multiplicação de matrizes ou inversão. Atualmente, o conceito de arquitetura sistólica é conhecido e utilizado em diversas aplicações, incluindo sistemas de computação dedicados à solução de problemas numéricos particulares. No caso de aplicações em algoritmos de criptografia de chave pública, as arquiteturas sistólicas são adequadas para os cálculos aritméticos envolvendo números inteiros grandes, da ordem de 1024 *bits* ou mais.

Basicamente, um arranjo sistólico é um sistema em que os dados fluem pela memória de um computador ritmicamente, passando por vários elementos de processamento antes de retornar novamente para a memória do sistema. O *array* sistólico possui um conjunto de células interconectadas capazes de executar operações simples. Por possuírem características como simplicidade, comunicação regular e estruturas de controle, essas células possuem muitas vantagens de implementação sobre outros tipos de elementos de processamento. Em sistemas sistólicos,

essas células são organizadas em topologias tipo árvore ou "*arrays*". A comunicação com o mundo externo é realizada apenas através das células de fronteira da topologia.

A arquitetura sistólica proposta foi projetada com base nas operações aritméticas do algoritmo de Montgomery, que por sua vez são realizadas em uma base numérica 2^k e com os termos de entrada *A*, *B* e *N* divididos em um número fixo de *m* palavras de *k bits*. Como visto no Capítulo 2, o algoritmo de Montgomery apresenta suas operações aritméticas em um contexto de precisão múltipla. Por convenção, mostra-se novamente o algoritmo de Montgomery na Figura 16. Uma vez que este algoritmo é empregado na exponenciação modular de Montgomery, supre-se a subtração final.

Algoritmo 4: Multiplicação Modular de Montgomery para o cálculo de ABR^{-1} mod N.

$N = \sum_{i=0}^{m-1}(2^k)^i n_i,\ n_i \in \{0, 1, ..., 2^k-1\}$
$A = \sum_{i=0}^{m-1}(2^k)^i a_i,\ a_i \in \{0, 1, ..., 2^k-1\}$
$B = \sum_{i=0}^{m-1}(2^k)^i b_i,\ b_i \in \{0, 1, ..., 2^k-1\}$
$R = (2^k)^m,\ A, B < 2N;\ N < R = (2^k)^m$
$N = -N^{-1} \bmod 2^k$

1. $S_0 = 0$;
2. for *i = 0* **to** *m-1* **do**
3. $q_i = ((S_0 + a_i b_i)N') \bmod 2^k$
4. $S_{i+1} = (S_i + q_i N + a_i B)/2^k$
5. end for

Figura 16: Algoritmo de Montgomery.

A Figura 17 apresenta a estrutura interna da arquitetura sistólica proposta. Ela é composta por *m* elementos de processamento (PE) distribuídos ao longo de uma matriz unidimensional. O número de elementos de processamento é igual ao número de palavras de cada operando de entrada *A*, *B* e *N* da multiplicação modular. Cada PE é responsável por processar uma palavra de *k bits* dos operandos de entrada *B* e *N* de mesmo índice do PE. Desse modo, por exemplo, o primeiro elemento de processamento (PE0) realiza operações aritméticas com as palavras menos significativas de cada um destes operandos, ou seja, b_0 e n_0. O operando *A* é fornecido serialmente (palavra por palavra a_i) e, juntamente com sinais de *carry*, é propagado entre os elementos de processamento, o que configura o modo sistólico. Os sinais de *carry* referem-se aos *bits* mais significativos (MSB - *Most Significant Bits*) do resultado das multiplicações e adições.

Os elementos de processamento são implementados com máquinas de estados e são gerenciados por um bloco de controle. O bloco de controle estabelece comunicação com o

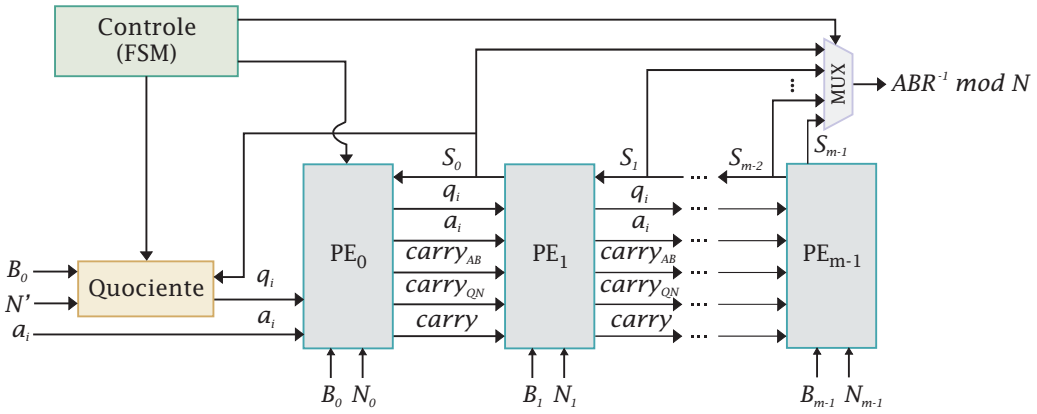


Figura 17: Arquitetura em matriz sistólica.

primeiro elemento de processamento e com o bloco responsável pelo cálculo do quociente $q_i = (S_0 + a_i.b_0)N' \, mod \, 2^k$, segundo a linha 3 do algoritmo de Montgomery. O termo q_i é fornecido ao primeiro elemento de processamento e repassado aos demais serialmente.

O arranjo unidimensional dos elementos de processamento tem por objetivo o cálculo de $S_{i+1} = (S_i + q_i N + a_i B)/2^k$, conforme a linha 4 do algoritmo de Montgomery. Nesta operação, observa-se que existem duas multiplicações entre um operando de entrada e uma palavra de *k bits*, além da adição dos resultados destas duas multiplicações. A Figura 18 apresenta o fluxograma das operações aritméticas dentro de cada elemento de processamento.

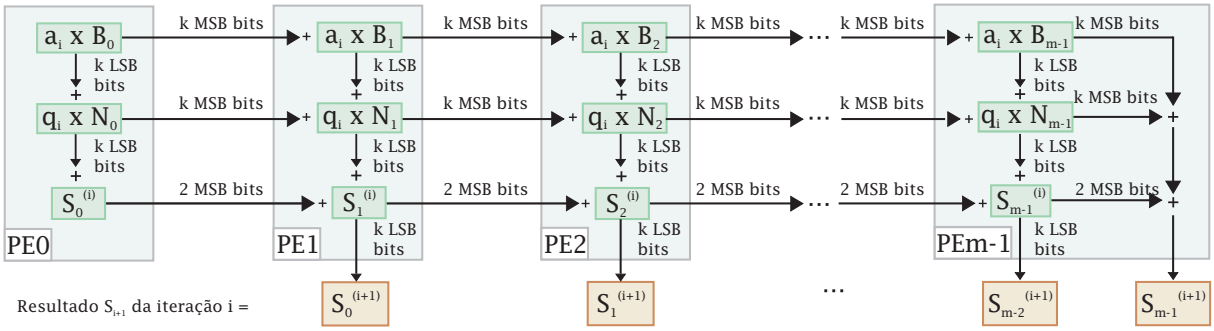


Figura 18: Operações aritméticas dentro dos PEs para o cálculo de $S_{i+1} = (S_i + q_i \times N + a_i \times B)/2^k$.

O primeiro elemento de processamento (PE0) estabelece uma comunicação com a unidade de controle da arquitetura e recebe as palavras a_i e q_i a cada iteração do algoritmo de Montgomery. Este primeiro elemento de processamento difere dos demais por que não recebe sinais de *carry* como entrada e não fornece como resultado uma palavra de S. Quando o primeiro elemento de processamento envia sinais de *carry* ao segundo elemento de processamento (PE1), este inicia suas operações aritméticas. Isso caracteriza um funcionamento em modo *pipeline* da arquitetura, pois o primeiro elemento de processamento recomeça um novo cálculo assim que termina o anterior. A arquitetura interna do PE0 é ilustrada na Figura 19.

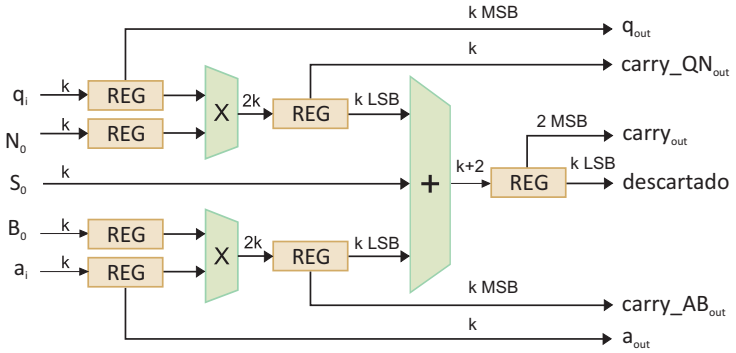


Figura 19: Arquitetura interna do primeiro Elemento de Processamento.

Os demais elementos de processamento possuem como resultado uma palavra de S_{i+1}, além dos sinais de *carry* das multiplicações e das adições. Apenas o último elemento de processamento (PE_{m-1}) fornece duas palavras do resultado S_{i+1} como resposta, ou seja, S_{m-2} e S_{m-1}. A Figura 20 apresenta a arquitetura interna dos elementos de processamento.

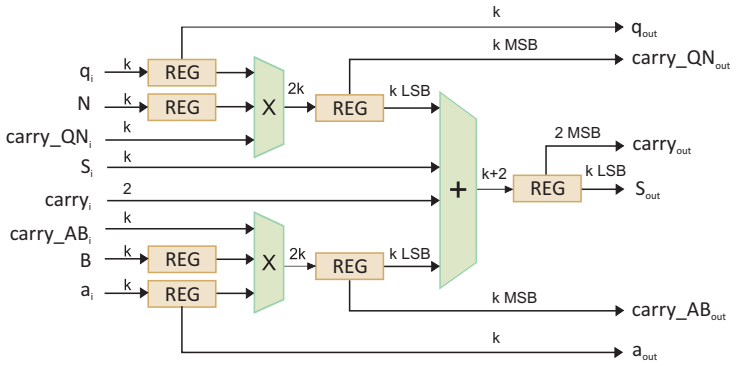


Figura 20: Arquitetura interna dos Elementos de Processamento.

A Figura 21 mostra a arquitetura interna do bloco quociente (esquerda) e o fluxo de operações (direita). O cálculo do quociente q_i se dá de acordo com a linha 3 do algoritmo de Montgomery da Figura 16. As palavras de k *bits* s_0, a_i, b_0 e N' são fornecidas para este bloco a cada iteração i.

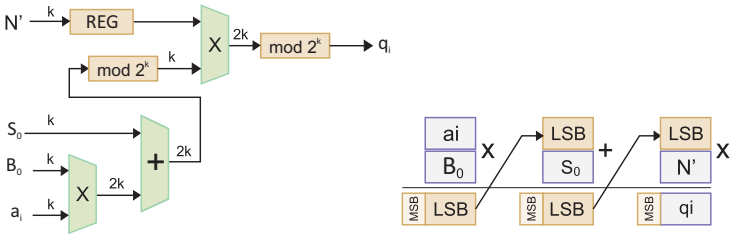


Figura 21: Arquitetura interna do bloco quociente.

A arquitetura sistólica é capaz de realizar multiplicações modulares para qualquer precisão

dos operandos de entrada. Tendo sua estrutura baseada em aritmética de múltipla precisão, o número de elementos de processamento varia com o número de palavras dos operandos de entrada.

4.1.1 Estimativa de Latência

O número de ciclos de *clock* necessários entre as iterações do algoritmo de Montgomery pode ser visualizado através do fluxo de operações atômicas da Figura 22. Nela, são ilustrados como ocorre a propagação de resultados e palavras de entrada em cada elemento de processamento, bem como o número de ciclos de *clock* empregados para os cálculos em cada elemento de processamento.

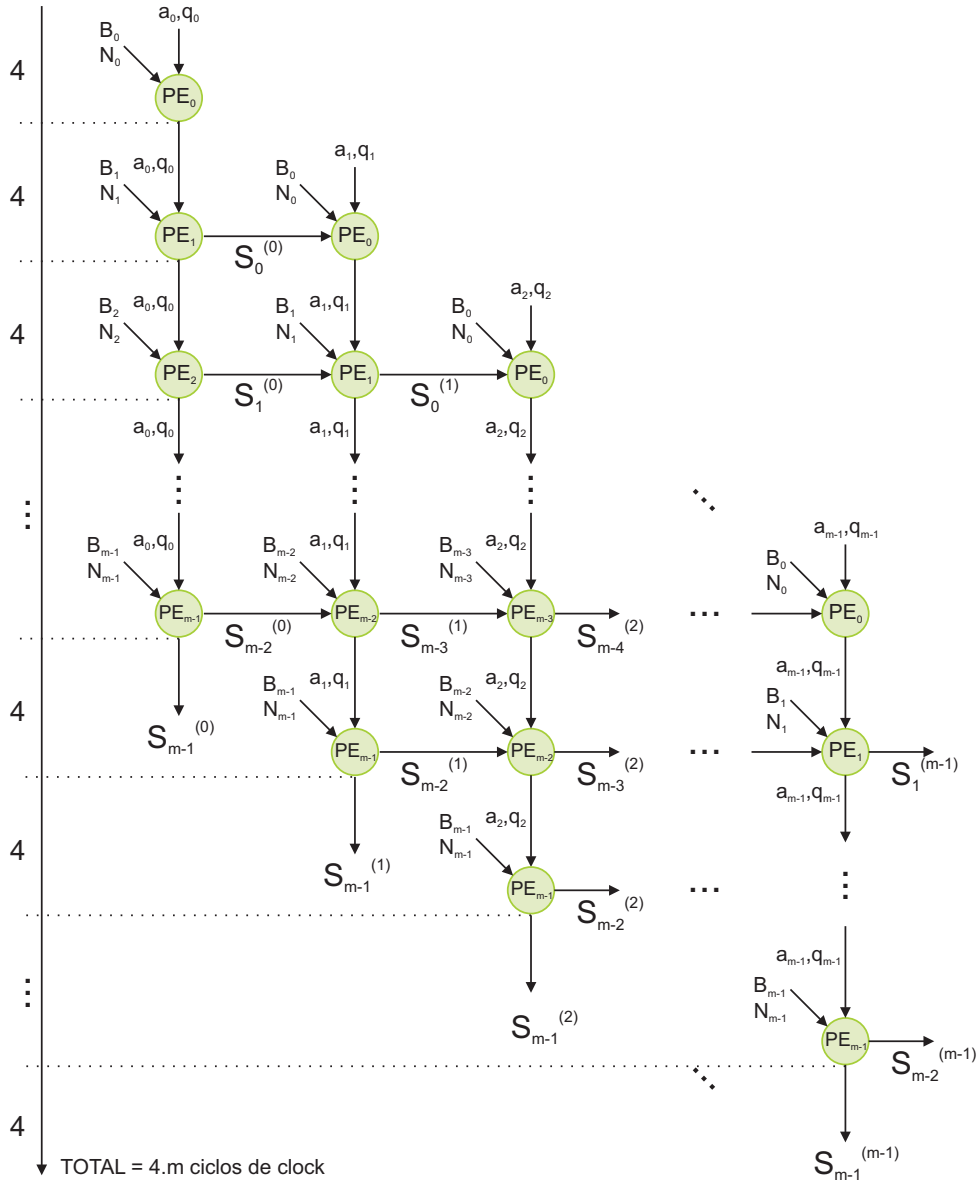


Figura 22: Estimativa de Latência.

Portanto, as m iterações do algoritmo de Montgomery são realizadas em exatamente $4.m$ ciclos de *clock*. Antes da multiplicação modular, os operandos B e N são lidos de memórias RAM (conforme será apresentado na estrutura para exponenciação modular), o que leva m ciclos de *clock*. Assim, número total de ciclos de *clock* por multiplicação modular é $5.m$. Esse tempo deve ser levado em conta quando a arquitetura sistólica é empregada em um processo de exponenciação modular, como será mostrado no final deste capítulo.

4.2 Arquitetura Multiplexada

Como visto na seção anterior, a arquitetura em matriz sistólica apresenta um arranjo de elementos de processamento dispostos unidimensionalmente, sendo que cada elemento de processamento é responsável por operações de adição e multiplicação entre palavras de k *bits*. Quando a base numérica (2^k) é relativamente elevada ($2^{16}, 2^{32}$), estas operações de multiplicação e adição aumentam a complexidade e limitam o desempenho do sistema, principalmente no caso de uma aplicação em *hardware*. Logo, quanto maior o número de multiplicadores dentro da arquitetura, maiores serão às limitações físicas, como por exemplo, frequência máxima de *clock* e área.

Baseando-se nesse fato, apresenta-se nesta seção uma arquitetura em matriz sistólica com multiplicadores multiplexados trabalhando em paralelo com os elementos de processamento. Isso propicia uma migração de multiplicadores $k \times k$ *bits* dos elementos de processamento para dentro de novos blocos multiplicadores. Cada bloco multiplicador, juntamente com quatro elementos de processamento, formam um *core* aritmético. O arranjo unidimensional dos *cores* aritméticos formam a estrutura da arquitetura multiplexada para multiplicação modular. As Figuras 23 e 24 apresentam a arquitetura multiplexada e a estrutura interna dos *cores* aritméticos, respectivamente.

A arquitetura multiplexada é composta por $n/4k$ *core* aritméticos, sendo que o primeiro (AC1) é gerenciado por um bloco de controle posicionado dentro da arquitetura multiplexada. Este bloco de controle é implementado com uma máquina de estados finita. De acordo com a Figura 24, cada *core* aritmético possui quatro elementos de processamento, um multiplicador e uma memória RAM de $8 \times k$ *bits* de capacidade de armazenamento. Sendo uma arquitetura baseada em aritmética de precisão múltipla, o número de elementos de processamento em toda a arquitetura é equivalente ao número de palavras de cada operando de entrada. Assim, a memória RAM alocada em cada *core* aritmético armazena quatro palavras de cada um dos operandos B e N.

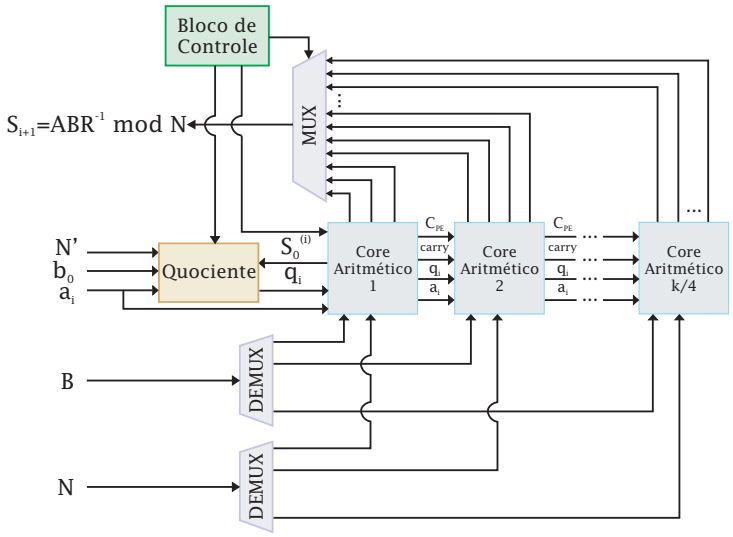


Figura 23: Arquitetura Multiplexada para Multiplicação Modular.

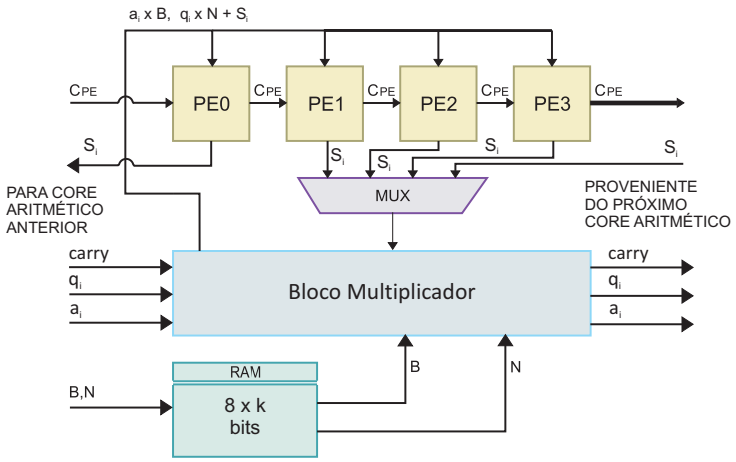


Figura 24: Arquitetura Interna dos *Cores* Aritméticos.

O bloco multiplicador realiza as multiplicações $q_i \times N_i$ e $a_i \times B_i$. Os *bits* menos significativos do resultado de $q_i \times N_i$ são adicionados à uma palavra de k *bits* proveniente do resultado da iteração anterior S_i, conforme o algoritmo de Montgomery. Os *bits* menos significativos desta última adição e os *bits* menos significativos da multiplicação $a_i \times B_i$ são enviados aos elementos de processamento para serem adicionados entre si.

Os elementos de processamento fornecem, então, as palavras do resultado S_{i+1} referente à iteração atual. Como exemplo, a Figura 25 apresenta a sequência de execuções realizadas pelo primeiro *core* aritmético (AC1). Analisando-se esta figura, verifica-se como ocorre a multiplexação dos processos aritméticos. O bloco multiplicador realiza todas multiplicações de precisão simples que antes eram executadas em cada elemento de processamento (no caso da arquitetura sistólica). Em outras palavras, o número de multiplicadores $k \times k$ *bits* é reduzido em

quatro vezes. Assim, em virtude desta multiplexação, cada elemento de processamento executa apenas uma adição. Definiu-se a multiplexação para quatro elementos de processamento por ser o melhor resultado atingido em termos de desempenho e área.

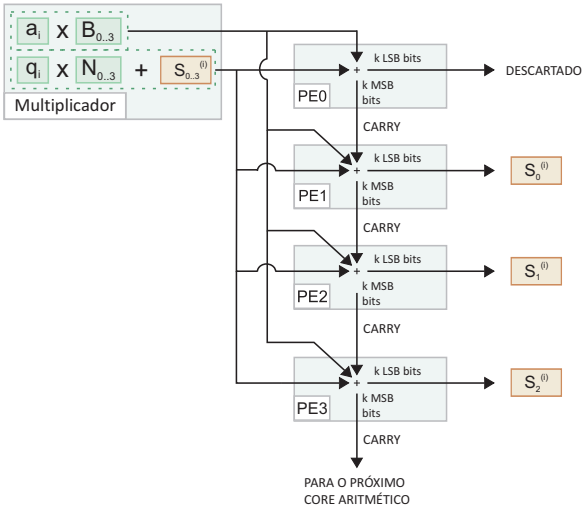


Figura 25: Operações aritméticas realizadas pelo *core* aritmético 1.

A arquitetura multiplexada apresenta uma estrutura adequada para aplicação em FPGA. Mesmo podendo realizar multiplicações modulares com operandos da ordem de 1024 bits, o número de multiplicadores $k \times k$ *bits* não é muito elevado e, para certas famílias de FPGAs, são sintetizados através de blocos DSPs. Os blocos DPS (referenciados como DSP48) são estruturas de processamento digital de sinais disponíveis em dispositivos *Virtex* da *Xilinx*. Um DSP48 combina, normalmente, um multiplicador 18×18 *bits* com um somador de 48 *bits* e um multiplexador programável de modo a selecionar as entradas do somador. A arquitetura interna dos blocos multiplicadores é mostrada na Figura 26.

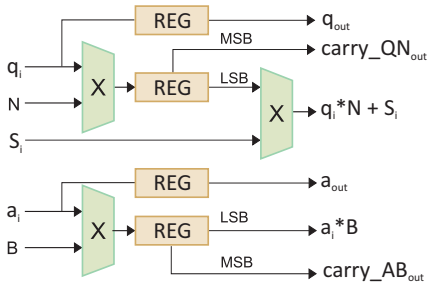


Figura 26: Arquitetura interna do bloco multiplicador.

O cálculo do termo q_i é realizado por um bloco combinacional com arquitetura idêntica ao bloco quociente apresentado na arquitetura sistólica.

Os sinais de *carry* propagados na arquitetura multiplexada são os *bits* mais significativos dos cálculos $q_i.N$ e $S_i + a_i.B$. Esses sinais de *carry* são propagados entre blocos multiplicadores.

Os sinais de *carry* provenientes do bloco multiplicador posicionado no último *core* aritmético é enviado ao último elemento de processamento da arquitetura. Outro sinal de *carry*, C_{PE}, refere-se ao *bit* mais significativo da adição entre os termos $q_i.N$ e $S_i + a_i.B$ em cada elemento de processamento.

Ao final da última iteração do algoritmo de Montgomery, o resultado $S_{i+1} = A.B.R^{-1}$ mod N é enviado (palavra por palavra) à um multiplexador de m entradas. Este resultado é, como veremos na próxima seção, armazenado em uma memória RAM alocada em uma estrutura própria para executar uma exponenciação modular. Neste caso, o resultado de cada multiplicação modular é utilizado como parâmetro de entrada para as próximas multiplicações modulares.

4.2.1 Elementos de Processamento - PE

A arquitetura multiplexada é composta por m elementos de processamento (sendo m o número de palavras de cada operando de entrada e também o número total de iterações do algoritmo de Montgomery). Devido à utilização de um bloco multiplicador em cada *core* aritmético, cada elemento de processamento necessita calcular apenas uma adição entre duas palavras de $2k$ *bits* e fornecer os *bits* menos significativos desta adição como sendo uma palavra do resultado S_{i+1}, referente à iteração atual. O primeiro elemento de processamento descarta os k *bits* menos significativos desta adição, de modo a executar uma operação de deslocamento à direita que corresponde à divisão de $(S_i + q_i \times N + a_i \times B)$ por 2^k.

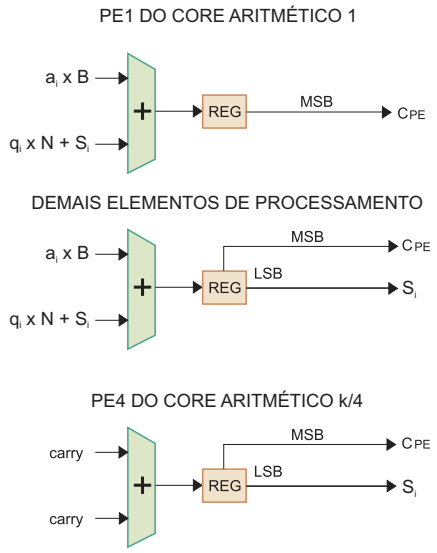


Figura 27: Elementos de Processamento.

Os $k+1$ *bits* mais significativos da adição realizada nos elementos de processamento são propagados como sinais de *carry*. O último elemento de processamento da arquitetura (PE_{m-1})

é responsável por fornecer duas palavras do resultado S_{i+1}, sendo que este elemento de processamento tem, também, como parâmetros de entrada os sinais de *carry* provenientes do bloco multiplicador alocado no último *core* aritmético. A Figura 27 apresenta a arquitetura interna dos elementos de processamento.

4.2.2 Estimativa de Latência

A latência da arquitetura multiplexada pode ser estimada a partir da Figura 28. Este fluxograma apresenta as operações atômicas da arquitetura e os ciclos de *clock* gastos para a execução das *m* iterações do algoritmo de Montgomery.

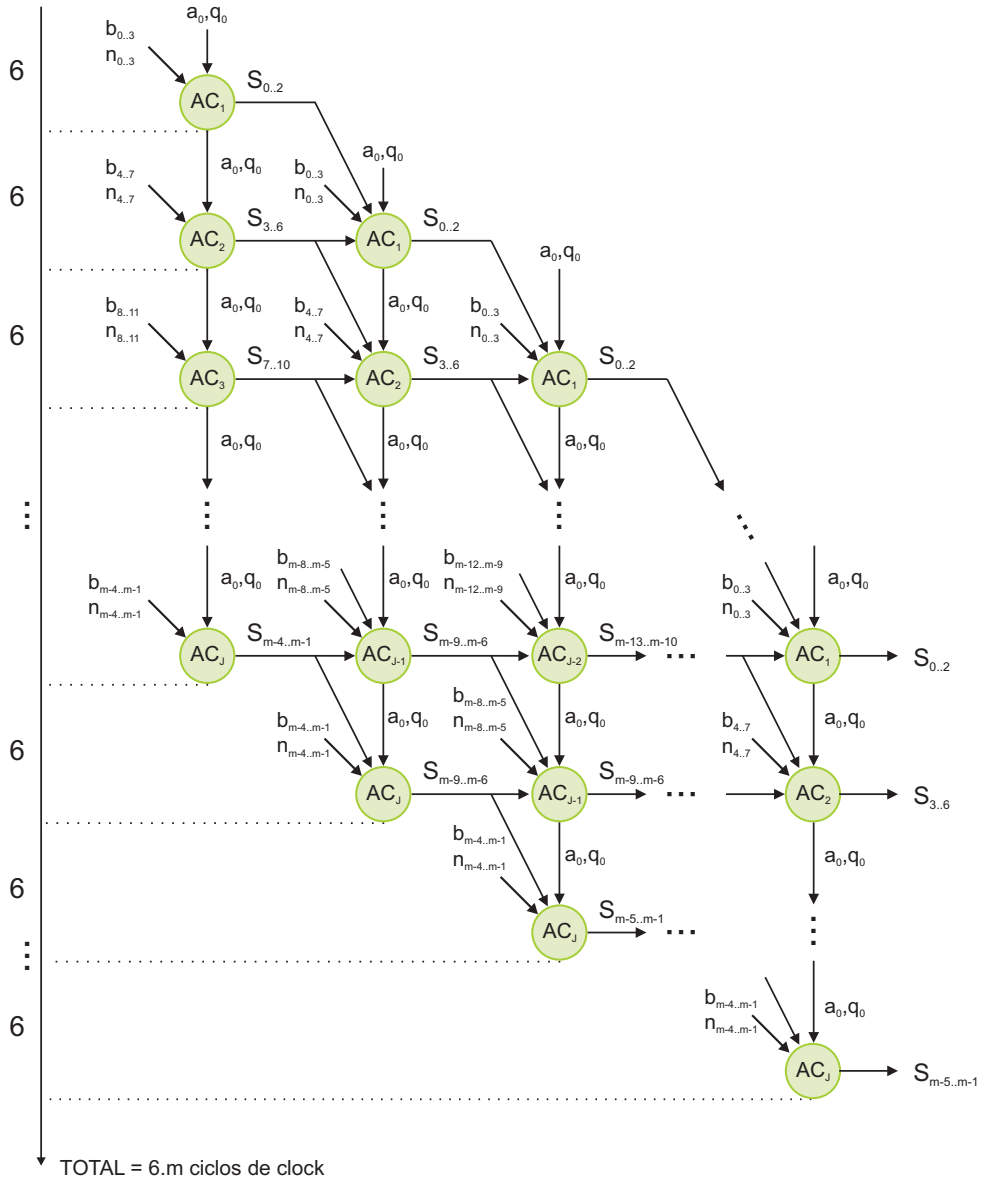


Figura 28: Estimativa de latência da arquitetura multiplexada.

O número total de ciclos de *clock* necessários para uma multiplicação modular através da

arquitetura multiplexada é definido da seguinte maneira: inicialmente, m ciclos de *clock* são reservados para o armazenamento do operando B em cada memória RAM alocada em cada *core* aritmético. Este operando é lido de uma memória RAM alocada na estrutura da exponenciação modular. Considerando-se que o modulo N já esteja alocado nas memórias RAM localizadas dentro dos *cores* aritméticos, a primeira iteração necessita de m ciclos de *clock* e, cada iteração resultante do algoritmo de Montgomery necessita de $6.m$ ciclos de *clock*. Portanto, o número total de ciclos de *clock* para uma multiplicação modular é dado por: $n_{MM} = m + m + 6.m = 8.m$.

4.3 Síntese Lógica e Utilização de Recursos em FPGAs

Ambas as arquiteturas foram desenvolvidas em linguagem de descrição de *hardware* VHDL e mapeadas para FPGAs das famílias Virtex-4 e Virtex-5 (Xilinx,2010). A Tabela 3 apresenta os resultados de síntese lógica de cada arquitetura. Os resultados foram obtidos sem quaisquer esforços de sínteses para área ou máxima frequência de *clock*.

Tabela 3: Resultados de Síntese e Desempenho das Arquiteturas Propostas.

Virtex-4					
Arquitetura Sistólica					
n	k	Slices	BRAM(Bytes)	DSP48	Freq. Máx(MHz)
512	16	3322	0	68	110
512	32	4199	0	36	78
1024	16	7889	0	129	110
Arquitetura Multiplexada					
n	k	Slices	BRAM(Bytes)	DSP48	Freq. Máx(MHz)
512	16	2199	128	32	120
512	32	2499	128	32	80
1024	16	4876	256	64	120
1024	32	5118	256	64	80
Virtex-5					
Arquitetura Sistólica					
n	k	Slices	BRAM(Bytes)	DSP48	Freq. Máx(MHz)
512	16	3205	0	68	130
512	32	3876	0	36	95
1024	16	6642	0	130	130
Arquitetura Multiplexada					
n	k	Slices	BRAM(Bytes)	DSP48	Freq. Máx(MHz)
512	16	2078	128	32	120
512	32	2370	128	32	90
1024	16	4876	256	64	120
1024	32	5005	256	64	90

Pela análise dos resultados da Tabela 3 percebe-se que a arquitetura sistólica apresenta frequências máximas de *clock* maiores que a arquitetura multiplexada. Porém, a arquitetura multiplexada ocupa uma quantidade consideravelmente menor de recursos lógicos.

4.4 Análise de Desempenho

Nesta seção apresenta-se uma análise de desempenho para as arquiteturas sistólica e multiplexada propostas.

4.4.1 Análise de Tempo de Execução

4.4.1.1 Multiplicação Modular de Montgomery

A Tabela 4 os resultados obtidos para ambas as arquiteturas em termos de tempo de execução de uma multiplicação modular de Montgomery.

Tabela 4: Resultados de Desempenho das Arquiteturas Propostas.

Virtex-4			
Arquitetura Sistólica			
n	k	Ciclos de *Clock*	Tempo para Multiplicação Modular
512	16	160	1.45 μs
512	32	80	1.025 μs
1024	16	320	2.91 μs
Arquitetura Multiplexada			
n	k	Ciclos de *Clock*	Tempo para Multiplicação Modular
512	16	256	2.13 μs
512	32	128	1.6 μs
1024	16	512	4.26 μs
1024	32	256	3.2 μs
Virtex-5			
Arquitetura Sistólica			
n	k	Ciclos de *Clock*	Tempo para Multiplicação Modular
512	16	160	1.23 μs
512	32	80	0.842 μs
1024	16	320	2.46 μs
Arquitetura Multiplexada			
n	k	Ciclos de *Clock*	Tempo para Multiplicação Modular
512	16	256	2.13 μs
512	32	128	1.42 μs
1024	16	512	4.26 μs
1024	32	256	2.84 μs

Os resultados da Tabela 4 evidenciam que a arquitetura multiplexada apresenta um tempo ligeiramente maior para a execução da multiplicação modular. Isso ocorre em função da multiplexação dos processos aritméticos e consequente aumento no número de ciclos de *clock*. No entanto, essa desvantagem em termos de tempo de execução é suprida pela redução de ocupação de recursos lógicos, principalmente em relação ao uso de blocos DSP48.

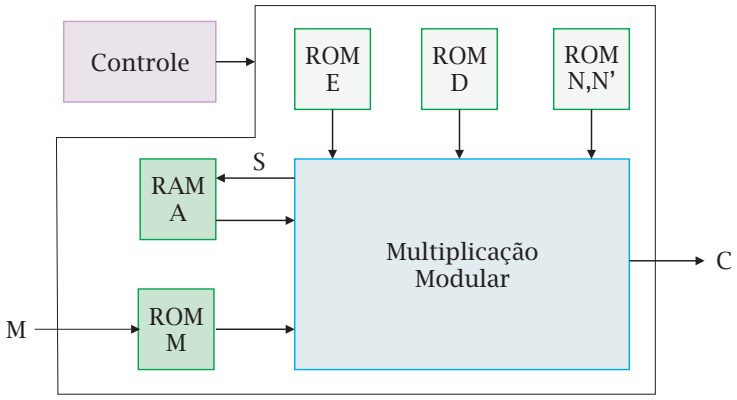


Figura 29: Arquitetura para exponenciação modular.

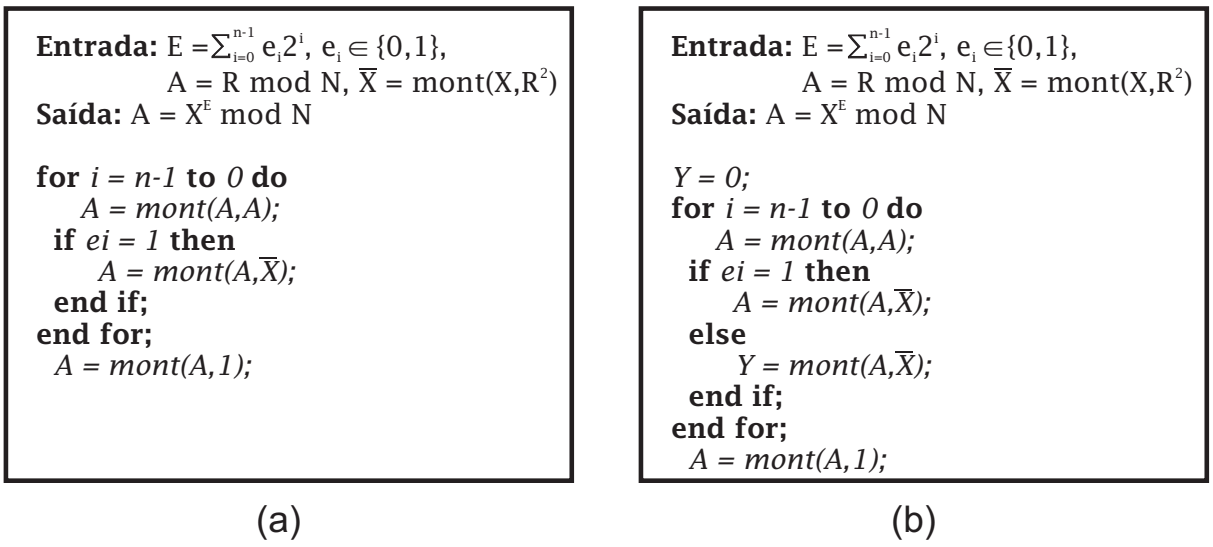


Figura 30: (a) *Left-to-Right Square-and-Multiply* e (b) *Square-and-Multiply Always*.

4.4.1.2 Exponenciação Modular: Aplicação no Algoritmo RSA

De modo a prover resultados em termos de aplicações para o algoritmo RSA, apresenta-se na Figura 29 uma estrutura para exponenciação modular que instancia, por sua vez, as arquiteturas propostas para multiplicação modular de Montgomery.

Um bloco de controle gerencia a ordem das sucessivas execuções de multiplicação ou quadrado modular. Os métodos de exponenciação modular que as arquiteturas são capazes de realizar são *left-to-right square-and-multiply* e *square-and-multiply always*. Ambos os métodos são ilustrados na Figura 30. Considera-se que os parâmetros de entrada *A* e *M* (mensagem) já estejam no domínio de Montgomery.

O controle baseia-se na leitura e interpretação dos *bits* dos expoentes (chave pública *e* ou chave privada *d*) que encontram-se armazenados na memória ROM *E* e ROM *D*, respectiva-

mente. A mensagem de entrada $\overline{M}$ (no domínio de Montgomery), o termo auxiliar $A = R \ mod \ N$ e o modulo N são armazenados nas memórias RAM M, RAM A. O modulo N e a inversa do módulo N' são armazenados na memória ROM N.

A Tabela 5 apresenta os resultados para os processos de cifração e decifração do algoritmo RSA. Os resultados são apresentados em termos de número de ciclos de *clock* e tempo necessário em *m*s considerando a frequência máxima de *clock*. Para o tempo de cifração, considera-se que a chave pública é $e = 2^{16} - 1$.

Tabela 5: Aplicação em Exponenciações Modulares - RSA

Virtex-4				
Arquitetura Sistólica				
n	k	Ciclos de *Clock* (Decifração)	Tempo para Cifração	Tempo para Decifração
512	16	245760	29.09 μs	2.23 *m*s
512	32	122880	20.51 μs	1.57 *m*s
1024	16	491520	58.18 μs	4.46 *m*s
Arquitetura Multiplexada				
n	k	Ciclos de *Clock* (Decifração)	Tempo para Cifração	Tempo para Decifração
512	16	393216	42.66 μs	3.27 *m*s
512	32	196608	32 μs	2.45 *m*s
1024	16	786432	85.33 μs	6.55 *m*s
1024	32	393216	64 μs	4.91 *m*s
Virtex-5				
Arquitetura Sistólica				
n	k	Ciclos de *Clock* (Decifração)	Tempo para Cifração	Tempo para Decifração
512	16	245760	24.61 μs	1.89 *m*s
512	32	122880	16.84 μs	1.29 *m*s
1024	16	491520	49.23 μs	3.78 *m*s
Arquitetura Multiplexada				
n	k	Ciclos de *Clock* (Decifração)	Tempo para Cifração	Tempo para Decifração
512	16	393216	42.66 μs	3.27 *m*s
512	32	196608	28.44 μs	2.17 *m*s
1024	16	786432	85.33 μs	6.55 *m*s
1024	32	393216	56.88 μs	4.37 *m*s

Na Tabela 5 os resultados foram obtidos para exponenciações modulares com o algoritmo *left-to-right square-and-multiply*, o que significa que aproximadamente $1.5n$ multiplicações modulares são efetuadas, sendo n o tamanho dos operandos de entrada. Caso aplique-se o algoritmo *square-and-multiply always*, o número de multiplicações modulares é exatamente $2.n$.

4.4.1.3 Comparação com Trabalhos Relacionados

A Tabela 6 apresenta uma comparação das arquiteturas propostas com trabalhos relacionados. Todos os trabalhos referenciados nesta tabela utilizam o algoritmo de Montgomery como método para multiplicação modular e apenas resultados considerando operandos de entrada de

1024 *bits* são apresentados. O tempo de execução de uma decifração para RSA, quando não explícito nas referências, foi estimado considerando-se uma exponenciação modular com o algoritmo *left-to-right square-and-multiply*.

Tabela 6: Comparação com Trabalhos Relacionados.

Trabalho	Tecnologia (FPGA)	Slices	Frequência de *Clock*	Ciclos	Tempo para Decifração do RSA
Systolic	XC5VLX110T	6642	130 MHz	491520	3.78 ms
Multiplexed	XC5VLX110T	5005	90 MHz	393216	4.37 ms
(Tenca,2001)	Virtex-2	3937	100 MHz	3270000	32.7 ms
(Huang,2008)	Virtex-2	4178	100 MHz	1670000	16.7 ms
(Blum,2001)	XC40250XC	6633	68 MHz	812600	11.95 ms
(McIvor,2005)	Virtex-2	5709	101.86 MHz	354472	3.48 ms

Comparando as arquiteturas propostas com trabalhos relacionados, percebe-se que o trabalho de McIvor *et al* apresenta um menor tempo de execução na decifração do RSA. Porém, conforme visto no capítulo anterior, o número de multiplicadores empregados nessa arquitetura é muito elevado, e caso a área seja o requisito principal, não compensa a baixa latência dessa implementação. A arquitetura de Huang *et al* apresenta um menor grau de utilização de recursos do FPGA, no entanto, o número de ciclos de *clock* para uma exponenciação modular é muito elevado, o que eleva o tempo de decifração do RSA.

4.5 Resumo

As arquiteturas sistólica e multiplexada propostas apresentam baixa latência em relação ao número de ciclos de *clock* necessários para uma multiplicação modular de Montgomery. A arquitetura sistólica apresenta valores de frequência máxima de *clock* mais elevada, em função do alto grau de paralelismo das operações. No entanto, a arquitetura multiplexada apresenta ocupação reduzida de elementos lógicos do FPGA, o que torna esta implementação mais flexível quando requisitos de área ou ocupação de recursos lógicos devem ser atendidos. Pode-se afirmar que ambas as arquiteturas são sistólicas, porém a arquitetura multiplexada reutiliza elementos aritméticos.

5 *FLUXO DE ANÁLISE DE ROBUSTEZ*

Este capítulo descreve os resultados experimentais obtidos através de ataques por canais laterais. Inicialmente, descreve-se os modelos de consumo de potência e emissões eletromagnéticas em dispositivos CMOS e o modo no qual estes modelos são tratados como fontes de informações. Em seguida, os ataques por canais laterais simples SPA e SEMA são aplicados à arquitetura multiplexada e a uma arquitetura RSA padrão (Opencores,2010). Por fim, como um melhoramento dos ataques SPA e SEMA, apresenta-se ataques baseados na demodulação em amplitude das curvas de emissões eletromagnéticas e por consumo de potência. Esta última análise tem como objetivos mostrar a influência do sinal de *clock* nas curvas analisadas e demonstrar que a informação confidencial pode ser melhor visualizada em sinais demodulados em amplitude.

5.1 Modelos de Fuga de Informações

Consumo de potência e emissão eletromagnética em circuitos integrados possuem propriedades diferentes. Primeiramente, a fonte e a natureza das fugas de energia e emissão eletromagnéticas são diferentes. Quando mede-se o consumo de potência de um circuito integrado, mede-se a soma do consumo realizado por todas as portas lógicas que compõem o circuito. Além disso, a medição do consumo de potência retorna um valor escalar representando a corrente consumida para um dado instante de tempo por todo o circuito. No caso de emissões eletromagnéticas, estas ocorrem principalmente devido ao deslocamento da corrente através das trilhas metálicas relacionadas ao chaveamento de portas lógicas.

Nesta seção descreve-se os modelos de consumo de potência e emissões eletromagnéticas dos dispositivos CMOS.

5.1.1 Consumo de Potência em Dispositivos CMOS

A tecnologia CMOS é certamente a mais usada atualmente em projetos de microprocessadores, microcontroladores, memórias RAM estáticas e outras aplicações de circuitos digitais. Duas importantes características dos dispositivos CMOS são a alta imunidade ao ruído e baixo consumo de potência estática. O consumo de potência significante destes dispositivos ocorre quando os transistores estão sendo chaveados entre diferentes estados. Do ponto de vista de ataques por canais laterais sobre módulos criptográficos, este tipo de atividade de chaveamento é tida como principal fonte de fuga de informações confidenciais.

As portas lógicas CMOS estáticas apresentam três fontes distintas de consumo de potência (Rabaey,1996). A primeira ocorre devido as correntes de fugas em transistores. A segunda fonte de consumo de potência é chamada corrente de caminho direto, ou seja, é uma corrente presente em curtos períodos de tempo durante o chaveamento de uma porta lógica em que os transistores NMOS e PMOS conduzem simultaneamente. Esta segunda corrente também é conhecida por corrente de curto-circuito. Finalmente, a terceira e mais importante fonte de consumo de potência (e mais relevante do ponto de vista dos ataques por canais laterais) é devido a carga e descarga da capacitância de carga C_L, como mostrado através do inversor CMOS na Figura 31.

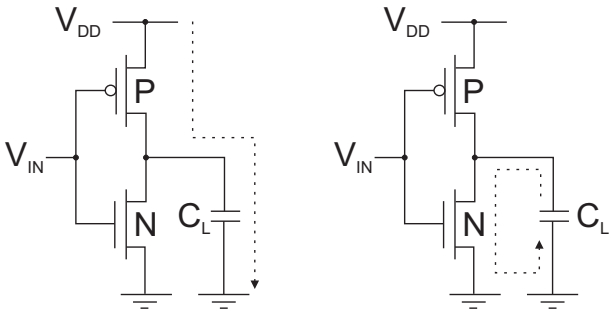


Figura 31: Inversor CMOS estático.

A expressão do consumo de potência dinâmico de um inversor é dado por:

$$P_{din} = C_L V_{DD}^2 P_{0\to1} f \tag{5.1}$$

no qual o termo $P_{0\to1} f$ é chamado de atividade de chaveamento ($P_{0\to1}$ é a probabilidade de uma transição de 0 para 1 ocorrer), f é a frequência de *clock* do dispositivo e V_{DD} é a tensão da fonte de alimentação.

A medição do consumo de potência em um dispositivo CMOS (tanto do terra quanto da fonte de alimentação), portanto, resultará em maiores magnitudes durante o carregamento da

capacitância C_L. No caso da descarga, mede-se a corrente de caminho direto.

5.1.2 Emissões Eletromagnéticas em Dispositivos CMOS

Os circuitos integrados são constituídos de milhões de transistores e interconexões nos quais a corrente que flui é dependente dos dados processados. Esses pequenos movimentos de cargas produzem um campo magnético variável, que por consequência produz um campo elétrico variável. Em ataques por análise eletromagnética, a monitoração dessas radiações, que são dependentes dos dados sendo processados, permite, então, obter informações sobre os dados sendo manipulados pelo dispositivo.

Do ponto de vista teórico, essas emissões eletromagnéticas podem ser explicadas como segue. Primeiramente, assume-se que a região localizada a menos de um comprimento de onda de distância da fonte de emissão é chamada de região de campo próximo (*near-field zone*). Isso permite usar a lei de Biot-Savart para descrever a variação do campo magnético $\overrightarrow{B}$:

$$d\overrightarrow{B} = \frac{\mu I \overrightarrow{dl} \times \widehat{r}}{4\pi|\overrightarrow{r}|^2} \tag{5.2}$$

no qual I é a corrente através de um condutor infinitesimal de comprimento $\overrightarrow{dl}$, μ é a permeabilidade magnética e $\widehat{r}$ é um vetor especificando a distância entre a corrente no condutor e um ponto no campo próximo ($\widehat{r} = \frac{\overrightarrow{r}}{|\overrightarrow{r}|}$).

Segundo, a lei de Faraday especifica que qualquer mudança no campo magnético induz uma tensão (*fem*) na redondeza do campo magnético:

$$fem = -N\frac{d\Phi}{dt} \tag{5.3}$$

$$d\Phi = \int_{surface} \overrightarrow{B}.\overrightarrow{dS} \tag{5.4}$$

no qual N é o número de voltas em uma bobina (na sonda, por exemplo) e Φ representa o fluxo magnético. Considerando, agora, que o condutor de corrente tenha comprimento finito, pode-se reduzir a equação de Biot-Savart à seguinte expressão:

$$\overrightarrow{B} = \frac{\mu I}{2\pi d}\widehat{a_\varphi} \tag{5.5}$$

em que d é a distância de um ponto no campo próximo ao condutor e $\widehat{a_\varphi}$ é um vetor azimutal orientado de acordo com este condutor. Esta última equação define que quanto mais próximo

do circuito integrado posiciona-se a sonda, maior é o campo magnético medido.

Embora estas equações não descrevam o comportamento exato do campo magnético, elas enfatizam dois detalhes importantes: (1) a magnitude do campo magnético (ou intensidade de corrente I) é dependente dos dados processados e (2) a orientação do campo depende diretamente da orientação da corrente, uma vez que $\widehat{a_\varphi} = \frac{\overrightarrow{dl}\times\widehat{r}}{|\overrightarrow{dl}\times\widehat{r}|}$.

A escolha da sonda é outro detalhe importante em medições do campo próximo. Uma sonda normalmente refere-se à uma pequena antena dotada de um mecanismo que permite uma movimentação sobre o campo a ser medido. As propriedades eletromagnéticas da sonda são assumidas como sendo conhecidas.

5.2 Aquisição de Curvas EM e de Consumo de Potência

As curvas de emissões eletromagnéticas foram coletadas através de uma plataforma de medições composta por:

1. Osciloscópio, com largura de banda de 2.5 GHz e taxa de amostragem de 40 GS/s.
2. Amplificador de baixo ruído, com ganho de 48 dB e 1GHz de largura de banda.
3. Uma sonda (ou antena), de $500\mu m$.
4. Um estágio motorizado para controle do posicionamento da sonda.
5. Uma placa de prototipagem FPGA (Spartan3-1000 (Xilinx,2008)).
6. Um PC para controle das medições munido com o *software* Matlab.

Um processo de cartografia foi realizado, de modo a adquirir curvas de emissões eletromagnéticas em 34×34 pontos (x,y) sobre a área do *die* do circuito integrado. Com isso, através de cálculos estatísticos pode-se determinar os pontos (x,y) que apresentam maior intensidade de fuga de informações. Descrições detalhadas dos processos de cartografia e como localizar estes pontos de maior fuga podem ser encontradas em (Dehbaoui,2009) e (Dehbaoui,2009a).

As curvas de consumo de potência foram obtidas com uma sonda que monitora a corrente elétrica na entrada de um regulador de tensão de 1.2V localizado na placa de desenvolvimento Spartan3-1000. Um amplificador de baixo ruído é posicionado entre a sonda e um osciloscópio. As arquiteturas RSA padrão e multiplexada foram configuradas no FPGA e curvas médias de consumo de potência foram coletadas.

Para ambas as medições de curvas de emissão eletromagnética e consumo de potência, a arquitetura multiplexada foi configurada com três diferentes tamanhos de palavras: 8, 16 e 32 *bits*. O objetivo é analisar o efeito do tamanho de palavra sobre o vazamento de informações. Uma vez que as operações aritméticas no algoritmo de Montgomery são feitas em um contexto de múltipla precisão, diversas adições e multiplicações com precisão simples devem ser realizadas ao longo das iterações do algoritmo. Portanto, quanto maior for o tamanho de palavra, maior é o número de transistores chaveados por ciclo de *clock* e, consequentemente, mais intensa é a emissão eletromagnética e maior é o consumo de potência. De modo similar, quanto maior for o tamanho das palavras lidas e escritas nas memórias RAM da arquitetura, mais informações podem vazar devido à esses acessos às memórias.

5.3 Análise Eletromagnética

Nesta seção são apresentados os ataques por análise eletromagnética sobre a arquitetura multiplexada. No caso dos ataques sobre a implementação do algoritmo *left-to-right square-and-multiply*, o objetivo é identificar a sequência de operações de quadrados e multiplicações modulares, a fim de associá-los diretamente aos *bits* do expoente. Já os ataques sobre o algoritmo *square-and-multiply always* busca encontrar a ocorrência das multiplicações falsas, as quais indicam a presença de um *bit* de valor zero no expoente.

A análise eletromagnética também é realizada sobre uma arquitetura RSA padrão, que efetua a exponenciação modular através do método *right-to-left square-and-multiply*. Esta arquitetura não apresenta nenhuma contra-medida a nível de algoritmo ou implementação.

O ataque SEMA é descrito primeiramente. Em seguida, apresenta-se um ataque baseado na demodulação em amplitude das curvas eletromagnéticas, com uma breve descrição teórica das propriedades dos sinais modulados em amplitude.

5.3.1 Análise Eletromagnética Simples (SEMA - *Simple Electromagnetic Analysis*)

Através de um ataque SEMA, um adversário tenta relacionar o traço que representa a emissão eletromagnética com os dados sendo manipulados pelo dispositivo criptográfico através da análise de uma única curva. Diferentemente da análise diferencial, a análise simples não envolve a aquisição de várias curvas de emissão eletromagnética para relacioná-las com hipóteses que dependam de detalhes de implementação. Este método tenta revelar os *bits* da chave criptográfica através de uma análise visual.

De modo a validar a análise eletromagnética simples, a Figura 32 mostra uma curva SEMA (sendo esta uma curva média obtida a partir da aquisição de 20 curvas, de modo a reduzir o ruído) adquirida a partir da atividade interna da arquitetura RSA padrão. Na Figura, estão selecionados os quadros de tempo correspondentes ao cálculo envolvendo cada *bit* do expoente. Analisando-se o algoritmo de exponenciação modular em questão, pode-se deduzir quando o *bit* do expoente é *zero* ou *um*. Portanto, este projeto padrão é extremamente vulnerável à análise SEMA.

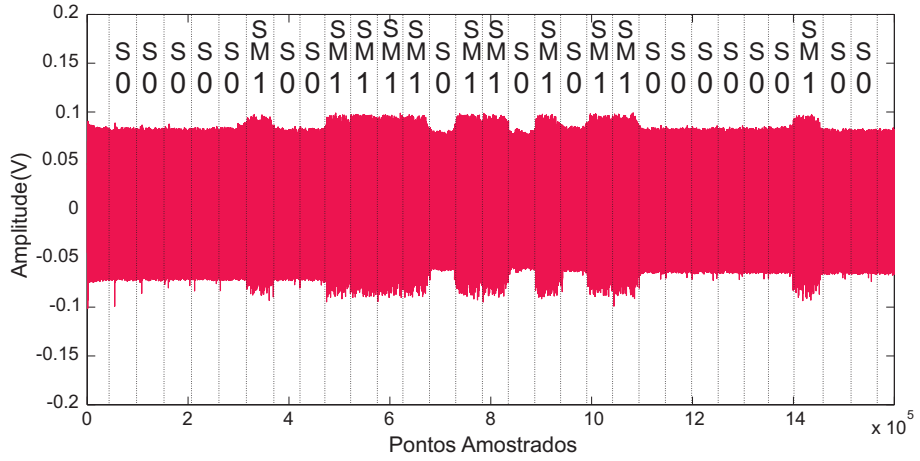


Figura 32: Emissão eletromagnética da arquitetura RSA padrão (Opencores,2010).

A Figura 33 apresenta uma curva média SEMA (para quaisquer coordenadas x,y) adquirida sobre a arquitetura multiplexada implementada com o algoritmo *left-to-right square-and-multiply*. Os tamanhos de palavra considerados são 8, 16 e 32 *bits*. O formato de cada traço permite distinguir os quadros de tempo relacionados aos sucessivos cálculos de multiplicações e quadrados modulares. Uma vez que o primeiro *bit* do expoente é sempre *um*, as três primeiras execuções são conhecidas, conforme indica a Figura 33.

No entanto, não há diferenças visíveis entre os traçados das multiplicações modulares (T_{1-8}), ou seja, as multiplicações e quadrados modulares não são discerníveis. Deste modo, pode-se concluir que a arquitetura multiplexada é robusta contra análise eletromagnética simples. A razão para isso é que a arquitetura multiplexada realiza os processos de quadrado e multiplicação modulares com o mesmo bloco de *hardware* e nem ao menos detalhes relacionados ao controle da arquiteturas auxiliam no discernimento entre ambas as operações. Em seguida, um ataque baseado na demodulação em amplitude das curvas eletromagnéticas é realizado, de modo a explorar as fraquezas da arquitetura multiplexada e destacar as vantagens da demodulação AM em ataques por canais laterais.

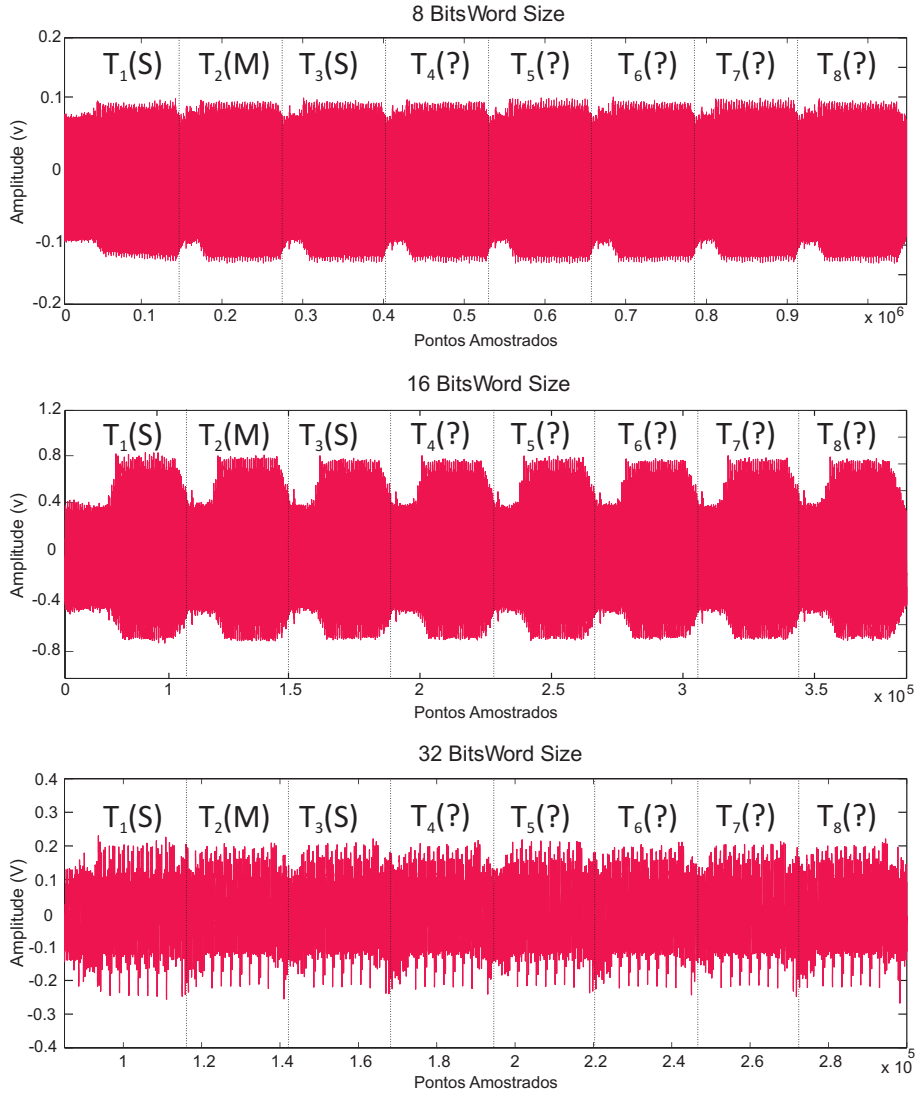


Figura 33: Emissão eletromagnética da arquitetura multiplexada.

5.3.2 Análise Eletromagnética Simples e por Demodulação (SDEMA - *Simple and Demodulated Electromagnetic Analysis*)

Um sinal de informações $f(t) = A_f cos(2\pi f_m t)$, com uma frequência f_m, quando modulado em amplitude por uma onda portadora $p(t) = cos(2\pi f_c t)$, com uma frequência f_c, resulta no sinal modulado em amplitude $s_{am}(t)$:

$$s_{am}(t) = A_c(t)A_f(t)cos(2\pi f_m t)cos(2\pi f_c t) \tag{5.6}$$

no qual $A_c(t)$ e $A_f(t)$ são as amplitudes da onda portadora e do sinal de informações, respectivamente, em um instante de tempo t. A densidade espectral de potência do sinal modulado $s_{am}(t) = f(t)cos(2\pi f_c t)$, de acordo com a propriedade da modulação da transformada de Fourier, pode ser descrita por:

$$\Phi(\omega) = \frac{1}{2}F(\omega - \omega_c) + \frac{1}{2}F(\omega - \omega_c) \tag{5.7}$$

no qual $\omega = 2\pi f$ e $\omega_c = 2\pi f_c$.

Uma vez que a maioria dos projetos de circuitos integrados digitais possuem um sinal de *clock*, os canais laterais de consumo de potência e emissão electromagnética transferem informações sobre os dados sendo processados como sendo sinais modulados em amplitude (AM), em diferentes larguras de banda. Assim, o sinal de *clock* atua como onda portadora e o sinal de informações $f(t)$ é representado pelo consumo de potência ou emissão electromagnética em certos instantes de tempo. A onda portadora $p(t)$, desse modo, é uma onda quadrada que pode ser representada por uma série de Fourier.

$$p(t) = \frac{4}{\pi} \sum_{n=1,3,5...}^{\infty} \frac{1}{n} sin(2\pi n f_c t) \tag{5.8}$$

Cada componente senoidal da equação acima contem uma frequência específica (f_c, $3f_c$, $5f_c$,...). Estas são as harmônicas de f_c e todas irão modular o sinal de informações em diferentes larguras de bandas. A Figura 34 apresenta a densidade espectral EM associada ao processamento de diferentes mensagens de entrada para uma arquitetura RSA padrão. Conforme ilustra a figura, a distribuição espectral de potência ao redor do sinal de *clock* possuem um comportamento que depende dos dados manipulados.

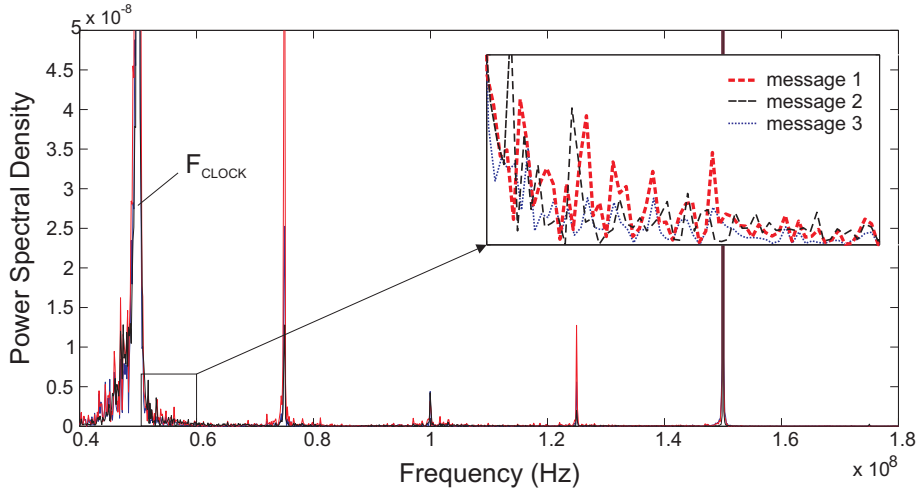


Figura 34: Densidade Espectral de Potência EM.

Logo, pode-se deduzir que a demodulação em amplitude de curvas electromagnéticas pode aumentar significativamente qualidade da análise SEMA. A Figura 35 ilustra essa melhoria, através da diferença de resultados gráficos para uma análise simples (SEMA) e por demodulação (SDEMA), considerando a frequência da onda portadora equivalente à frequência de *clock*.

A curva de emissão eletromagnética apresentada foi adquirida para um processo de exponenciação modular de uma arquitetura RSA padrão, executando o algoritmo *right-to-left square-and-multiply*. Percebe-se através da figura que o sinal demodulado discrimina melhor o vazamento de informações, uma vez que os *bits* do expoente são mais claramente identificados.

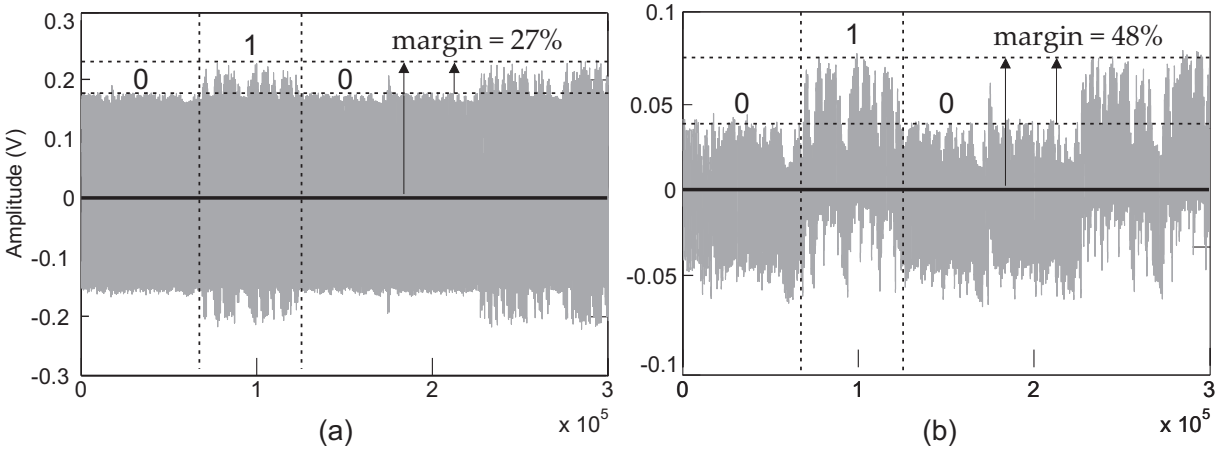


Figura 35: Resultados para SEMA(a) e SDEMA(b).

A aplicação da análise eletromagnética simples e por demodulação tem por objetivo recuperar os *bits* da chave privada d, e *bits* de referência são necessários. Dois métodos são comumente aplicados:

- Chave privada conhecida de referência: nesta solução, uma curva de emissão electromagnética $C_1(t)$ adquirida para uma chave privada conhecida é subtraída de outra curva de emissão electromagnética $C_2(t)$ adquirida para uma chave privada desconhecida. Como consequência, diferenças visíveis fazem-se presentes na curva resultante $C_1(t) - C_2(t)$ quando os *bits* da duas chaves diferem.

- Método por janela deslizante: este método adota somente uma curva de emissão eletromagnética. Considera-se, neste caso, que o primeiro *bit* do expoente é conhecido (*bit* mais significativo, no caso da utilização do algoritmo *left-to-right square-and-multiply*) e seleciona-se os pontos da curva referentes à execuções de quadrado (S) e multiplicação (M) modular conhecidas, para serem usados como referência. Subtraindo-se essa janela de pontos de outra janela que refere-se ao processo de quadrado ou multiplicações modular (S ou M) desconhecido, pode-se recuperar os *bits* do expoente, uma vez que picos de amplitude devem aparecer sobre certos instantes de tempo quando as operações diferem.

Para o ataque SDEMA, a arquitetura multiplexada realiza a exponenciação modular com os algoritmos *left-to-right square-and-multiply* e *square-and-multiply always*. Esta análise enfatiza mais precisamente que partes da arquitetura vazam informações confidenciais.

5.3.2.1 Método por Janela Deslizante para Análise SDEMA

O método por janela deslizante necessita somente uma curva média de emissão eletromagnética para recuperar os *bits* do expoente. Este ataque inicia-se pela seleção de uma janela de tempo contendo os pontos amostrados de uma multiplicação ou quadrado modular conhecido, para ser usada como referência.

Primeiramente, analisa-se a arquitetura multiplexada que é embarcada no algoritmo *left-to-right square-and-multiply*. Logo, sabe-se que as três primeiras multiplicações modulares são: quadrado(S-*square*)-multiplicação(M)-quadrado(S-*square*) modular. Por conveniência, a segunda operação de multiplicação modular (T2, como indicado na Figura 33) é escolhida para ser a referência. A seguir, os pontos amostrados de T2 são subtraídos dos pontos de T4, T5, T6 e assim por diante.

A Figura 36 mostra os resultados das subtrações para os traços demodulados em amplitude. Para esta análise, os tamanhos de palavras considerados na arquitetura multiplexada são 8, 16 e 32 *bits*.

Analisando-se a Figura 36, percebe-se que as diferenças T2-T5 e T2-T7 apresentam maiores picos de amplitude, o que significa que, se T2 é uma operação de multiplicação modular, T5 e T7 devem ser operações de quadrado modular. Por outro lado, as diferenças T2-T4 e T2-T6 são menores, o que significa que T4 e T6 são operações de multiplicação modular. Portanto, com a dedução da sequência de operações de multiplicações e quadrados modulares, deduz-se os *bits* do expoente.

O algoritmo *left-to-right square-and-multiply* não apresenta nenhuma proteção contra ataques SEMA. Como visto anteriormente, a implementação da arquitetura multiplexada apresenta, de certa forma, uma contra-medida contra a análise simples. Porém, quando aplica-se uma técnica SEMA evoluída, pode-se recuperar facilmente os *bits* do expoente. Portanto, a seguir aplica-se o ataque SDEMA sobre uma implementação protegida contra análise simples, ou seja, com o algoritmo *square-and-multiply always*. Este também realiza os sucessivos processos de multiplicação modular com a arquitetura multiplexada.

A Figura 37 apresenta a análise por janela deslizante sobre a implementação do algoritmo *square-and-multiply always*. Similarmente, a janela de pontos de T2 é usada como referência. Após a localização das execuções de multiplicação e quadrado modular, as operações "falsas"de multiplicação modular são reveladas pela localização de picos de amplitude ao final do processo de multiplicação modular. Nestes casos, como mostra a Figura 37, os resultados das execuções representadas por T4 e T8 são operações "falsas"do algoritmo *square-and-multiply always*, e

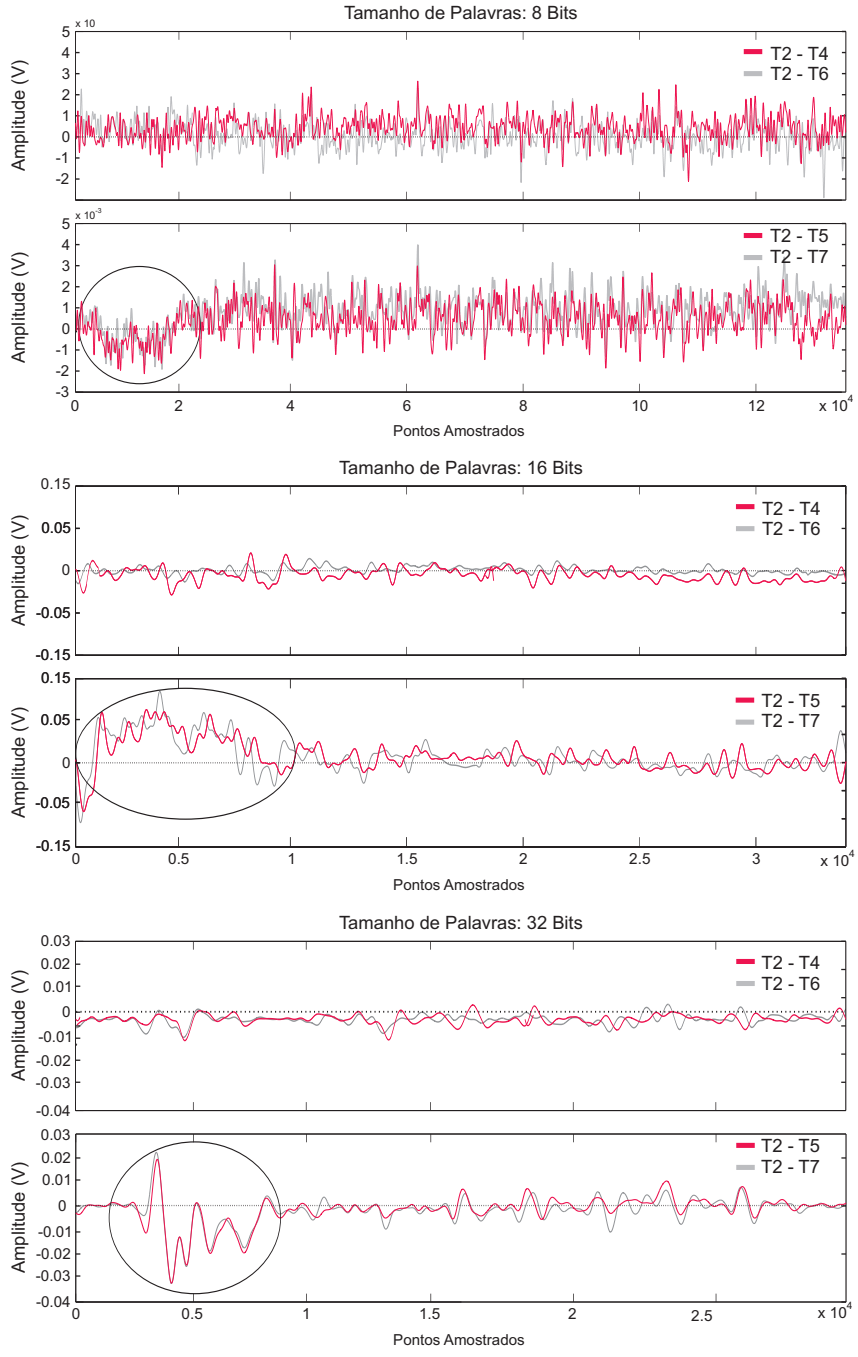


Figura 36: Ataque SDEMA sobre o algoritmo *left-to-right square-and-multiply*.

portanto, não são armazenados em uma memória RAM.

5.4 Análise por Consumo de Potência

Nesta seção, apresenta-se a aplicação de ataques SPA sobre as arquiteturas RSA padrão e multiplexada. A seguir, a análise simples por demodulação é aplicada sobre a arquitetura multiplexada.

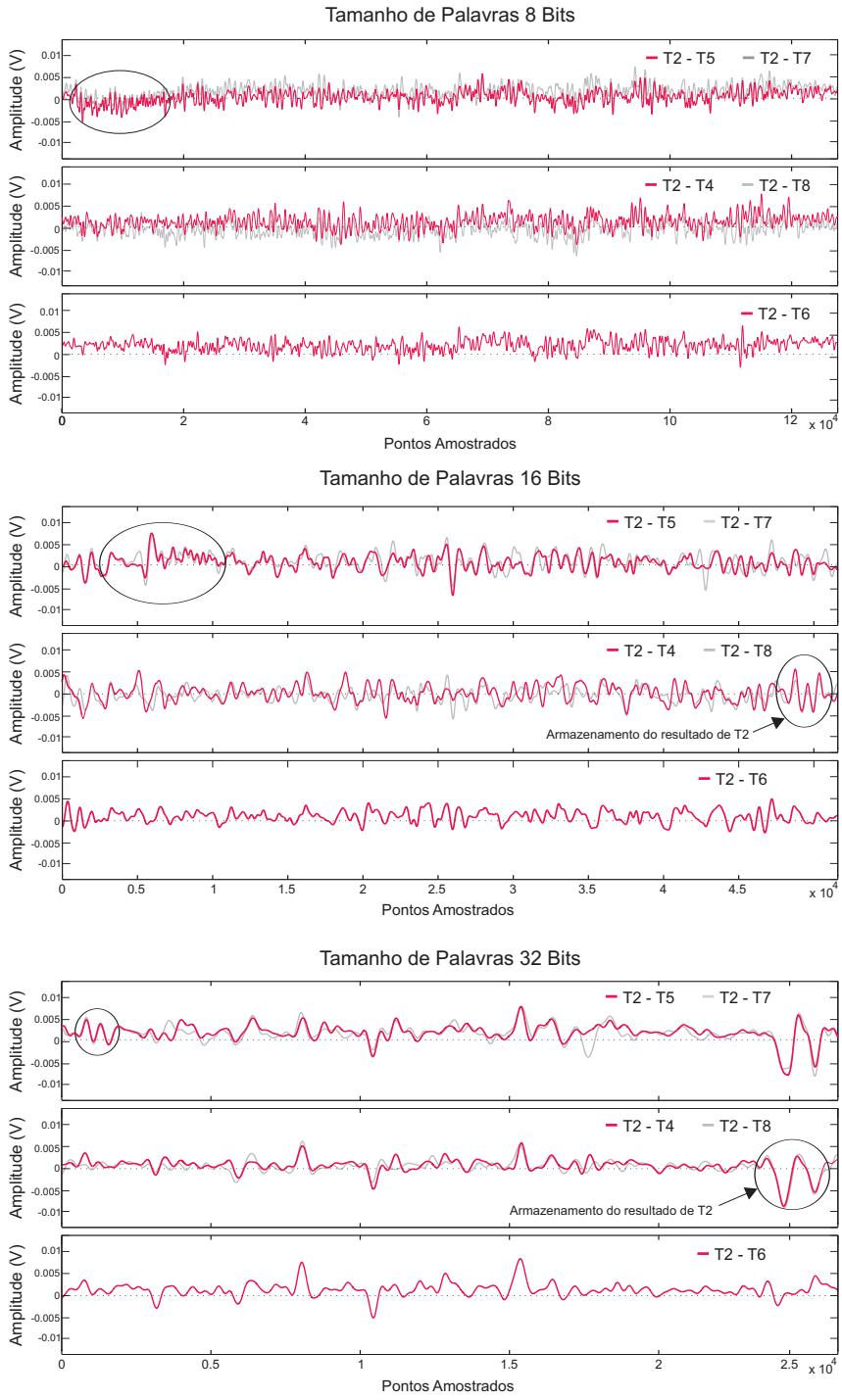


Figura 37: Ataque SDEMA sobre o algoritmo *square-and-multiply always*.

5.4.1 Análise Simples por Consumo de Potência (SPA)

A análise simples por consumo de potência (SPA) foca na relação entre o consumo de potência e os dados sendo manipulados através da análise de uma simples curva. De modo a validar a análise simples por consumo de potência, a Figura 38 mostra a curva de consumo de potência adquirida da arquitetura RSA padrão.

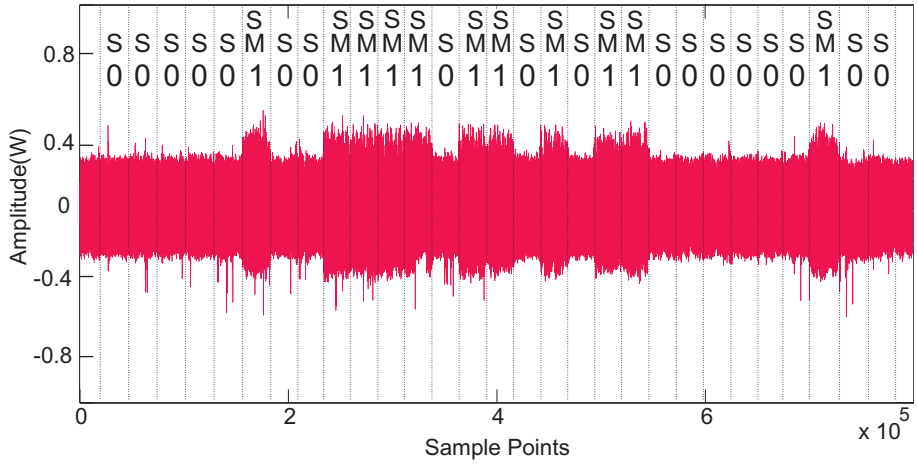


Figura 38: Curva de Consumo de Potência adquirida da Arquitetura RSA Padrão.

O comportamento assimétrico do algoritmo *right-to-left square-and-multiply* é claramente visto na Figura 38. As maiores amplitudes referem-se às execuções em paralelo das operações de multiplicação e quadrado modular. Assim, os *bits* do expoente são facilmente revelados através deste ataque SPA, pois a arquitetura alvo não apresenta nenhuma contra-medida a nível de algoritmo ou *hardware*.

A Figura 39 mostra as curvas de consumo de potência adquiridas da arquitetura multiplexada. Três tamanhos de palavras foram empregados na análise: 8, 16 e 32 *bits*. Os quadros de tempo das sucessivas multiplicações modulares são facilmente discerníveis nestas curvas e, por empregar o algoritmo *left-to-right square-and-multiply*, as três primeiras execuções são conhecidas: *Square*(S)-*Multiply*(M)-*Square*(S). As execuções restantes são desconhecidas.

Na arquitetura multiplexada, as execuções de multiplicação e quadrado modular são feitas com o mesmo bloco de *hardware* e, portanto, nem aspectos referentes à parte de controle são discerníveis para estas operações. Logo, pode-se concluir que a arquitetura multiplexada é robusta contra análise simples por consumo de potência, mesmo considerando vários conhecimentos ao adversário com relação aos detalhes da arquitetura.

5.4.2 Análise Simples e por Demodulação do Consumo de Potência (SDPA - *Simple and Demodulated Power Analysis*)

No ataque SDPA, as curvas de consumo de potência são demoduladas em amplitude de modo a suprimir as componentes harmônicas do sinal de *clock*. Assim como no ataque SDEMA, a arquitetura multiplexada é inserida no contexto da exponenciação modular com os algoritmos *left-to-right square-and-multiply* e *square-and-multiply always*. Nesta análise, o método por janela deslizante também é aplicado sobre curvas médias do consumo de potência.

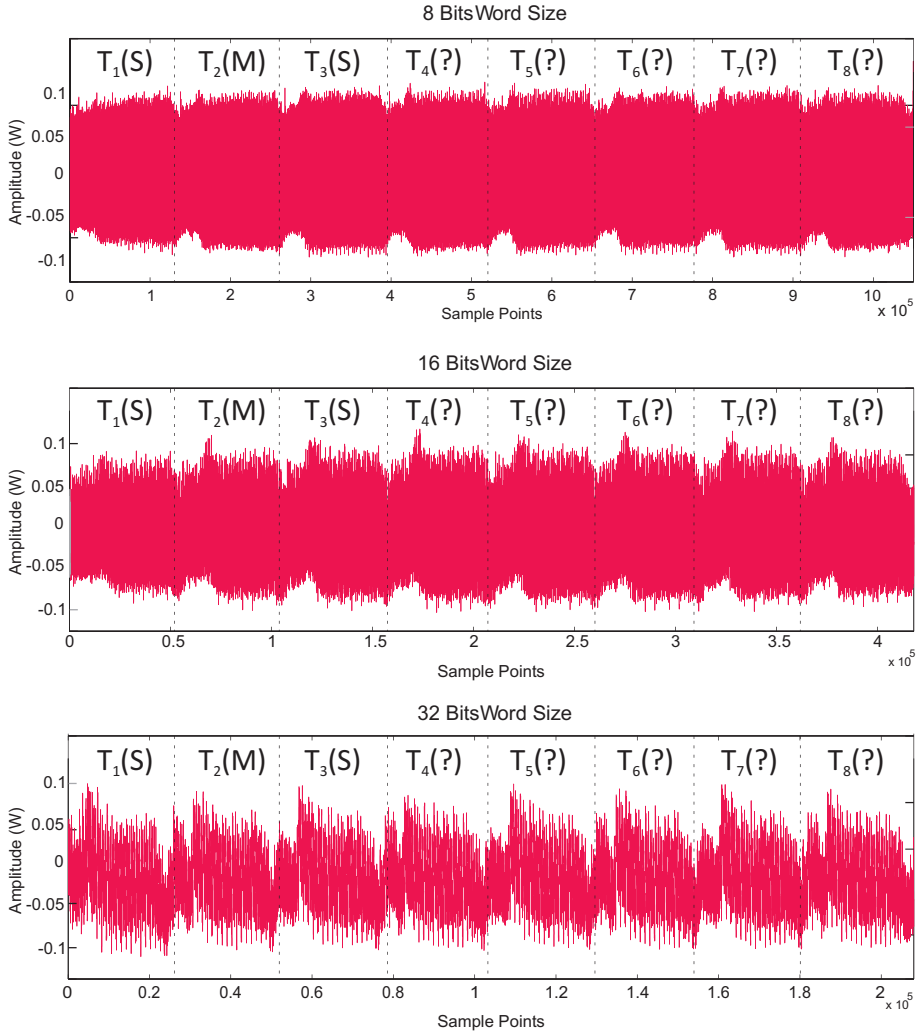


Figura 39: Curvas de consumo de potência e respectivas execuções de multiplicações modulares.

5.4.2.1 Método por Janela Deslizante para Análise SDPA

Na Figura 39, as curvas de consumo de potência permitem que um adversário identifique os quadros de tempos das sucessivas multiplicações modulares (T_{1-8}) adquiridas da arquitetura multiplexada inserida no algoritmo *left-to-right square-and-multiply*. Assim, seleciona-se facilmente cada multiplicação modular pelo truncamento de pontos e sabe-se que as três primeiras execuções são: *Square*(S)-*Multiply*(M)-*Square*(S).

Pelo método por janela deslizante, escolhe-se a execução referente à T2 (por conveniência) como sendo a referência e subtrai-se esse pontos truncados das janelas de pontos referentes à execuções desconhecidas ($T_{4,5,...}$). A Figura 40 ilustra os resultados da análise SDPA por janela deslizante sobre a arquitetura multiplexada para os tamanhos de palavras de 8, 16 e 32 *bits*.

Para os três tamanhos de palavras considerados, as diferenças nas subtrações T2-T5 e T2-T7 são bastante perceptíveis. Portanto, o tipo de execução representado por T2 (multiplicação

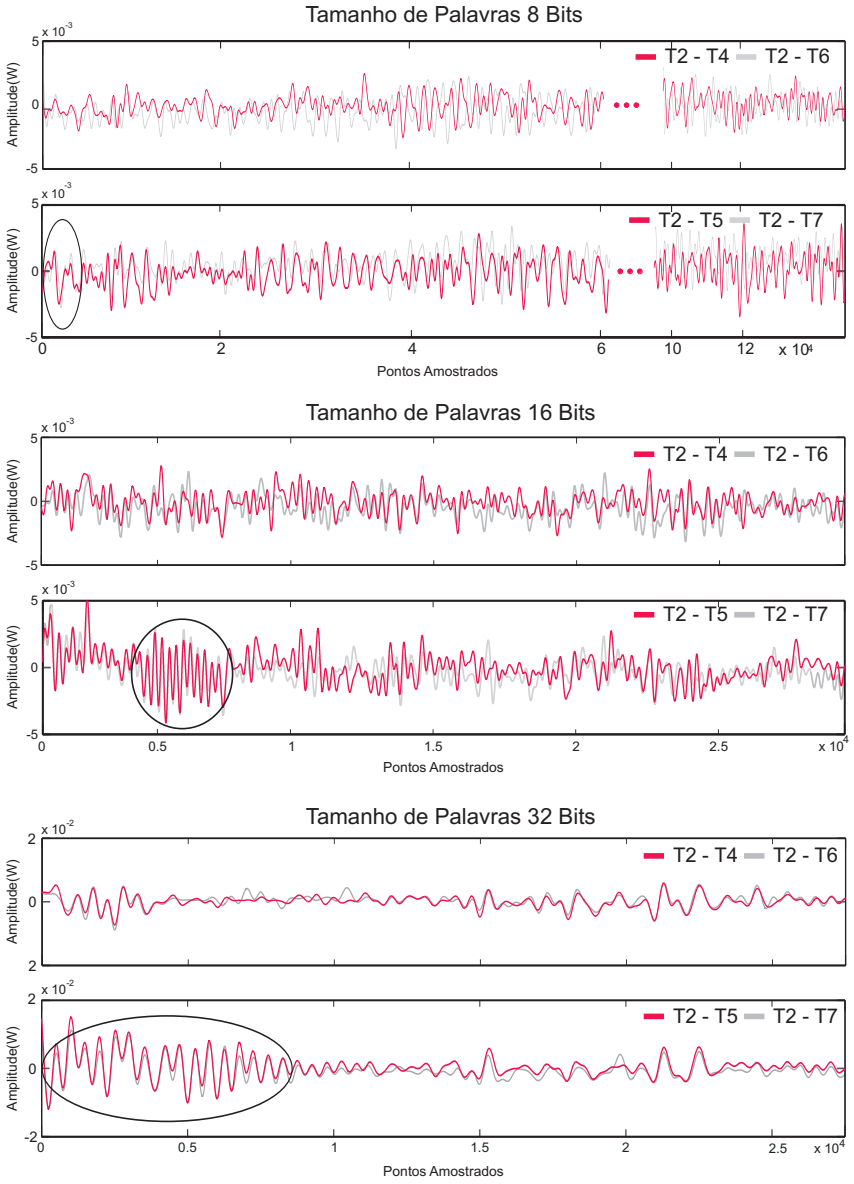


Figura 40: Ataque SDPA sobre o algoritmo *left-to-right square-and-multiply*.

ou quadrado) é diferente da execução representada por T5 e T7. Assim, como T2 é uma multiplicação modular, T5 e T7 são quadrados modulares e T4 e T6 devem ser multiplicações, uma vez que as diferenças T2-T4 e T2-T6 são insignificantes. Para o tamanho de palavra 8 *bits*, as diferenças são menos evidentes.

É importante salientar que as diferenças entre as operações de quadrado e multiplicação modulares aparecem sempre sobre os pontos que referem-se a parte de controle da arquitetura multiplexada, mais precisamente em momentos onde a arquitetura realiza leituras de operandos nas memórias RAM e também seleciona entradas e saídas dos multiplexadores. Os pontos restantes das curvas apresentam diferenças pouco significativas e pode-se concluir que os elementos da arquitetura que realizam os cálculos das iterações do algoritmo de Montgomery (bloco

quociente, *cores* aritméticos) não apresentam fugas de informações confidenciais.

A Figura 41 apresenta os resultados do mesmo ataque SDPA sobre uma implementação com o algoritmo *square-and-multiply always*. Para o traço analisado, foram truncados oito execuções de multiplicações modulares (T_{1-8}) e três tamanhos de palavras distintos foram aplicados: 8, 16 e 32 *bits*.

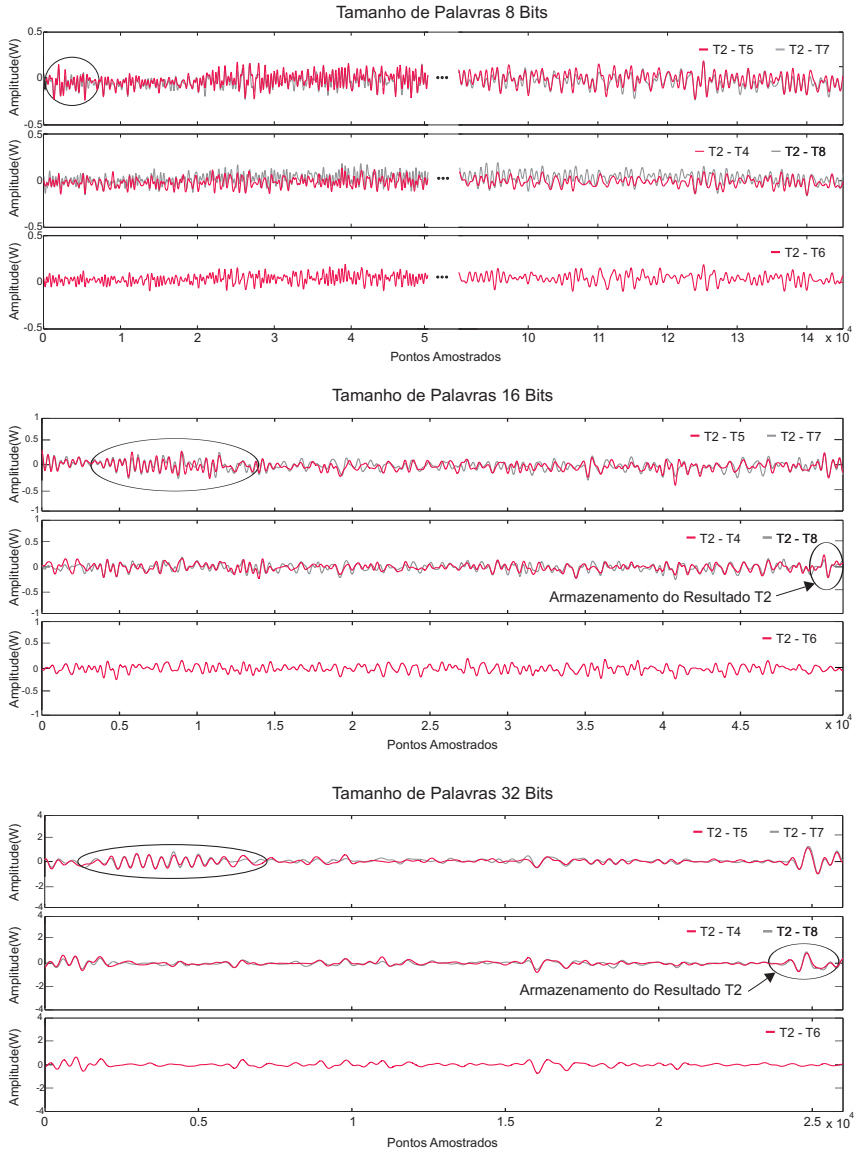


Figura 41: Ataque SDPA sobre o algoritmo *square-and-multiply always*.

As multiplicações "falsas"do algoritmo *square-and-multiply always* podem ser reveladas com a análise por demodulação. Após a localização das execuções de multiplicação e quadrado modular, observa-se que, ao final do cálculo das multiplicações referentes à T4 e T8, há picos de amplitudes, que indicam que para estas execuções não armazena-se os resultados em memórias RAM. Assim, T4 e T8 são as multiplicações "falsas". No entanto, para o tamanho de palavras de 8 bits, esses picos de amplitude não são localizados.

5.5 Interpretações

Os ataques simples por análise do consumo de potência e emissão eletromagnética realizados sobre a arquitetura padrão RSA evidenciam que esta implementação é extremamente vulnerável, uma vez que os *bits* do expoente foram recuperados com um único traço em uma análise visual. Por outro lado, a arquitetura multiplexada foi projetada para que o mesmo bloco de *hardware* realizasse tanto operações de quadrado modular quanto de multiplicação modular. Desse modo, a arquitetura multiplexada é protegida contra análise simples, pois através de uma análise visual não é possível discernir se o *bit* do expoente é zero ou um.

Apesar disso, quando o ataque torna-se mais robusto, como é o caso dos ataques SDPA e SDEMA, percebe-se que existem certas diferenças entre execuções de quadrado e multiplicação modulares. No entanto, as maiores diferenças são visíveis apenas quando o controle da arquitetura multiplexada realiza operações de leitura e escrita nas memórias RAM A e M (as quais armazenam a mensagem *m* e os resultados intermediários de multiplicações modulares) e a seleção de entradas e saídas de multiplexadores. Outra constatação é que quanto maior o tamanho de palavra, maior é a fuga de informações. Como resultado, a arquitetura multiplexada inserida no algoritmo *square-and-multiply always* e configurada com tamanho de palavras de 8 *bits* é robusta aos ataques SDPA e SDEMA.

O comportamento das curvas de consumo de potência e de emissões eletromagnéticas sobre os pontos que representam os cálculos aritméticos das iterações da multiplicação modular de Montgomery não apresentam fugas de informações e, desse modo, não apresentam fraquezas frente análise por demodulação de traços de consumo de potência e emissão eletromagnética.

A análise eletromagnética mostrou-se mais potente que a análise por consumo de potência. Os traços de emissão eletromagnéticas demodulados revelaram mais precisamente a informação confidencial com uma quantidade relativamente menor de traços, uma vez que a quantidade de ruído em um traço de emissão eletromagnética é menor que em traços que representam o consumo de potência. Para realizar o ataque SDEMA, calculou-se a média para apenas 20 traços coletados. Por outro lado, para o ataque SDPA, 200 traços foram necessários para realizar um traço médio de consumo de potência que permitisse revelar os *bits* do expoente. Além disso, a análise eletromagnética permite encontrar locais ou coordenadas sobre a superfície do circuito integrado onde a fuga de informações é mais intensa.

5.6 Contra-medidas

Uma possível contra-medida para evitar a fuga de informações na parte de controle da arquitetura é através da randomização do acesso às memórias RAM. Assim, os operandos A e B são lidos de posições randômicas de cada memória RAM. No entanto, para uma operação de quadrado modular, ($A.A \ mod \ N$) somente o operando A deve ser lido, ao contrário quando os operandos A e B são lidos para uma multiplicação modular ($A.B \ mod \ N$). A solução seria colocar duas memórias RAM: a primeira armazenando apenas o operando A e os resultados intermediários enquanto outra memória RAM armazenaria ambos A e B. Assim, tanto em uma operação de quadrado modular quanto de multiplicação modular as duas memórias serão acessadas, pois para o quadrado modular o operando A será lido das duas memórias.

Mesmo que a arquitetura multiplexada torne-se robusta contra análise simples, ataques de maior complexidade, como é o caso dos ataques DPA/DEMA, CPA/CEMA e ataques por mensagem escolhida (*chosen-message attacks* (Homma,2010)) poderiam ainda revelar os *bits* do expoente. No entanto, uma possível contra-medida frente à estes ataques é a randomização da mensagem de entrada e do expoente, conforme discutido no Capítulo 2.

6 CONCLUSÕES

Este trabalho apresentou o projeto de duas arquiteturas para multiplicação modular de Montgomery, sendo a primeira implementada em matriz sistólica e a segunda uma proposta de multiplexação dos processos aritméticos. Ambas as arquiteturas foram sintetizadas para as famílias de FPGAs Virtex-4 e Virtex-5. Apresentou-se uma análise de desempenho, através do tempo de execução para uma multiplicação modular de Montgomery e da aplicação das arquiteturas em um processo de exponenciação modular, a principal operação do algoritmo de criptografia de chave pública RSA.

A arquitetura sistólica apresentou melhores resultados em termos de tempo de execução, uma vez que apresenta maior grau de paralelismo que a arquitetura multiplexada. Esta, por sua vez, apresentou uma grande reutilização de elementos aritméticos, o que aumenta sua flexibilidade. Assim, a maior latência apresentada pela arquitetura multiplexada é compensada pela diminuição de utilização de recursos lógicos do FPGA.

Este trabalho também apresentou uma análise de robustez da arquitetura multiplexada frente à ataques por canais laterais simples SPA e SEMA. A validação destes ataques foi feita sobre uma arquitetura RSA padrão, que não apresenta nenhuma contra-medida a nível de algoritmo ou *hardware*. A arquitetura multiplexada provou ser robusta frente aos ataques SPA e SEMA, apresentando um comportamento de arquitetura protegida contra análise simples. No entanto, através da demodulação em amplitude dos traços de consumo de potência e emissão eletromagnética coletados (ataques SDPA e SDEMA), enfatizou-se os pontos fracos da arquitetura multiplexada, no caso, a parte de controle (multiplexadores, memórias RAM), mesmo na presença de uma contra-medida contra análise simples, como foi o caso do algoritmo *square-and-multiply always*. Ficou claro que o tamanho de palavra empregado na arquitetura influencia em aspectos de robustez, pois os ataques SDPA e SDEMA não foram capazes de revelar os *bits* do expoente para tamanho de palavras de 8 *bits*, quando a implementação era protegida.

A análise eletromagnética mostrou-se mais potente que a análise por consumo de potência, pois o número de traços necessários para revelar os *bits* do expoente através do método por janela deslizante foi consideravelmente menor para a análise SDEMA.

6.1 Trabalhos Futuros

- Implementação das contra-medidas propostas no capítulo anterior, principalmente a contra-medida relacionada aos acessos nas memórias RAM;
- Implementação dos ataques SDEMA e SDPA sobre o algoritmo de exponenciação modular Montgomery Powering Ladder (Joye,2003);
- Melhoria dos ataques SDEMA e SDPA através da localização de bandas de frequências nas quais localiza-se a informação confidencial. Através de técnicas de filtragem e demodulação em amplitude dos traços de consumo de potência e emissão eletromagnética pode-se recuperar mais claramente a informação confidencial;
- Implementação de randomização de mensagem e expoente, de modo a evitar ataques por mensagem escolhida (*chosen-message attacks*).

REFERÊNCIAS BIBLIOGRÁFICAS

Advanced Encryption Standard, 2001. *FIPS PUB 197*.

Agrawal, D., Archambeault, B., J. Rao., and Rohatgi, P. The EM Side-Channel(s). In *CHES 2002*, pages 29–45.

Amiel, F., Feix, B. and Villegas, K. Power Analysis for Secret Recovering and Reverse Engineering of Public Key Algorithms. In *CHES 2007*, pages 110–125.

Blum, T., Paar, C. High-Radix Montgomery Modular Multiplication on Reconfigurable Hardware. In *IEEE Transactions on Computers, 2001*, 50:759–764.

Brier, E., Clavier, C., and Olivier, F. Correlation Power Analysis With a Leakage Model. In *CHES 2004*, pages 16–29.

Coron, J.-S. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In *CHES 1999*, volume 1717, pages 292–302.

Dehbaoui, A., Lomne, V., Maurine, P. and Torres, L. Magnitude Squared Incoherence EM Analysis for Integrated Cryptographic Module Localisation. *Electronic Letters 2009*, vol.45:778–780.

A. Dehbaoui, V. Lomne, P. Maurine, L. Torres and M. Robert. Enhancing Electromagnetic Attacks Using Spectral Coherence Based Cartography. *International Conference on Very Large Scale Integration 2009*.

Data Encryption Standard, 1999. *FIPS PUB 46-3*.

Diffie, W. and Hellman, M. New Directions in Cryptography. *Information Theory 1976, IEEE Transactions on*, 22(6):644 – 654.

Gandolfi, K., Mourtel, C., and Olivier, F. Electromagnetic Analysis: Concrete Results. In *CHES 2001*, pages 251–261.

Homma, N., Miyamoto, A., Aoki, T., Satoh, A., and Shamir, A. Comparative Power Analysis of Modular Exponentiation Algorithms. *IEEE Transactions on Computers 2010*, vol. 59, pages 795–807.

Huang, M., Gaj, K., Kwon, El-Ghazawi, T. An Optimized Hardware Architecture for the Montgomery Multiplication Algorithm. *Public-Key Cryptography 2008*, pages 214–228.

Joye, M. and Yen, S. The Montgomery Powering Ladder. In *CHES 2003*, pages 291–302.

Joye, M., Paillier, P., and Schoenmakers, B. On Second-Order Differential Power Analysis. In *CHES 2005* pages 293–308.

Joye, M. Highly Regular Right-to-Left Algorithms for Scalar Multiplication. In *CHES 2007*, pages 135–147.

Koblitz, N. Elliptic curve cryptosystems. In *Mathematics of Computation 1987*, Volume 48, pages 203–209.

Kocher, P. C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996*, pages 104–113.

Kocher, P. C., Jaffe, J., and Jun, B. Differential Power Analysis. In *CRYPTO 1999*, pages 388–397.

Koç, Ç. K., Acar, T., and Kaliski Jr, B.S. Analyzing and Comparing Montgomery Multiplication Algorithms. In *IEEE Micro 1996*, 16:23–33.

Kung, H. T. and Leiserson, C.E. Algorithms for VLSI Processor Arrays. In *Introduction to VLSI Systems, 1978*, Addison-Wesley, USA.

Mamiya, H., Miyaji, A., and Morimoto, H. Efficient Countermeasures Against RPA, DPA, and SPA. *CHES 2004*, pages 243–319.

McIvor, C., McLoone, M., and McCanny, J. High-Radix Systolic Modular Multiplication on Reconfigurable Hardware. In *FPT 2005*, pages 13–18.

Menezes, A. *Handbook of Applied Cryptography - fifth printing*. CRC Press, Boca Raton, 2001, USA.

Messerges, T. S., Dabbish, E. A., and Sloan, R. H. Power Analysis Attacks of Modular Exponentiation in Smartcards. *CHES 1999*,pages 144–157.

Michalski, E. A., Buell, D. A. A scalable architecture for Cryptography on large FPGAs. *16th International Conference on Field Programmable Logic and Applications 2006*, 145–152.

Montgomery, P. L. Modular Multiplication Without Trial Division. *Mathematics of Computation 1985*, 44:519–521.

Standard RSA Cryptocore, 2010. http://www.opencores.com.

Orup, H. Simplifying Quotient Determination in High-Radix Modular Multiplication. In *Proceedings of ARITH-12 1995*, pages 193–199, Bath, England. IEEE Computer Society.

Quisquater, J.J, and Samyde, D. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. *E-smart 2001*, 2140.

Rabaey, J.M,. *Digital Integrated Circuits*, 1996. Prentice Hall International.

Réal, D., Valette, F., and Drissi, M. Enhancing Correlation Electromagnetic Attack Using Planar Near-Field Cartography. In *DATE 2009*, pages 628–633.

R.L Rivest, A. Shamir, L. A. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM 1978*, 21:120–126.

Sauvage, L., Guilley, S., and Mathieu, Y. Electromagnetic Radiations of FPGAs: High Spatial Resolution Cartography and Attack on a Cryptographic Module. *TRETS 2009*.

Stremler, Ferrel G. *Introduction to Communication Systems*, 1982. Addison-Wesley, University of Wiscosin, Madison.

Tenca, A. F., Koç, C. K. A Scalable Architecture for Montgomery Multiplication. *CHES 1999*, 94–108.

Tenca, A. F., Todorov, G., Koç, C. K. High-Radix Design of a Scalable Modular Multiplier.. *CHES 2001*, 185–201.

Tenca, A. F., Koç, C. K. A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm. *IEEE Transactions on Computers 2003*, 52:1215–1221.

Walter, C. D. Montgomery Exponentiation Needs no Final Subtractions. *IEEE Electronics Letters 1999*, 35:1831–1832.

Virtex-2 Datasheet, 2007 http://www.xilinx.com.

Spartan-3E Starter Kit, 2008 http://www.xilinx.com.

Virtex-4 and Virtex-5 Families, 2010 http://www.xilinx.com.

ANEXO A – ARTIGOS PUBLICADOS

A.1 Artigos publicados

PERIN, G.; MESQUITA, D.; HERRMANN, F.L.; MARTINS, J. B. S **Montgomery Modular Multiplication on Reconfigurable Hardware:Fully Systolic Array vs Parallel Implementation**. In: Southern Programmable Logic Conference, 2010, Ipojuca,PE - Brazil.

PERIN, G.; MESQUITA, D.; HERRMANN, F.L.; MARTINS, J. B. S **An Efficient Implementation of Montgomery Powering Ladder on Reconfigurable Hardware**. In: XXIII Symposium on Integrated Circuits and System Design - SBCCI, 2010, São Paulo, SP - Brazil.

PERIN, G.; MESQUITA, D.; MARTINS, J. B. S **Montgomery Modular Multiplication on Reconfigurable Hardware: Systolic vs Multiplexed Implementation**. In: International Journal of Reconfigurable Computing, 2011.

A.2 Artigo Publicado na IJRC (*International Journal on Reconfigurable Computing*, 2011)

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2011, Article ID 127147, 10 pages
doi:10.1155/2011/127147

Research Article

Montgomery Modular Multiplication on Reconfigurable Hardware: Systolic versus Multiplexed Implementation

Guilherme Perin,[1] Daniel Gomes Mesquita,[2] and João Baptista Martins[1]

[1] *Grupo de Microeletrônica (GMICRO), Universidade Federal de Santa Maria (UFSM), Rio Grande do Sul, Santa Maria 97105-900, RS, Brazil*
[2] *Arquiteturas, Sistemas Operacionais e Sistemas Distribuídos, Grupo de Redes (GRASS), Universidade Federal de Uberlândia (UFU), Uberlândia 38401-136, MG, Brazil*

Correspondence should be addressed to Guilherme Perin, guilhermeperin@ymail.com

Received 2 June 2010; Accepted 30 November 2010

Academic Editor: Elías Todorovich

This paper describes a comparison of two Montgomery modular multiplication architectures: a systolic and a multiplexed. Both implementations target FPGA devices. The modular multiplication is employed in modular exponentiation processes, which are the most important operations of some public-key cryptographic algorithms, including the most popular of them, the RSA. The proposed systolic architecture presents a high-radix implementation with a one-dimensional array of Processing Elements. The multiplexed implementation is a new alternative and is composed of multiplier blocks in parallel with the new simplified Processing Elements, and it provides a pipelined operation mode. We compare the *time* × *area* efficiency for both architectures as well as an RSA application. The systolic implementation can run the 1024 bits RSA decryption process in just 3.23 ms, and the multiplexed architecture executes the same operation in 4.36 ms, but the second approach saves up to 28% of logical resources. These results are competitive with the state-of-the-art performance.

1. Introduction

Modular multiplication is widely employed in public-key cryptography, especially where modular exponentiation is essential. For instance, the most commonly used asymmetric cryptographic algorithm is the RSA [1]. The RSA security depends on the difficulty of factoring large numbers. Here, large numbers mean prime numbers of up to 4096 bits, used as cryptographic keys.

In this cryptosystem the main operation is the modular exponentiation using the public and private keys, the first to encrypt and the second to decrypt messages. So, the performance of the whole system depends on the efficiency of modular arithmetic implementations.

As modular operations are time consuming, it is common to use hardware devices to perform both the modular multiplication and the exponentiation. Among the hardware approaches, the increased use of reconfigurable devices to implement cryptographic operations, especially the FPGAs, is evident.

One of the most suitable methods for performing modular multiplications in hardware is the Montgomery multiplication [2]. This algorithm is fast and power efficient in hardware implementations. Assuming the modular multiplication as $A \cdot B \bmod N$, the Montgomery multiplication avoids the division by N by replacing the division by right shifts. Also, this method allows the use of multi-precision arithmetic, which is useful for employing high-radix operations. High-radix operations in turn make it easier to develop modular multiplication architectures.

Aiming to implement RSA systems based on hardware, many authors proposed Montgomery multiplications in FPGAs [3–9]. Fully systolic architectures designed to speed up the modular multiplication have been presented. These architectures offer a Processing Elements (PEs) array where each PE performs arithmetic additions and multiplications

Require: $N = \sum_{i=0}^{m-1}(2^k)^i n_i$, $n_i \in \{0, 1, \ldots, 2^k - 1\}$
$B = \sum_{i=0}^{m-1}(2^k)^i b_i$, $b_i \in \{0, 1, \ldots, 2^k - 1\}$
$A = \sum_{i=0}^{m-1}(2^k)^i a_i$, $a_i \in \{0, 1, \ldots, 2^k - 1\}$, $R = (2^k)^m$
$A, B < 2N$; $N < R = 2^{km}$; $N' = -N^{-1} \bmod (2^k)$.
return $S_{i+1} = \mathrm{ABR}^{-1} \bmod N$
$S_0 = 0$
For $i = 0$ to $m - 1$ **do**
$q_i = ((S_0 + a_i \times b_0)N') \bmod (2^k)$
$S_{i+1} = (S_i + q_i \times N + a_i \times B)/2^k$
End for

Algorithm 1: Montgomery modular multiplication.

in a multiprecision context with carry propagation [10]. Depending on the word size (or radix) used, the architecture can employ a high number of Processing Elements, consequently increasing the needs of the logic elements (area) in FPGA implementations.

As a new alternative in terms of implementation, the execution of additions and multiplications can be multiplexed by a block positioned parallel to the Processing Elements. This can be done by inserting multiplexed multipliers in parallel with Processing Elements. Forcing a pipelined operation mode and using a high-radix architecture (16 or 32 bits), the multiplexed multipliers ensure the high speed performance provided by systolic architectures, with reduced arithmetic and logic elements and also minimal carry signals propagation.

This paper presents a trade-off between two proposed modular multiplication architectures: a systolic and very high-radix multiplexed implementation. Our approach uses a radix-16 and radix-32 in both implementations to speed up the processes and to match the resource usage of Virtex-4 and Virtex-5 Xilinx FPGA Series [11]. The proposed architectures show significant improvements compared to our previous work [12]. Systolic architecture provides more simplified Processing Elements in order to reduce the utilization of FPGA resources. The multiplexed implementation is arranged in arithmetic cores, which allow us to handle the quantity of Processing Elements and multiplier blocks. Our goal is to highlight that the small increase in the number of clock cycles needed due to multiplexed multipliers made up for the significant reduction in the use of logical and architectural arithmetic.

This paper is organized as follows: Section 2 presents the Montgomery modular multiplication algorithm. Section 3 discusses related state-of-the-art works. The proposed architectures are presented in Section 4. Finally, the results and conclusion are presented in Sections 5 and 6, respectively.

2. Montgomery Modular Multiplication

The Montgomery Multiplication Algorithm is a method of performing modular multiplication $A \cdot B \bmod N$ without needing to divide by N. In cryptography, the Montgomery Algorithm is very suitable for the hardware implementation of modular multiplication, because it allows long integer numbers to be represented in a numeric precision given by a radix (generally a power of two).

The algorithm version used in this work is the original one, with some preconditions. Algorithm 1 shows the modular multiplication with the notation proposed on [13], and used for the remainder of this text.

The N' value is the modular inverse of N regarding the N modulus, computed so that $N \cdot N' = 1 \bmod N$. The final result is placed on S, after m iterations, and is equal to $A \cdot B \cdot R^{-1} \bmod N$, which must be corrected to retrieve the expected result ($A \cdot B \bmod N$). The correction is done by performing an additional Montgomery multiplication with S and $R^2 \bmod N$ as parameters. It is interesting to highlight that this correction is inexpensive during a modular exponentiation, because it only needs to be made one time after the whole exponentiation.

Since its publication in 1985 by Montgomery [2], the Montgomery Algorithm has undergone many modifications and improvements [14, 15]. One of those is particularly interesting, because it avoids the final subtraction simply by choosing the input data correctly. By limiting the operands A and B to integers less than $2N$ and by defining $2N$ as less than 2^{km}, the final S is guaranteed to be less than N [15]. These pre-conditions are shown in Algorithm 1 and applied to our architecture, as explained in Section 4.

3. Related Works

Tenca and Koç are widely referenced for their work on radix-2 Montgomery Algorithm implementations. These authors initially proposed architectures with improvements for the radix-2 Montgomery Algorithm, like in [16]. Even though the input operands are large numbers, radix-2 modular multiplications avoid expensive multiplications, which are visible on high-radix implementations (8 or more). Different from the classic radix-2 Montgomery Algorithm [13], Tenca and Koç's modifications allow the scalable property for modular multiplication architecture, that is, their proposed Montgomery multiplier is able to work with any precision of the input operands. In terms of hardware implementation, there is a systolic array architecture composed of Processing Elements and control blocks for managing the I/O words of the architecture. Each Processing Element contains only a few logic elements, providing a reduced area and high clock frequency, when synthesized for FPGA or ASIC.

Based on the above work, in [4, 17] improvements are presented to the Tenca and Koç proposition. The advantage of these new approaches is concentrated in the Processing Elements optimizations and, consequently, in the reduced latency of the Montgomery modular multiplications by a minimum factor of two, that is, the modular multiplication is twice as fast than [16]. So, the main contributions are in the modular multiplication speed improvement, and in the reduced number of logical elements for the Processing Elements. In [4], a radix-4 scalable Montgomery modular multiplication architecture is proposed to enhance the speed. Despite improvements in speed, these radix-2 and radix-4 architectures are still limited by the large number of clock cycles required.

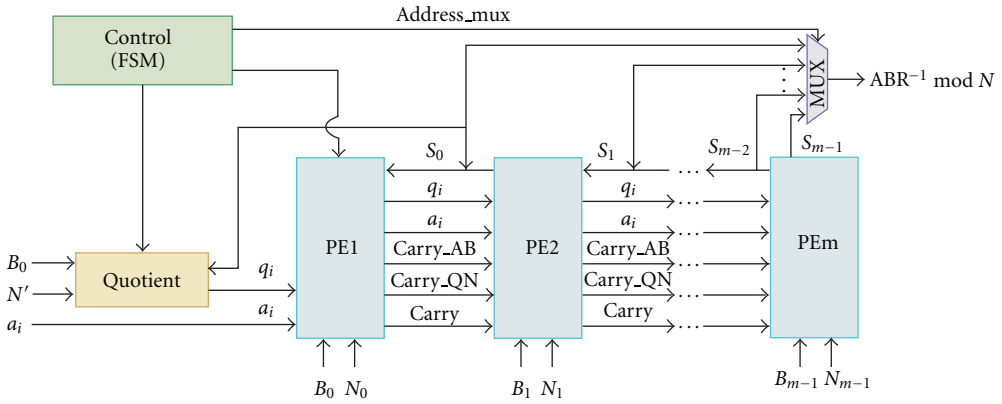


Figure 1: Systolic Architecture.

Furthermore, in the context of high-radix implementations, a systolic architecture is presented in [3] which is composed of Processing Elements able to provide modular multiplication for a radix greater than 4. Despite its time and area efficiency, this architecture requires preprocessing before the modular multiplication execution. The authors make use of the optimized Montgomery algorithm initially proposed in [14], which presented a way to simplify the q_i quotient calculus, making the quotient determine a simple truncation operation $S \bmod 2^k$. However, as a consequence, the input operands must meet the following limitations: $N' = -N^{-1} \bmod 2^k = 1$ and $A, B < 2(N' \bmod 2^k) \cdot N$, and the optimized Montgomery Algorithm will need three additional iterations, because the B input operand is left shifted by 2^k and has to be corrected with these further iterations.

To avoid preprocessing in a high-radix modular multiplication, [5] presents a fully systolic array architecture composed of Processing elements containing internal multipliers and adders. The Montgomery algorithm version used in this implementation is also the optimized version proposed in [14]. As an implementation in radix-16, the modular multiplications take only 103 clock cycles, significantly less than other architectures [3, 16, 17].

4. The Proposed Architectures

The proposed architectures for performing Montgomery modular multiplication are detailed in this section. First, the systolic architecture is described in detail as well as the Processing Elements behaviour. Second, the multiplexed and systolic Montgomery modular multiplication architecture is presented.

4.1. The Systolic Architecture. The concept of systolic architecture combines a highly parallel array of identical Processing Elements or data-paths with local connections, which take external inputs and process them in a predetermined manner and in a pipelined fashion.

The proposed systolic architecture is directly based on the arithmetic operations of the Montgomery Algorithm, which are performed in a numerical base 2^k, in which the large input operands are processed in a multi-precision context containing m words of k bits. As seen in Section 2, the Montgomery Algorithm has additions and multiplications involving large integers that make use of multiple-precision arithmetic.

The architecture is composed of m Processing Elements distributed in a one-dimensional array, where each Processing Element is responsible for the calculus involving k bits words of the input operands with the same index of the Processing Element. For example, for a 1024 bits modular multiplication with radix-32, the operands are split in 32 words of 32 bits which results in a one-dimensional array of 32 Processing Elements.

Between the Processing Elements, there is a propagation of carry signals which are the most significant bits of the arithmetic processes in each PE. The carry signals are processed as input parameters by the Processing Elements that receive them.

In the systolic architecture, the Processing Elements are designed by finite state machines. The control block communicates with the first Processing Element (PE1) and with the block responsible for the quotient calculation $q_i = (S_0 + a_i \cdot b_0)N' \bmod 2^k$, according to line 4 of the Montgomery Algorithm. Figure 1 presents the systolic architecture.

The finite state machine structure of the control block is designed to provide the required words for a modular multiplication to the Processing Elements and to the quotient block. Thus, at each Montgomery Algorithm iteration, these words are read from an external RAM memory and passed to the remaining architecture. At the end of the modular multiplication, the control block provides the Montgomery multiplication result $A \cdot B \cdot R^{-1} \bmod N$ through an output multiplexer.

The one-dimensional array of Processing Elements performs the calculation of $S_{i+1} = (S_i + q_i \times N + a_i \times B)/2^k$, according to the Montgomery Algorithm. In this operation, there are two multiplications between an input operand and a k bits word, and after the addition between the result of these two multiplications. Therefore, the systolic architecture works in a multi-precision context, and each Processing Element is responsible for performing the

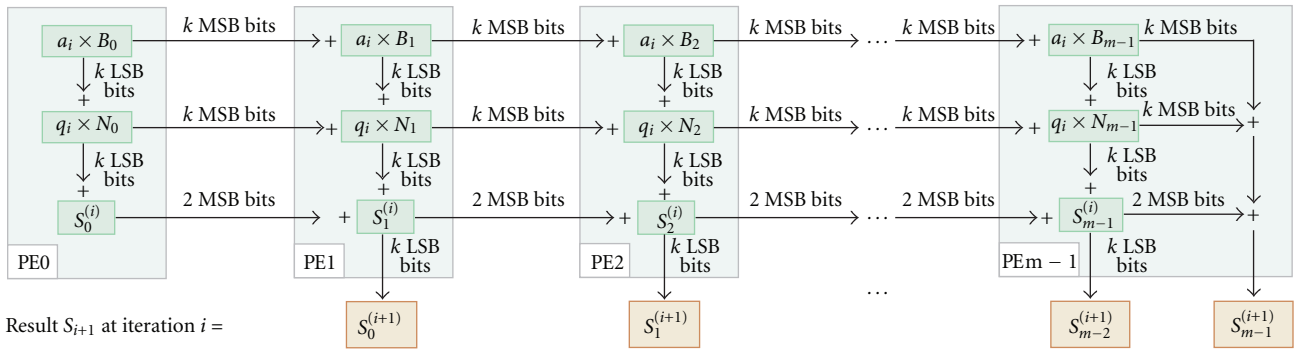


Figure 2: Arithmetic operations of the Processing Elements to obtain $S_i + 1 = (S_i + q_i \times N + a_i \times B)/2^k$.

arithmetic operations involving one word of each input operand. Thus, the number of words of each operand is equal to the number of Processing Elements. Figure 2 shows the arithmetic operations flowchart within each processing element.

According to Figure 2, the multiplication between a_i and B_i words returns a $2k$ bits result, where the least significant bits of this multiplication are added to the least significant bits of the $q_i \times N_i$ multiplication result. Finally, the least significant bits of this add are also added to a k bits word of the S result of the previous iteration. The carry signals propagated to the next Processing Element are the most significant bits of the two multiplications and the most significant bits of the last addition.

4.1.1. First Processing Element. The first Processing Element (PE1) establishes communication with the control block and receives a_i and q_i words at each Montgomery Algorithm iteration. This PE differs from the other Processing Elements because it does not receive any carry signal as input and it discards the first word of the S result, which means the division of $S_{i+1} = (S_i + q_i \times N + a_i \times B)$ by 2^k. The zero index words of B and N (N_0 and B_0) are also provided to this first Processing Element. The internal architecture of PE1 is shown in Figure 3.

4.1.2. General Processing Element. The other Processing Elements are different from PE1 because they have a word from the S result as output and they also transmit and receive carry signals of the multi-precision multiplications and additions. Each Processing Element is activated by the previous Processing Element when the latter finishes its calculation and sends out its carry signals, which means that the architecture works with a pipeline behaviour. Only the last Processing Element provides two words of the S result as a response at each iteration of Algorithm 1 because the S^{m-1} word is obtained with a sum of carry signals. By avoiding a new Processing Element instantiation juts to perform this sum, it is calculated in the last Processing Element. Figure 4 presents the internal architecture of the general Processing Elements.

4.1.3. Quotient Block. At each iteration of Algorithm 1, line 3 presents the q_i quotient computation so that $S + a_i * B + q_i * N$ becomes a multiple of 2^k. The internal architecture of the quotient block is shown in Figure 5. This structure has a combinational behaviour where the q_i result is obtained in one clock cycle. S_0, a_i, B_0, and N' are k bits words which are provided for this block at each iteration of Algorithm 1.

The zero index of B and S means that these words contain the k least significant bits (LSBs) of B and S operands, respectively. As we can see in the right side of Figure 5, a multiplication between a_i and b_0 will provide a $2k$ bits result. Just the LSB part of this result is used in the next operation. Another input of the quotient block, S_0, is then added to the LSB part obtained from the first multiplication. Again, we only need the LSB part of this addition, which is finally multiplied by N', which corresponds to the modular inverse of N modulo 2^k. The LSB part of this last multiplication is the q_i desired result. As seen in Algorithm 1, the numerical basis is power of 2, so for hardware architecture, the mod 2^k operation is simply performed by a right shift operation (LSB selection).

So, the complexity of the quotient block relies on two single precision multiplications and one single precision addition. To evaluate the number of clock cycles for a modular multiplication, we have to consider the first m cycles to read the A and B operands from RAM memories for a square or modular multiplication, respectively. The first iteration of Algorithm 1 also needs m clock cycles. The remaining iterations of Algorithm 1 are performed in $4 * m$ clock cycles.

4.2. The Multiplexed Systolic Architecture. As seen in the previous section, the systolic architecture presents a one-dimensional array of Processing Elements, and each PE is responsible for operations of addition and multiplication. When the numerical basis (2^k) is high (2^{16}, 2^{32}), the internal multiplications become more complex, mainly if the design is applied to an FPGA or an ASIC. So, as the number of multipliers increases, the physical limitations will increase proportionally, for example, in the maximum clock frequency, area, (etc.).

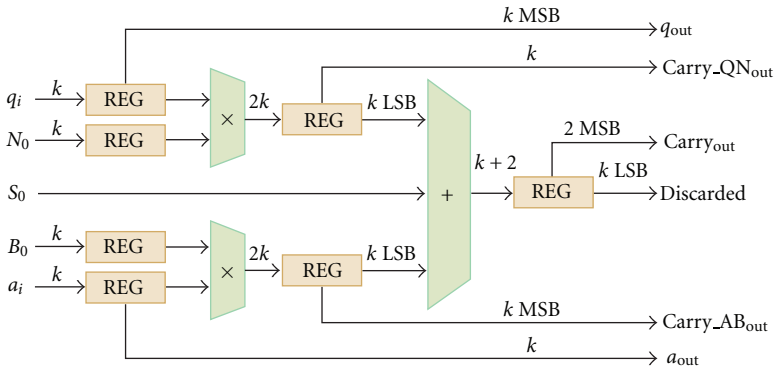


Figure 3: First Processing Element internal architecture.

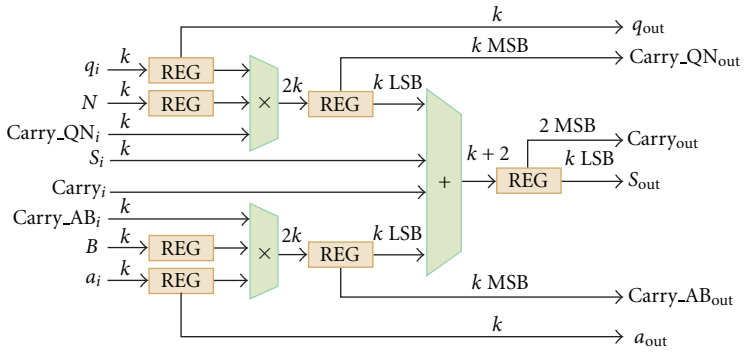


Figure 4: General processing element internal architecture.

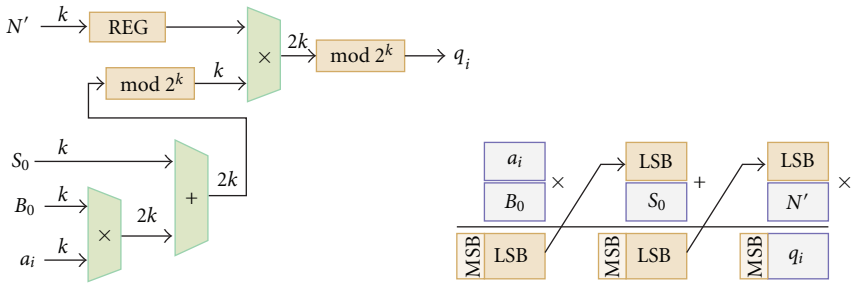


Figure 5: Internal architecture of the quotient block.

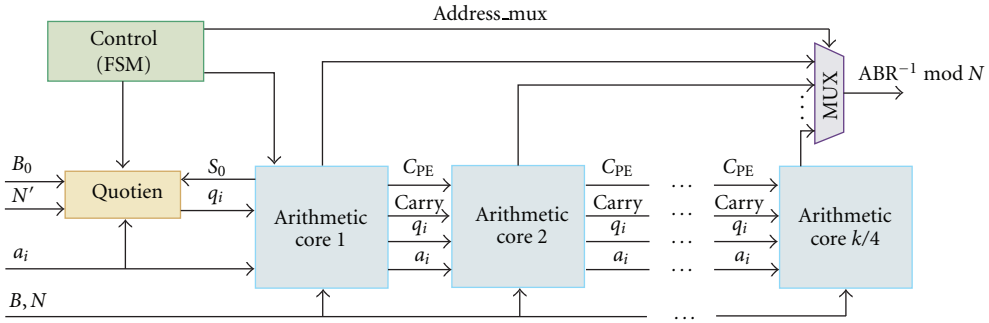


Figure 6: Proposed multiplexed modular multiplication architecture.

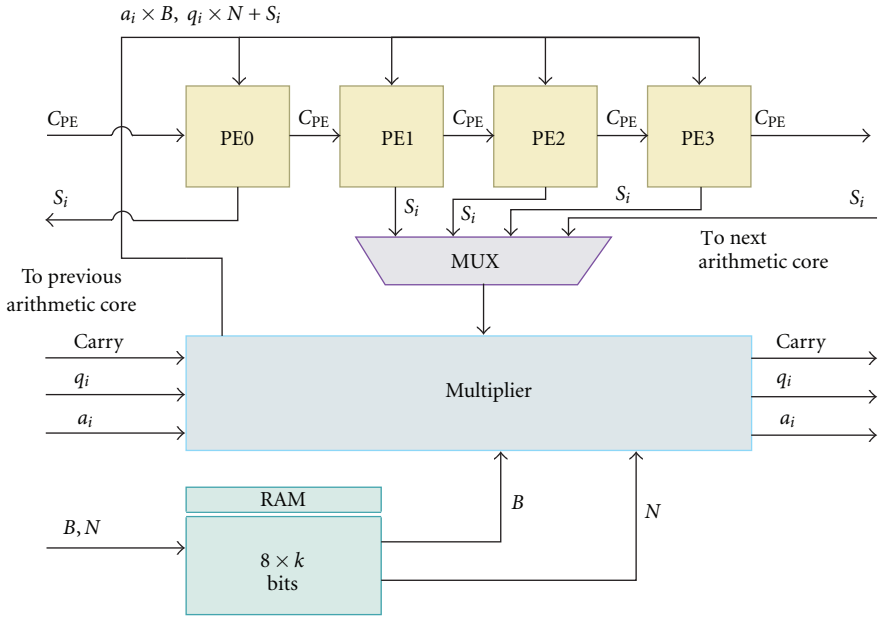


Figure 7: The Arithmetic core architecture.

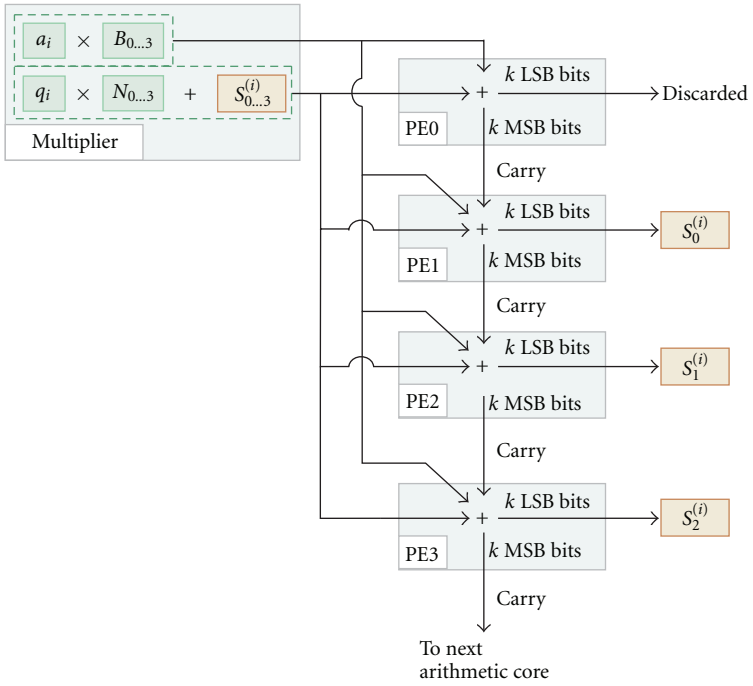


Figure 8: Arithmetic operations performed in arithmetic core 1.

Based on these constraints, a multiplexed and systolic architecture with multiplier blocks working parallel to the Processing Elements is presented in this section. It provides a migration of $k \times k$ bits multipliers from the Processing Elements to the multipliers blocks. Each multiplier block, together with the four Processing Elements, forms an arithmetic core. The one-dimensional arrangement of these arithmetic cores forms the structure of the modular multiplication architecture. Figures 6 and 7 show the multiplexed and systolic architecture and the arithmetic core structure, respectively.

The multiplexed architecture is composed of exactly $k/4$ arithmetic cores, and the first one is managed by a control block designed by a finite state machine. According to Figure 7, each arithmetic core contains four Processing Elements, a multiplier, and an $8 \times k$ bits RAM memory. Being a multi-precision arithmetic architecture, the number of Processing Elements is equivalent to the number of words

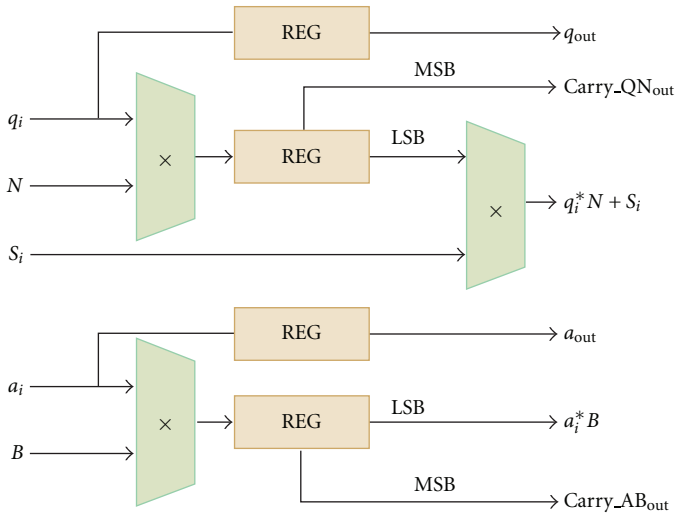


Figure 9: Multiplier block architecture.

in each input operand. So, the RAM memory placed in each arithmetic core stores four words of B and N operands.

The multiplier block performs the $q_i \times N_i$ and $a_i \times B_i$ multiplications. The least significant bits of $q_i \times N_i$ multiplication are added to a k bits word of previous S_i result. The least significant bits of this add operation and the least significant bits of the $a_i \times B_i$ multiplication are sent to the Processing Elements to be added. The Processing Elements provide the S words of the current iteration result. Figure 8 illustrates the executions performed by Arithmetic Core 1. By analysing this illustration, we can realize that instead of having two single precision multiplications in each Processing Element, there is a multiplier block that performs all single precision multiplications for a total of four Processing Elements. In other words, the quantity of single precision multiplications is reduced four times. With these improvements, each Processing Element needs to perform just one addition.

The calculation of the quotient q_i is performed by a block with architecture that is identical to that of the quotient block presented in the systolic architecture.

The Montgomery Algorithm's multiplications are made by a multiplier block that utilizes the multipliers available in the FPGA. The internal architecture of the multiplier blocks is shown in Figure 9.

The carry signals propagated inside the multiplexed architecture are the k most significant bits of the $q_i \cdot N$ and $S_i + a_i \cdot B$ operations presented in Algorithm 1 and are propagated between the multiplier blocks. The last multiplier block sends its carry signals to the fourth and final Processing Element placed in the last arithmetic core. The other carry signal, C_{PE}, is the most significant bits of the result of the addition between the $q_i \cdot N$ and $S_i + a_i \cdot B$ terms. This last addition is performed by the Processing Elements.

At the end of the $m - 1$ iteration, the $S_{i+1} = A \cdot B \cdot R^{-1} \bmod N$ is sent out by an $m : 1$ k bits multiplexer. This result is sent to the memory that is part of the modular exponentiation architecture (described in the next section).

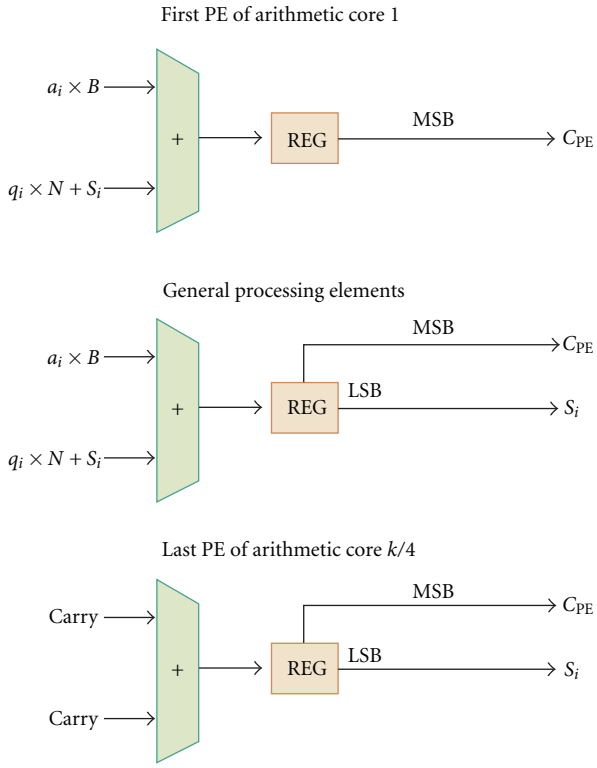


Figure 10: General PE with the carry propagation.

In terms of clock cycles for the Montgomery modular multiplication, we can define the following: initially, m clock cycles are reserved for B operand internal storage. This operand is read from RAM memories. Considering that the N modulus is already available on internal RAM memories placed in arithmetic cores, the first iteration also takes m clock cycles and, it takes the architecture $6 \times m$ clock cycles to perform the remaining iterations of Montgomery Algorithm.

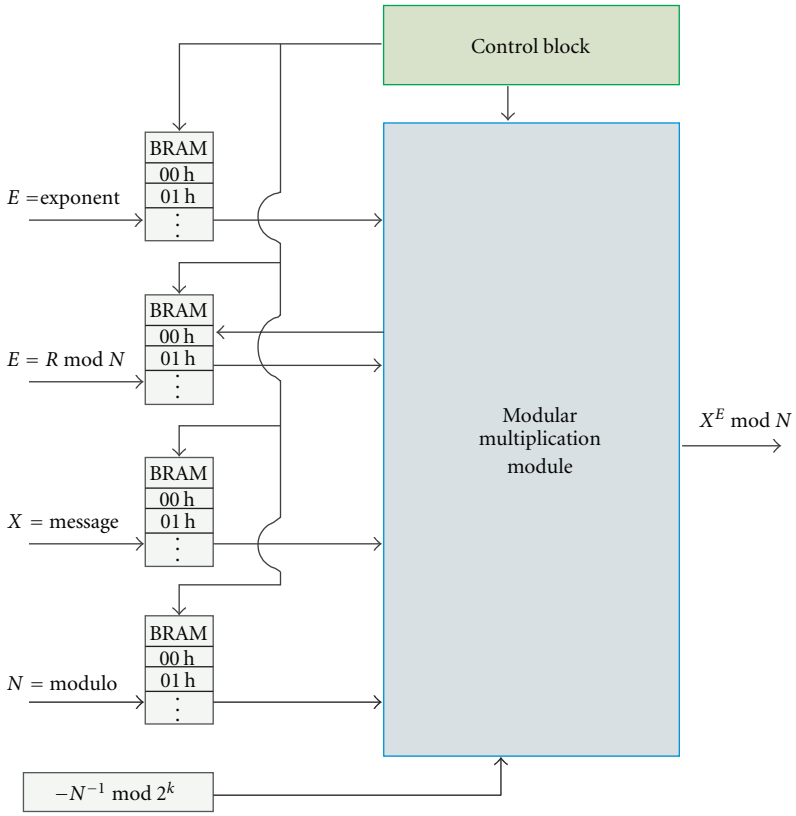


Figure 11: Modular exponentiation architecture.

Thus, the total number of clock cycles, for a modular (or squared) multiplication is $n_{\mathrm{MM}} = m + m + 6 \times m = 8m$.

4.2.1. The Processing Elements PEs. The proposed modular multiplication architecture is composed of m Processing Elements (where m is the number of words of the operands and also the number of iteration on Algorithm 1). Due to the placement of a multiplier block in each arithmetic core, each Processing Element needs to perform just one addition between two $2k$ bits words and sends out a word of S_{i+1} result at each iteration of the Algorithm 1. The first Processing Element must discard the least significant bits of its first addition in order to perform the right shift operation, which corresponds to the division of $(S_i + q_i \times N + a_i \times B)$ by 2^k.

The remaining Processing Elements perform the addition between $(S_i + a_i \times B)$ and $q_i \times N$ terms and the resultant k least significant bits word of this addition are sent out as a word of the S result. The $k + 1$ most significant bits are sent to the next Processing Element as a carry signal. The last Processing Element (PE_m) is responsible for providing two words of S result (S^{m-2} and S^{m-1}), considering that the input words for S^{m-1} calculus are the carry signals from the last multiplier block. Figure 10 shows the first, general case and the last Processing Elements.

4.3. Modular Exponentiation. For a real cryptographic application concerning the RSA algorithm, a modular exponentiation structure that incorporates the modular multiplication architecture is proposed in this section. The modular exponentiation algorithm used in this work is left-to-right square and multiply [13], and thus in average $1.5 * n$ modular multiplications (including squares and multiplies executions) are performed to achieve the final exponentiation result, which n is the operand's precision. Algorithm 2 shows the Montgomery modular exponentiation algorithm.

Require: $E = \sum_{i=0}^{n-1} e_i 2^i$, $e_i \in \{0, 1\}$.
retun: $A = X^E \bmod N$
$A = R \bmod (N)$
$\overline{X} = \mathrm{mont}(X, R^2 \bmod (N))$
for $i = n - 1$ to 0 **do**
$A = \mathrm{mont}(A, A)$
if $e_i = 1$ **then**
$A = \mathrm{mont}(A, \overline{X})$
end if
end for
$A = \mathrm{mont}(A, 1)$

Algorithm 2: Montgomery modular exponentiation—square and multiply.

Four Block RAM memories generated through *Xilinx Coregen* tool were placed to store the input operands of size n. These input operands are the N modulus, the E exponent, the message X in the Montgomery domain

Table 1: Proposed architectures synthesis.

Virtex-4						
n	k	Slices	Clock cycles	DSP48	Freq. (MHz)	BRAM (Bytes)
Systolic architecture						
512	16	3322	192	68	110	128
512	32	4199	96	36	78	128
1024	16	7012	384	130	110	256
Multiplexed architecture						
512	16	2199	256	32	120	256
512	32	2499	128	32	80	256
1024	16	4876	512	64	120	512
1024	32	5118	256	64	80	512
Virtex-5						
Systolic architecture						
512	16	3205	192	68	130	128
512	32	3876	96	36	95	128
1024	16	6642	384	130	130	256
Multiplexed architecture						
512	16	2078	256	32	120	256
512	32	2370	128	32	90	256
1024	16	4876	512	64	120	512
1024	32	5005	256	64	90	512

Table 2: RSA application (Virtex-5).

n	Freq. (MHz)	RSA decryption	Clock cycles
Systolic Architecture			
1024	130	3.23 ms	491520
Multiplexed Architecture			
1024	90	4.36 ms	393216

Table 3: State-of-art implementations of modular multiplication architectures.

Design	FPGA	Clock	Area	Mod exp
Systolic	XC5VLX110T	130 MHz	6642 Slices	3.23 ms
Multiplexed	XC5VLX110T	90 MHz	5005 Slices	4.36 ms
[5]	XV2VP70	101.86 MHz	5709 Slices	3.01 ms
[12]	XC5VLX110T	95 MHz	3044 Slices	6 ms
[4]	XC2V2000	248 MHz	4051 Slices	9.4 ms
[1]	Virtex-4	150.5 MHz	2613 Slices	13.94 ms

($X = X \cdot R \mod N$), and an auxiliary term $A = R \bmod N \cdot A$ control block with a finite state machine manages the read and write operations from the memories (see Figure 11).

The results of the successive modular multiplications are stored in the RAM memory that previously has stored the $A = R \bmod N$ operand, because this operand is necessary just in the first square execution.

5. Results

Table 1 summarizes the FPGA synthesis results of two proposed modular multiplication architectures. The designs were described in hardware description languages (VHDL and Verilog) and synthesized for Virtex-4 and Virtex-5 Xilinx FPGAs. All results are postimplementation, and no area or speed optimizations were set for the synthesis. The results presented in this paper are improvements when compared with our previous work [12]. The multiplexed architecture is implemented with a reduced number of slices registers and DSP48s. However, synthesis for the systolic architecture presented high clock frequencies.

Table 2 presents an RSA encryption and decryption applications of the proposed architectures. Since the modular exponentiation is performed by successive modular multiplication executions, the left-to-right (MSB) binary square and multiply algorithm was employed in the modular exponentiation. The results show that, considering the amount of clock cycles for a modular multiplication execution, the multiplexed architecture is faster than the systolic implementation. On the other hand, the systolic architecture has a clock frequency higher than the clock frequency presented by the multiplexed architecture.

Table 3 shows a state-of-art comparison with our results. Every work referred in this table used the Montgomery Algorithm for their hardware modular multiplication architectures, and for a direct comparison with our approaches just 1024 bits applications are exposed. The time of modular multiplications, when not explained in the references, are estimated considering a modular exponentiation of

$n = 1024$ bits through the Square and Multiply algorithm, running $1.5n$ modular multiplications.

6. Conclusion

This paper presented two Montgomery modular multiplication architectures and the results of their synthesis for Xilinx Virtex-4 and Virtex-5 FPGAs. A systolic implementation and a multiplexed implementation, suitable for RSA public-key cryptosystem, were developed, and the designs were carefully matched with features of the FPGAs, utilizing embedded DSP48Es Slices and Block RAM. The designs are improvements of a previous work. The multiplexed implementation presented a good performance considering *time* $\times$ *area* efficiency. The systolic architecture can run the 1024 bits RSA decryption process in 3.23 ms, and the multiplexed implementation executes the same operation in 4.36 ms. Because of the multiplexed approach, the architecture is scalable. If the key size increases, the architecture can be easily modified by adding arithmetic cores, keeping the performance. Another speed improvement can be achieved by using a parallel modular exponentiation algorithm, for example, the Montgomery Powering Ladder [18] where a full modular exponentiation would be performed in exactly $n \times n_{\mathrm{MM}}$ clock cycles, that is, 33% faster than square and multiply algorithm.

Acknowledgment

This paper is result of project "INOVALAB-Laboratories Technological Innovation in Electronic and Microelectronic". We acknowledge the financial support received from FINEP.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] P. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[3] T. Blum and C. Paar, "Montgomery modular exponentiation on reconfigurable hardware," in *Proceedings of the Symposium on Computer Arithmetic (ARITH '99)*, pp. 70–77, IEEE Computer Society, Adelaide, Australia, 1999.

[4] N. Pinckney and D. M. Harris, "Parallelized radix-4 scalable montgomery multipliers," in *Proceedings of the 20th Symposium on Integrated Circuits and System Design (SBCCI '07)*, pp. 306–311, 2007.

[5] C. McIvor and J. V. McCanny, "High-radix systolic modular multiplication on reconfigurable hardware," in *Proceedings of the IEEE International Conference on Field Programmable Technology*, vol. 2005, pp. 13–18, 2005.

[6] C. D. Walter, "Systolic modular multiplication," *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 376–378, 1993.

[7] F. Bernard, "Scalable hardware implementing high-radix Montgomery multiplication algorithm," *Journal of Systems Architecture*, vol. 53, no. 2-3, pp. 117–126, 2007.

[8] Y. Wang, D. L. Maskell, and J. Leiwo, "A unified architecture for a public key cryptographic coprocessor," *Journal of Systems Architecture*, vol. 54, no. 10, pp. 1004–1016, 2008.

[9] A. Daly and W. Marnane, "Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic," in *Proceedings of the ACM/SIGDA 10th International Symposium on Field Programmable Gate Arrays (FPGA '02)*, pp. 40–49, 2002.

[10] D. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Reading, Mass, USA, 1951.

[11] Xilinx, "Virtex-5 user guide," March 2006, http://www.xilinx.com.

[12] G. Perin, D. G. Mesquita, F. L. Herrmann, and J. B. Martins, "Montgomery modular multiplication on reconfigurable hardware: fully systolic array vs parallel implementation," in *Proceedings of the 6th Southern Programmable Logic Conference (SPL '10)*, pp. 61–66, 2010.

[13] A. Menezes, *Handbook of Applied Cryptography*, CRC Press, New York, NY, USA, 5th edition, 2001.

[14] H. Orup, "Simplifying quotient determination in high-radix modular multiplication," in *Proceedings of the 12th Symposium on Computer Arithmetic (ARITH '95)*, pp. 193–199, July 1995.

[15] C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics Letters*, vol. 35, no. 21, pp. 1831–1832, 1999.

[16] A. F. Tenca and C. K. Koç, "A scalable architecture for modular multiplication based on montgomery's algorithm," *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1215–1220, 2003.

[17] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, "An improved unified scalable radix-2 montgomery multiplier," in *Proceedings of the Symposium on Computer Arithmetic (ARITH '05)*, pp. 172–178, 2005.

[18] M. Joye and S. M. Yen, "The montgomery powering ladder," in *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES '02)*, vol. 2523 of *Lecture Notes in Computer Science*, pp. 291–302, 2003.