PROJETO DE CIRCUITOS CRIPTOGRÁFICOS PARA APLICAÇÃO EM SEGURANÇA

Antônio Carlos Castañon Vieira

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

_____
Prof. Antônio Carneiro de Mesquita Filho, Dr. d'État

_____
Prof. Aloysio de Castro Pinto Pedroza, Dr.

_____
Prof. Jorge Lopes de Souza Leão, Dr. Ing.

_____
Prof. Julius César Barreto Leite, Ph.D.

_____
Prof. Luci Pirmez, D.Sc.

_____
Prof. Marco Aurélio Cavalcanti Pacheco, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2005

VIEIRA, ANTÔNIO CARLOS CASTAÑON

Projeto de Circuitos Criptográficos para Aplicação em Segurança [Rio de Janeiro] 2005

XVII, 148 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia Elétrica, 2005)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Descrição de Circuitos em *Hardware*

2. Circuitos Criptográficos

3. Aplicações em Segurança

I. COPPE/UFRJ II. Título (série)

Dedicatória

Aos meus queridos pai e mãe, *Waldesino Augusto Vieira* e *Solange Castañon Vieira*, que me orientaram por meio do exemplo, incentivando-me a ser um profissional cada vez melhor. E, sobretudo, à minha amada filha, *Bruna Gurgel Castañon*, razão de qualquer conquista.

Agradecimentos

Aos meus queridos pai e mãe, meus melhores amigos, Waldesino Augusto Vieira e Solange Castañon Vieira, pelo eterno incentivo, orientação, admiração, apoio à carreira escolhida e também pela presença constante para manter-me sempre no caminho correto da vida.

À minha irmã Carla Castañon Vieira, que fielmente esteve sempre a meu lado, numa demonstração de confiança, amizade, carinho e respeito.

À minha filha Bruna Gurgel Castañon, pela sua existência, significado de tudo que faço, e pelo seu sorriso, tão importante nos momentos difíceis de labor.

Aos professores Antônio Carneiro de Mesquita Filho e Aloysio de Castro Pinto Pedrosa, meus orientadores, pela honra e privilégio de ter sido por eles orientado e pelos valiosos conselhos e constantes incentivos para a realização deste trabalho, sempre dispostos a esclarecer qualquer dúvida e apontar soluções.

Aos professores Jorge Lopes de Souza Leão, Julius César Barreto Leite, Luci Primez e Marco Aurélio Cavalcanti Pacheco, por participarem da banca e contribuírem para o desenvolvimento deste trabalho.

Aos amigos que a Universidade Federal do Rio de Janeiro me proporcionou, em especial Ricardo Belém, Marcelo Azambuja, Fábio Dutra, Renato Bagatelli, David Fernandes, André da Costa Pinho, João Marcelo Alcântara e Sérgio Luiz Cardoso Salomão, pelo valioso incentivo, apoio, bom humor e amizade.

Aos amigos do Laboratório de Projetos de Circuitos e do Grupo de Teleinformática e Automação (GTA), pela ajuda e pelo uso dos equipamentos, particularmente ao Mestre Alexandre Andrade pela ajuda no uso do Network Simulator (NS-2), ferramenta fundamental na obtenção dos resultados deste trabalho.

À amiga Ana Paula Guimarães Gama, pela revisão minuciosa deste trabalho e pelo incentivo e apoio para concluí-lo.

Aos funcionários da COPPE, em especial à secretária do Programa de Engenharia Elétrica, Solange Coelho de Oliveira, que permitiram que o máximo possível do tempo fosse dedicado a atividades de pesquisa.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

PROJETO DE CIRCUITOS CRIPTOGRÁFICOS PARA APLICAÇÃO EM SEGURANÇA

Antônio Carlos Castañon Vieira

Setembro/2005

Orientadores: Aloysio de Castro Pinto Pedroza

Antônio Carneiro de Mesquita Filho

Programa: Engenharia Elétrica

Este trabalho tem como principal objetivo estudar algumas implementações em *hardware* dos algoritmos criptográficos: RSA, Curvas Elípticas, Blowfish e Godzuk, considerados muito seguros na atualidade, com aplicação em segurança. Os dois primeiros algoritmos são assimétricos para aplicações em que seja necessário o emprego de chaves públicas sem a existência de um canal seguro para a troca das chaves, e os demais são duas implementações de algoritmos simétricos de chave privada, para cenários em que se obtenha as chaves por um canal seguro ou a sua geração, também de forma segura. O trabalho se inicia com a apresentação das características dos algoritmos criptográficos, sua segurança, os principais aspectos da criptografia e a administração das chaves criptográficas. O projeto dos circuitos emprega conceitos de paralelismo e de síntese de circuitos em linguagem VHDL e desenvolve uma metodologia para estimativa do consumo de potência dos circuitos desenvolvidos nessa linguagem, além de realizar a integração desses circuitos em ferramentas para simulação de ambiente de redes no Network Simulator (NS-2). Por este trabalho mostra-se que o projeto de circuitos eletrônicos capazes de executar algoritmos criptográficos pode atender à demanda por segurança de informações.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

PROJECT OF CRIPTOGRAPHIC CIRCUITS FOR SECURITY APPLICATIONS

Antônio Carlos Castañon Vieira

September/2005

Advisors: Aloysio de Castro Pinto Pedroza

Antônio Carneiro de Mesquita Filho

Department: Electrical Engineering

This work has as main objective the study of some hardware implementations of the cryptographic algorithms: RSA, Elliptic Curve, Blowfish and Godzuk, considered very safe at the present time, with applications in security. The first two are asymmetric algorithms for applications where it is necessary the use of public keys without the existence of a safe channel for exchanging keys, and the other two are implementations of private key algorithms, for scenarios in which keys are obtained through a safe channel or generated by hosts in an unsafe channel with privacy. The work begins by defining the cryptographic algorithms characteristics and safety, the main aspects of the cryptography and the management of the cryptographic keys. During the circuit development, concepts of parallelism were used as well synthesis of circuits in language VHDL, besides the integration of these circuits in tools for networks simulation using Network Simulator (NS-2). It has been shown in this work that the design of an electronic circuit capable to execute cryptographic algorithms can assist the demand for information's safety.

Sumário

Lista de Figuras

Lista de Tabelas

Nomenclatura

3GPP – *Third Generation Partnership Project*

AC – Autoridade Certificadora

ANSI – *American National Standards Institute*

AODV – *Ad Hoc On-Demand Distance Vector*

ASCII – *American Standard Code for Information Interchange*

ASIC – *Application Specific Integrated Circuit*

CBR – *Constant Bit Rate*

CI – Circuito Integrado

CMOS – *Complementary Metal-Oxide Semiconductor*

DES – *Data Encryption Standard*

ECC – *Elliptic Curve Cryptography*

EPLD – *Erasable Programmable Logic Device*

FIFO – *First In First Out*

FPGA – *Field Programmable Gate Array*

IDEA – *International Data Encryption Standard*

IEEE – *Institute of Electrical and Electronics Engineers*

IETF – *Internet Engineering Task Force*

IMT-2000 – *International Mobile Telecommunications* – 2000

ISO – *International Organization for Standardization*

ITU – *International Telecommunications Union*

KDC – *Key Distribution Center* [Centro de Distribuição de Chaves]

KTC – *Key Translation Center* [Centro de Tradução de Chaves]

LAN – *Local Area Network*

LSB – *Least Significant Bit*

MAC – Controle de Acesso ao Meio

MANET – *Mobile Ad Hoc Networking*

MMB – *Modular Multiplication-based Block cipher*

MSB – *Most Significant Bit*

NS-2 – *Network Simulator version* 2.26

PGP – *Pretty Good Privacy*

RAM – *Random Access Memory*

RISC – *Reduced Instruction Set Computer*

RM-OSI – *Open System Interconnection Reference Model*

RSA – Algoritmo criptográfico assimétrico desenvolvido por Ron Rivest, Adi Shamir, e Leonard Adleman

RTL – *Register Transfer Level*

SCOB – *Software Core for the Blowfish Cryptographic Algorithm*

SER – *Scheduling by Edges Reversal*

SMER – *Scheduling by Multiple Edges Reversal*

TE – Terceira Entidade

TTP – *Trusted Third Party*

UDP – *User Datagram Protocol*

VHDL – *Very High Speed Integrated Circuit Hardware Description Language*

VLSI – *Very Large Scale Integration*

WAN – *World Area Network*

WLAN – *Wireless Local Area Network* [Rede Local Sem Fio]

Capítulo 1

Introdução

Os sistemas de segurança de dados baseados em criptografia têm assumido crescente importância como principal meio de garantir a confiabilidade das informações, quer sejam armazenadas em computadores, quer sejam transmitidas pelas redes de telecomunicações [1].

Com o avanço dessas redes, o controle sobre a informação se tornou cada vez mais estratégico. Além disso, quanto maior o fluxo de informação em redes de telecomunicações ou quanto maior a quantidade de informação armazenada em meio computacional, maior a necessidade de empresas, governos e até pessoas físicas de se protegerem contra ameaças que crescem proporcionalmente ao desenvolvimento da informática [2].

A criptografia tem surgido como uma solução possível não apenas para proteger a informação e prover segurança, mas também para fornecer autenticidade das informações enviadas, característica exigida sobretudo pelas redes de comunicações atuais caracterizadas pela mobilidade.

Atualmente, houve uma mudança na forma de se comunicar por meio dos equipamentos móveis. Os serviços de comunicação celular, particularmente, são usados não apenas para as comunicações móveis de voz, mas também para o comércio eletrônico e trocas de informações.

Cada vez mais a transmissão de dados, e não mais a voz, torna-se o serviço mais importante dos sistemas móveis [1]. Está prevista a necessidade de transmissão de voz, imagem e acesso à Internet [3], num mesmo terminal e simultaneamente, tornando essas novas redes móveis sensíveis à integridade e à segurança dos dados [4] e, conseqüentemente, aumentando a necessidade de sistemas de proteção e de processamento por parte desses terminais [5].

A criptografia é uma técnica tão antiga quanto a própria escrita, mas até recentemente tinha aplicações apenas no campo militar. Com os computadores, a área cresceu incorporando complexos algoritmos matemáticos [6]. Dessa forma, ela foi adaptada pela informática para atingir o nível de segurança desejado nos dias de hoje pelos diversos serviços de comunicações digitais pessoais e comerciais [7].

Essencialmente, esses algoritmos aplicam uma fórmula matemática aos dados a serem protegidos. No entanto, para serem realmente seguras, essas expressões matemáticas são normalmente muito complexas e exigem muitos recursos de processamento.

Devido a essas características, os algoritmos criptográficos passaram a ter o seu processamento mais demorado, em função tanto do maior número de operações internas necessárias a sua execução, como também do tamanho dos dados e chaves de código a serem processados, ocasionando um compromisso em relação à compatibilidade entre a velocidade de execução desses algoritmos e a velocidade das redes de computadores a serem protegidas.

As técnicas de projeto de circuitos integrados de aplicação específica (ASIC – *Application Specific Integrated Circuit*) e as arquiteturas de processadores, sobretudo de plataforma RISC (*Reduced Instruction Set Computer*) e dispositivos programáveis (FPGA – *Field Programmable Gate Array*), tiveram grande evolução nos últimos anos e, combinadas com a tecnologia de fabricação de circuitos, têm permitido um avanço significativo nas concepções de circuitos eletrônicos dedicados mais complexos.

O objetivo deste trabalho, por conseguinte, é implementar alguns algoritmos criptográficos em forma de circuitos integrados dedicados, ou, pelo menos, as partes mais relevantes para a velocidade de processamento, aproveitando as principais vantagens da criptografia através de *hardware*: o aumento da velocidade de processamento, a redução do consumo de potência e a dificuldade de reprodução do

produto.

1.1. Trabalhos correlatos

O aumento significativo das transações comerciais eletrônicas gerou uma demanda por segurança e privacidade. É amplamente aceito, pela comunidade científica, que implementações em *hardware* de algoritmos criptográficos, sobretudo os algoritmos assimétricos, têm melhor desempenho e são fisicamente mais seguras que as correspondentes implementações em *software* [7], permitindo prover segurança às redes de transmissão de dados sem perda significativa de desempenho.

O artigo [8] descreve as tendências de segurança para as redes móveis *Ad Hoc*, abordando suas necessidades e particularidades e indicando que essas redes não podem se beneficiar dos mecanismos de segurança desenvolvidos para as redes fixas, devendo utilizar serviços de segurança distribuídos e cooperativos.

Muitos grupos de pesquisa em diversas universidades têm desenvolvido alguns algoritmos criptográficos em *hardware* em diferentes plataformas. Os trabalhos [7], [9], [10] e [11] descrevem a implementação do algoritmo criptográfico RSA.

Existem também alguns trabalhos de implementações em *hardware* do algoritmo criptográfico de curvas elípticas. Entre as arquiteturas mais eficientes estão as apresentadas em [12], [13] e [14]. O trabalho desenvolvido em [12] usa um circuito de aplicação específica (ASIC) para implementar um acelerador para matemática em corpos finitos, [13] realiza as operações mais críticas do algoritmo em FPGA e [14] executa o algoritmo em microprocessadores.

Os algoritmos criptográficos simétricos possuem menor número de implementações em *hardware* que os algoritmos assimétricos. [15] descreve o desenvolvimento de um cricuito eletrônico que executa o algoritmo Blowfish. O algoritmo Godzuk não possui, até o momento, outra implementação em *hardware*.

1.2. Objetivos do trabalho

Este trabalho enfatiza a implementação em *hardware* de algoritmos criptográficos robustos, simétricos e assimétricos, para aplicações específicas em redes sem fio. O emprego da criptografia nessas redes deve-se ao fato de que toda a informação, tanto dados como controle, deve ser protegida, tendo como conseqüência que parte do processamento do protocolo de comunicação é destinada à execução desses algoritmos criptográficos.

Existem vários compromissos que definem a implementação de algoritmos criptográficos. Para melhor atender esses compromissos será empregada uma técnica para avaliar seu impacto e desempenho nas redes sem fio e compará-los com outras soluções. Esses resultados serão importantes para analisar os compromissos de implementação dos protocolos de comunicações, abordando o problema das perdas de pacotes, atrasos e consumo de potência nos terminais.

Para isso, os seguintes objetivos intermediários deverão ser atingidos:

- propor arquiteturas de hardware para os algoritmos criptográficos RSA, ECC, Blowfish e Godzuk;
- aplicar a metodologia de avaliação do consumo de potência desenvolvida no Laboratório de Projetos de Circuitos (LPC) da COPPE/UFRJ nas arquiteturas projetadas de forma comparativa;
- avaliar as arquiteturas desenvolvidas, elaborando testes no simulador de rede sem fio Network Simulator (NS-2), de modo a comprovar a eficiência relativa dos algoritmos utilizados e o impacto do seu uso. O desempenho das arquiteturas dos algoritmos propostos são avaliados em uma aplicação de segurança em termos de taxa de transmissão, em função dos atrasos e do consumo de potência estimado, obtendo uma indicação do algoritmo mais adequado.

1.3. Estrutura do trabalho

Esta tese está dividida em sete capítulos. O primeiro apresenta o tema de forma geral e introdutória.

No Capítulo 2 são abordados os algoritmos criptográficos sua segurança, os principais aspectos da criptografia e a administração das das chaves criptográficas.

No Capítulo 3 é realizado o desenvolvimento de uma arquitetura para implementar os algoritmos criptográficos assimétricos RSA e ECC propostos.

No Capítulo 4 são apresentadas as arquiteturas dos algoritmos criptográficos simétricos Blowfish e Godzuk propostos.

O Capítulo 5 apresenta a metodologia de estimativa do consumo de potência, baseada na atividade dos circuitos, a partir de suas descrições em linguagem de descrição de *hardware* das arquiteturas desenvolvidas dos algoritmos criptográficos. Esta metodologia é uma contribuição deste trabalho de tese, tendo sido desenvolvida em conjunto com [72] [75].

No Capítulo 6 é apresentada uma breve aplicação em segurança, com base no impacto da inserção do *hardware* de criptografia desenvolvido no desempenho de protocolos de comunicações de uma rede sem fio, avaliado através de simulação no ambiente NS-2.

Finalmente, no Capítulo 7 são apresentadas as conclusões e propostas de trabalhos futuros baseados no procedimento adotado.

Capítulo 2

Algoritmos criptográficos

As soluções tradicionais para garantir segurança e autenticidade de mensagens sujeitas a ataques sempre envolveram técnicas de escrita secreta. Usualmente destinadas a manter o conteúdo de uma mensagem desconhecido por um inimigo, essas técnicas também asseguravam a origem e a integridade da mensagem. A criptografia é uma das principais técnicas empregadas para garantir tal segurança aos sistemas de informação [16]. Um sistema de criptografia pode ser subdividido em algoritmo criptográfico, chave criptográfica, texto em claro e texto cifrado [17]. Neste Capítulo são apresentados alguns desses subconjuntos e a forma como eles interagem.

Os processos de transformação da mensagem em claro em criptograma e do criptograma na mensagem em claro original são realizados por algoritmos criptográficos através da cifragem e da decifragem, respectivamente. Tanto para cifragem quanto para decifragem é necessário um segundo parâmetro: a chave.

Um algoritmo criptográfico é uma função matemática usada nos processos de cifragem e decifragem, que trabalha em conjunto com uma chave – uma palavra, número ou frase – para cifrar um texto. Assim, a segurança de um sistema criptográfico não pode estar baseada apenas nos algoritmos de codificação e de decodificação, mas também nas chaves [17]. O mecanismo deve ser tão seguro que nem mesmo o autor de um algoritmo possa ser capaz de decodificar uma mensagem se não possuir a chave.

São dois os tipos de algoritmos criptográficos [17][18] classificados quanto à chave usada no processo de cifragem/decifragem: os sistemas de criptografia clássica, conhecidos também como sistemas convencionais, simétricos, de chave secreta, ou ainda, de chave privada, em que a chave é única para cifrar e decifrar, ou seja, a chave é compartilhada pelo emissor e pelo receptor; e os sistemas mais recentes, que são denominados sistemas assimétricos ou de chave pública, que possuem chaves distintas para cifrar e decifrar, sendo uma privada e outra pública.

A vantagem da utilização da criptografia baseada em chave pública sobre a criptografia baseada em chave secreta envolve maior segurança, não do algoritmo em si, mas na forma como as chaves do sistema criptográfico de chave pública são administradas [19][20][21], em que cada usuário protege sua chave privada sem a necessidade de compartilhá-la com outros usuários do sistema. Neste caso, as chaves privadas não precisam ser transmitidas nem recebidas. Essa característica fornece ainda um método simples para assinatura digital.

A principal desvantagem da criptografia baseada em chave pública é a velocidade na cifragem. Em geral os métodos de criptografia por chave secreta são significativamente mais rápidos que os de chave pública [22].

Os algoritmos simétricos mais conhecidos são o DES (*Data Encryption Standard*) [23] e o IDEA (*International Data Encryption Standard*) [24], além dos algoritmos FEAL (*Fast Data Encipherment Algorithm*) [25], RC2 [17], Blowfish [26] e Godzuk [27], entre outros. Os algoritmos assimétricos mais populares atualmente são o RSA (iniciais de Rivest, Shamir e Adleman, seus inventores) [28] e o ECC (*Elliptic Curve Cryptography*) [29]. Outros exemplos de algoritmos de chave pública são o El Gamal e Pohlig-Hellman [30].

2.1. Algoritmos de chave privada (simétricos)

Até 1978, todos os sistemas criptográficos necessitavam que a cifragem e a decifragem tivessem alguma informação em comum (chave secreta) [31]. Essa informação tinha que ser mantida em segredo, pois a chave para decifragem era a mesma utilizada na cifragem. Por isso eles são também chamados de algoritmos de

chave única ou algoritmos de chave secreta.

A Figura 1 apresenta um diagrama em blocos do algoritmo de chave secreta.

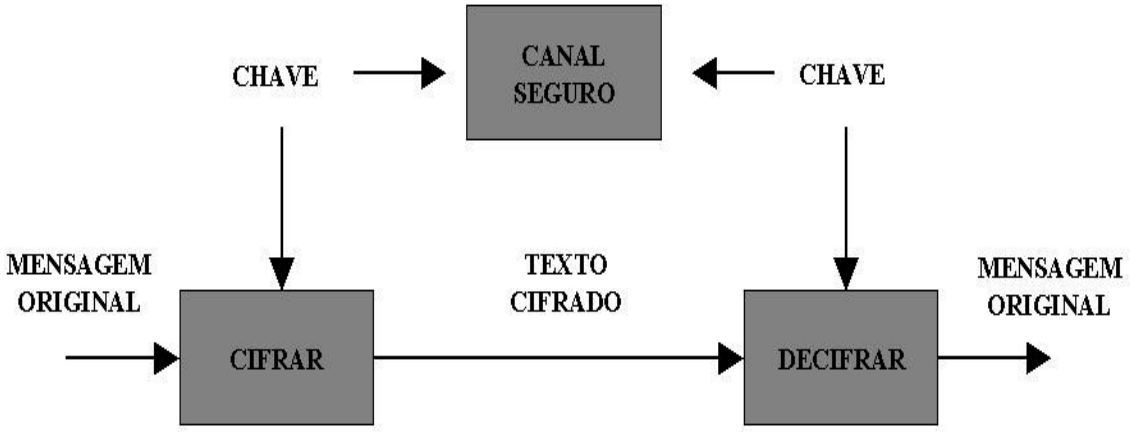


Figura 1 – Sistema criptográfico simétrico

Apesar dos sistemas criptográficos de chave secreta serem ainda muito usados, eles apresentam algumas desvantagens descritas a seguir:

- distribuição da chave – a fim de estabelecer uma comunicação secreta, o remetente e o destinatário devem trocar a chave secreta ou descobrir algum meio seguro para compartilhá-la. Sendo assim, eles dependem de um canal seguro para essa troca;
- gerenciamento da chave – muitos dos sistemas criptográficos modernos são implementados para redes de computadores. Caso existam N usuários e cada um deseje trocar segredos com todos os outros, então deverá haver uma chave privada para cada par de usuários, perfazendo um total de $N(N-1)/2$ chaves no total. Para um sistema com um grande número usuários, o gerenciamento das chaves pode se tornar inviável;
- assinatura digital – aplicações modernas de criptografia vão além da tradicional troca de segredos. Muitas novas aplicações contam com um método de autenticação conhecido como assinatura digital. Nos sistemas de chave privada, essas aplicações são potencialmente limitadas.

O algoritmo simétrico mais difundido atualmente é o *Data Encryption Standard* ou DES [23]. Esse algoritmo foi desenvolvido pela IBM e adotado como padrão pelos

Estados Unidos da América em 1977. O DES cifra blocos de 64 *bits* (oito caracteres) usando uma chave de 56 *bits*, mais oito *bits* de paridade (o que soma 64 *bits*).

O DES aplica, em cada bloco de 64 *bits*, operações matemáticas muito simples, e essas operações são realizadas em 16 passos de cifragem. Nesses 16 passos de cifragem usam-se 16 subchaves, todas derivadas da chave original por meio de deslocamentos e transposições.

Um passo de cifragem do DES tem dois objetivos básicos: a difusão e a confusão. A difusão visa eliminar a redundância da mensagem original, distribuindo-a pela mensagem cifrada. O propósito da confusão é tornar a relação entre a mensagem e a chave tão complexa quanto possível.

Contudo, como o DES possui uma chave de apenas 56 *bits*, é possível, por meio do método de força bruta, que um ataque tenha sucesso. Em média, um ataque necessita tentar 2^{56} chaves diferentes, ou aproximadamente 10^{17} possibilidades. Diffie e Hellman [32] conjeturaram a construção de uma máquina a 10^{12} ciframentos por segundo. A esta taxa seria possível exaurir todas as combinações em um dia.

2.2. Algoritmos de chave pública (assimétricos)

Uma alternativa para a criptografia de chave privada foi publicada em 1976 por Whitfield Diffie e Martin Hellman [33]. Eles propuseram um sistema criptográfico de chave pública no qual segredos podem ser trocados em canais inseguros sem a necessidade de primeiramente trocar a chave privada. A Figura 2 representa, de forma simplificada, um algoritmo de chave pública.

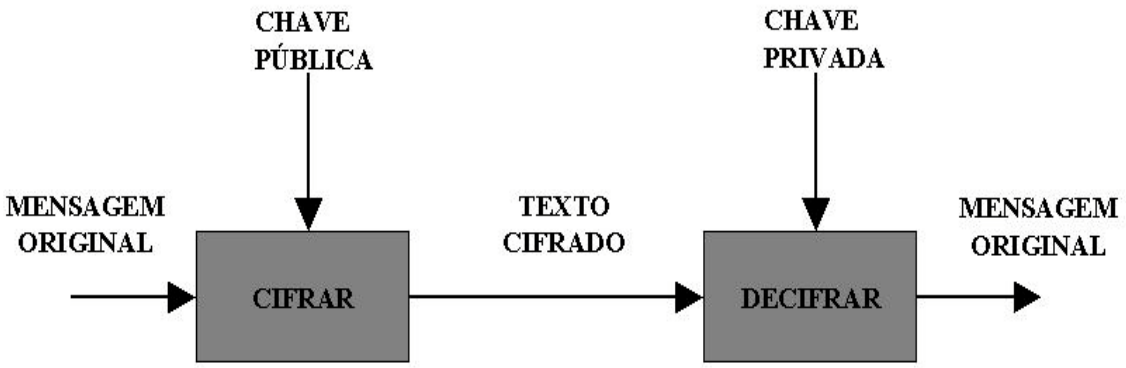


Figura 2 – Sistema criptográfico assimétrico

No sistema criptográfico de chave pública, como foi visto, cada usuário possui um par de chaves chamadas de chave pública e chave privada. A chave pública de cada pessoa é divulgada, enquanto a chave privada é mantida em segredo. A chave pública é usada para cifrar e é amplamente transmitida por meio dos canais de comunicação, seguros ou não. A chave privada é usada para decifrar e nunca é transmitida. A mensagem cifrada pela chave pública só pode ser decifrada pela chave privada correspondente. O inverso também pode ser feito, ou seja, usar a chave privada para cifrar e a chave pública para decifrar. Isso é conhecido como autenticação ou assinatura digital.

Outra vantagem do recurso descrito acima é que, além de autenticar a fonte da mensagem, é possível garantir que os dados não foram alterados ou destruídos de forma não autorizada.

Nos sistemas criptográficos assimétricos, as chaves (pública e privada) possuem uma relação matemática entre si, através da qual a chave pública é facilmente calculada a partir da chave privada. Contudo, para que este tipo de sistema seja confiável, é fundamental que não se possa calcular a chave privada com base na chave pública.

A função matemática que satisfaz essas duas propriedades é conhecida como função alçapão de caminho único (do inglês *trap-door one-way function*) [34]. O termo *caminho único* vem do fato de que, mesmo sem conhecimento da chave privada, é muito mais fácil cifrar uma mensagem do que decifrá-la. O termo *função alçapão* justifica-se por funcionar como uma armadilha que permite decifrar a mensagem com a chave privada.

Os sistemas criptográficos de chave pública superam muitos dos problemas encontrados com os sistemas de chave privada e que são descritos a seguir:

- distribuição de chaves – um sistema de criptografia de chave pública permite que dois usuários estabeleçam uma comunicação segura, mesmo que eles nunca tenham se encontrado ou dividido um canal seguro para troca de chaves;
- gerenciamento da chave – caso existam *N* usuários de um sistema de criptografia de chave pública, então, para cada par de usuários ser capaz de trocar seus segredos, serão necessárias somente *N* chaves, uma para cada usuário;
- assinatura digital – nesse tipo de sistema, pode-se empregar a chave privada (que é de conhecimento apenas do usuário) para cifrar, garantindo que somente o proprietário da chave privada pode ter enviado a mensagem;
- integridade dos dados – corroborar que a mensagem enviada não sofreu alteração no seu conteúdo durante a transmissão; esse sistema pode fornecer proteção contra mensagens alteradas, inseridas, apagadas ou reenviadas.

Certamente, os sistemas de chave pública têm desvantagens. Por usar um processo matemático complexo para cifrar e decifrar, esses sistemas possuem um desempenho baixo em termos de velocidade de processamento. Por exemplo, o RSA, sistema criptográfico assimétrico mais popular, pode ser mais de 10.000 vezes mais lento que o DES, sistema criptográfico simétrico mais popular [35].

As principais diferenças entre os dois tipos de sistemas criptográficos estão resumidas na Tabela 1.

Características	Sistemas Simétricos	Sistemas Assimétricos
Desempenho	Mais rápidos	Mais lentos
Chave (distribuição)	Difícil (em segredo)	Fácil (em aberto)
Número de chaves em uma rede de *N* usuários	$N(N-1)/2$ chaves (uma chave para cada par de usuários)	*N* chaves diferentes
Assinatura digital	Limitado	Permite
Função criptográfica	Possui inversa (decifragem)	Não possui inversa (*trap-door-function*)

Tabela 1 – Comparação entre algoritmos de chave pública e chave privada

2.3. Criptoanálise

A criptoanálise é a área da criptologia que abrange os princípios, métodos e meios para se chegar à decifragem de um texto cifrado, sem prévio conhecimento dos códigos ou chaves empregadas na sua produção. Uma tentativa de criptoanálise é chamada de ataque.

Primeiramente, deve-se notar que qualquer sistema criptográfico inclui não apenas um método matemático para cifrar e decifrar, mas também um protocolo de comunicação, pois em muitos casos a segurança está comprometida não por um ataque à matemática do sistema, mas pelos aspectos práticos de sua utilização e emprego [36]. Por exemplo, qualquer sistema de decifragem deve armazenar a chave secreta em *software* ou em *hardware*. Um ataque a essa chave pode ser mais eficaz que qualquer ataque matemático ao texto cifrado.

Uma pessoa não autorizada que tem acesso a alguns dos elementos de um sistema criptográfico é denominada criptoanalista. A segurança dos sistemas criptográficos está baseada em sua capacidade de resistir aos ataques. Todos esses ataques supõem que o criptoanalista conhece os métodos de cifrar e decifrar utilizados, mas não conhece as chaves. Os tipos de ataque são [18][37]:

- ataque somente com texto cifrado conhecido: o criptoanalista possui o texto cifrado e qualquer acesso a outras informações públicas; este é o tipo de

ataque mais fraco, e um sistema que não resiste a ele é considerado totalmente inseguro;

- ataque conhecendo o texto original: neste caso, o criptoanalista possui uma quantidade razoável de textos cifrados e mensagens originais correspondentes, e utiliza essas informações para tentar encontrar qualquer informação privada, tal como a chave secreta ou mesmo a função matemática para decifrar;
- ataque com textos originais escolhidos: o criptoanalista pode submeter ao sistema criptográfico uma quantidade de textos originais e observar o texto cifrado correspondente; este método é mais poderoso que os outros, pois o criptoanalista pode escolher qualquer combinação de texto original e analisar seu resultado;
- ataque da chave escolhida: o criptoanalista pode testar o sistema com diversas chaves diferentes, ou pode convencer diversos usuários legítimos do sistema a utilizarem determinadas chaves; neste último caso, a finalidade seria decifrar as mensagens cifradas com essas chaves;
- ataque por força bruta: pelo método de pura exaustão, aplica-se o algoritmo a um determinado texto cifrado, variando-se a chave até que seja produzido um texto em claro; descobre-se desta forma qual a chave usada, e todos os textos cifrados com ela são, conseqüentemente, decifrados. Na maioria dos algoritmos criptográficos em que é empregado um grande número de chaves possíveis, mesmo quando a sistemática da força bruta for realizada por computadores muito rápidos, a descoberta da chave correta levaria muito tempo, inviabilizando este processo.

2.4. Administração de chaves criptográficas

Administração de chaves criptográficas é o conjunto de processos e mecanismos que definem a forma como as chaves secretas serão estabelecidas, compartilhadas, disponibilizadas, manutenidas e substituídas entre as diversas partes da comunicação do sistema criptografado, de modo a garantir que os requisitos de segurança estabelecidos

sejam atendidos [18].

Com o emprego dos sistemas criptográficos, a administração e o estabelecimento das chaves representam um aumento do processamento devido ao crescimento da troca de mensagens na rede e do número de mensagens, quantidade de dados transmitidos e tempo de execução desses protocolos [19][38].

Para se ter uma idéia do custo do gerenciamento de chaves é necessário estudar os diversos mecanismos utilizados na administração das chaves criptográficas.

2.4.1. Gerenciamento de chaves

A maioria dos mecanismos dos serviços de segurança que utilizam algoritmos criptográficos requer o emprego de chaves criptográficas que precisam ser compartilhadas entre as partes do processo de comunicação. Os propósitos fundamentais da administração de chaves criptográficas são [18]:

- autorizar o acesso dos usuários de sistemas dentro de um domínio;
- gerar, distribuir e instalar chaves;
- controlar o uso das chaves;
- atualizar e eliminar as chaves;
- armazenar as chaves.

Além de atender às características acima, os sistemas que administram as chaves devem garantir pelo menos três compromissos fundamentais junto aos usuários das chaves: confidencialidade das chaves, autenticidade das chaves e uso das chaves somente com autorização e válidas.

O estabelecimento da chave a ser utilizada pode ocorrer de forma centralizada, por meio de uma terceira entidade, ou distribuída. Na forma centralizada, uma entidade é responsável pela geração da chave e sua distribuição aos demais membros. Esta abordagem apresenta a grande vantagem de ser simples. Já na forma distribuída, todos os membros do grupo contribuem para a geração da chave. Pode haver uma abordagem híbrida, na qual apenas um subconjunto dos membros é responsável pela geração da

chave, a qual é então distribuída aos outros membros.

Embora um usuário possa enviar uma mensagem cifrada e assinada para outro usuário, existe ainda a necessidade de garantir que realmente está havendo comunicação segura entre eles com o uso dessas chaves.

É importante saber que, ao utilizar uma chave criptográfica, uma terceira entidade precisa dar a garantia de que está sendo usada uma chave que é efetivamente correspondente às partes, haja vista que qualquer entidade tem a possibilidade de gerar chaves correlacionadas e distribuí-las.

A solução para esse problema é a criação de uma terceira entidade (TE) na qual todos os usuários do sistema confiam e que é usada para prover os serviços fundamentais de administração das chaves.

Para evitar que um atacante publique uma chave fazendo-se passar por outra entidade da rede, faz-se necessário que cada usuário registre sua chave pública junto à Autoridade Certificadora (AC), que é confiável e que divulga um certificado, garantindo a autenticidade da chave desse usuário. Essa entidade confiável equivale a um cartório eletrônico onde se pode depositar uma chave pública, obtendo-se um certificado associando tal chave pública ao seu proprietário, ou seja, à entidade que gerou e que detém a chave privada correspondente.

As técnicas de estabelecimento das chaves criptográficas a serem utilizadas nas redes *Ad Hoc* podem ser classificadas quanto ao emprego dessas terceiras entidades (TE) em: *in-line*, *on-line* e *off-line*, como mostra a Figura 3.

A Figura 3 (a) mostra uma comunicação segura entre dois nós utilizando a TE *in-line*, ou seja, neste caso, a TE participa ativamente da comunicação entre os nós, se colocando como intermediária na troca de mensagens. Já a TE *on-line* participa, também ativamente, mas somente para propósitos de administração, e a comunicação entre os usuários é direta, como ilustra a Figura 3 (b).

Na TE *off-line*, Figura 3 (c), o estabelecimento das chaves é realizado antes do desdobramento e iniciação da rede, indicado pela ligação pontilhada entre a TE e os nós. A TE não é ativa durante a comunicação das partes, sendo que, na realidade não é necessário nem mesmo estar conectada à rede.

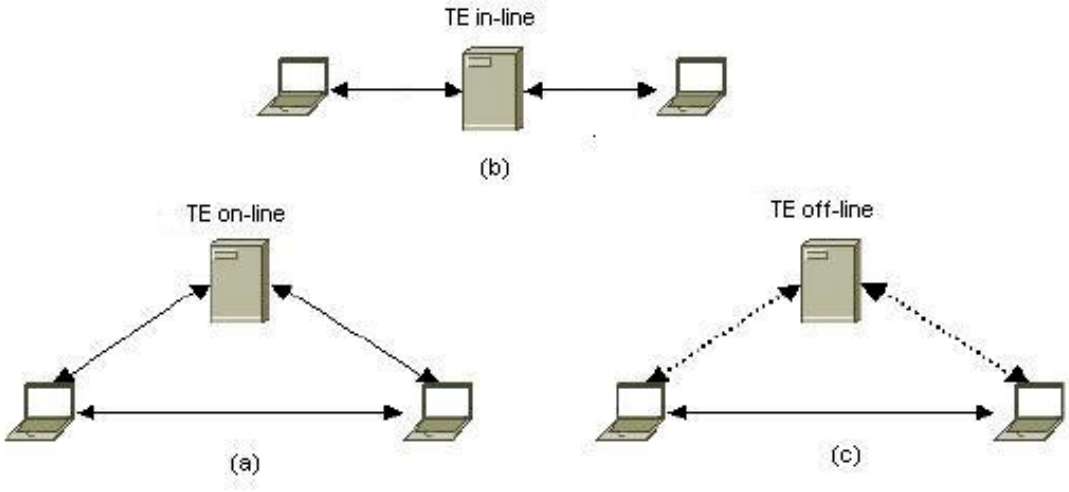


Figura 3 – Tipos de terceira entidade no processo de comunicação segura

Os exemplos de terceiras entidades são Centros de Distribuição de Chaves (KDC – *Key Distribution Centers*), Centros de Tradução de Chaves (KTC – *Key Translation Centers*) e Autoridades Certificadoras (AC). O KDC e o KTC são sistemas administradores de chaves simétricas e a AC é um sistema de administração de chaves públicas. Os KDC e KTC são usados para simplificar a administração de chaves. Ao invés de cada usuário ter que compartilhar uma chave secreta com cada outro usuário, eles só precisam compartilhar a chave pessoal com o *Trusted Third Party* (TTP). Isso minimiza o número de chaves que precisam ser administradas de $N(N-1)/2$ para apenas N, onde N é o número total de usuários da rede.

A Figura 4 ilustra um exemplo do emprego de uma TE para obtenção de chave criptográfica em uma sessão de comunicação entre *A* e *B*. 1) O usuário *A* solicita uma chave para ser compartilhada com o usuário *B*. Se a TE é do tipo KDC, ela gera a chave. Nesta etapa, a comunicação é realizada de forma segura por meio da chave compartilhada entre *A* e a TE. 2) A TE codifica a chave gerada com a chave que ela compartilha com *B*. 3) O usuário *A*, então, envia a chave de sessão codificada para o usuário *B*, que pode decifrá-la e utilizá-la para uma sessão segura com *A*.

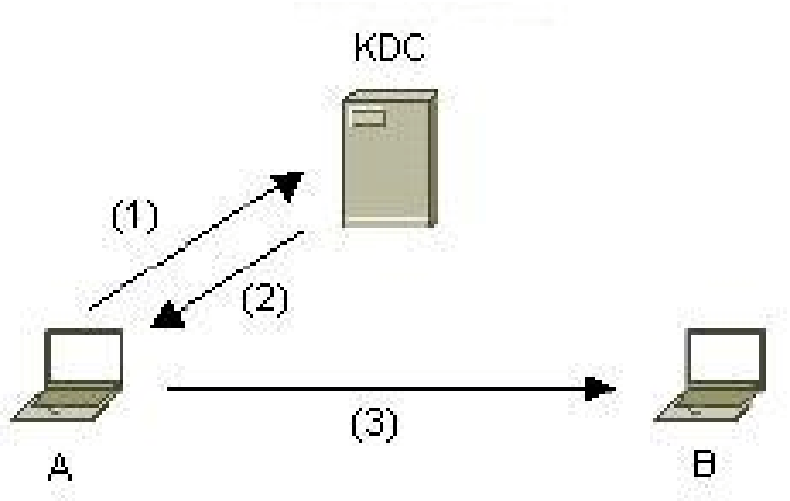


Figura 4 – Exemplo do emprego de uma terceira entidade do tipo KDC

2.4.2. Certificado digital

O uso de criptografia de chave pública requer a garantia da autenticidade das chaves, assegurada por uma terceira entidade chamada Autoridade Certificadora (AC), que é a componente principal da infra-estrutura de chave pública e responsável pela emissão dos certificados digitais.

Uma entidade que possui um certificado digital, quando se comunica com outro usuário, avisa que possui um certificado. Este segundo usuário vai à AC do emissor e confere os dados do certificado, podendo verificar suas credenciais e principalmente a veracidade de sua chave pública.

Um certificado digital é um arquivo composto por chave pública e informações do certificado, tais como validade, nome da pessoa, órgão emissor, uma ou mais assinaturas dos órgãos que garantem o certificado.

Nessas condições, em uma troca de mensagens entre duas entidades comunicantes, se cada parte tem um certificado assinado por uma autoridade confiável, então ambas as entidades têm alguma garantia de que estão efetivamente se comunicando com uma entidade bem identificada e autêntica. A terceira entidade confiável, que emite os certificados, é a AC, e os certificados digitais são usados para autenticação de entidades comunicantes, para determinação de chaves a serem usadas

nas comunicações e para registrar históricos das comunicações, de modo que as transações possam ser verificadas e comprovadas mais tarde. Esta característica é fundamental em redes que exijam responsabilidade e tenham grande hierarquia entre os usuários, como em aplicações militares.

2.5. Comentários

Neste Capítulo foram introduzidos os fundamentos dos sistemas criptográficos baseados em chaves privadas e públicas, além das formas de administração das chaves criptográficas, sendo apresentadas e discutidas suas principais características e limitações. Por meio dessa teoria foi possível verificar que os sistemas de chave pública têm o desempenho como desvantagem e a facilidade de estabelecimento de chaves como vantagem, quanto comparados aos sistemas de chave privada.

Devido à baixa velocidade do processo de cifragem e decifragem dos algoritmos assimétricos, torna-se ineficiente a sua utilização para cifrar textos grandes. Adicionalmente, o tamanho considerado seguro das chaves utilizadas nesses algoritmos é um inconveniente, pois são muito grandes.

Pode-se notar que ambos os tipos de algoritmos criptográficos oferecem vantagens e desvantagens. Nenhum dos sistemas simétrico e assimétrico, separadamente, oferece total eficiência no processo de cifragem e decifragem, no tocante a número de chaves, velocidade de processamento, distribuição de chaves.

No próximo Capítulo serão apresentados os quatro algoritmos criptográficos, sendo dois simétricos (Blowfish e Godzuk) e dois assimétricos (RSA e ECC) implementados neste trabalho, explanando suas características e comparando suas possibilidades e limitações no emprego em sistemas de redes sem fio.

Capítulo 3

Implementação de algoritmos criptográficos assimétricos

O aperfeiçoamento de ferramentas automáticas de projetos de circuitos eletrônicos digitais, tais como ferramentas de síntese de alto nível, permitem que sistemas possam ser implementados de um modo mais rápido. A metodologia usada neste trabalho empregou essas ferramentas automáticas, possibilitando o desenvolvimento de sistemas criptográficos com alto desempenho.

Neste Capítulo são apresentadas descrições do funcionamento dos algoritmos criptográficos RSA e ECC, além de suas implementações em *hardware*. Considerando-se os aspectos referentes ao desempenho de tais algoritmos, permitiu-se proceder a um estudo comparativo. Os resultados obtidos podem auxiliar na escolha de algoritmos criptográficos em aplicações que necessitem de determinados aspectos de desempenho.

3.1. Algoritmo criptográfico assimétrico RSA

O sistema criptográfico RSA [28], publicado em 1978, foi a primeira implementação prática de um sistema de chave pública. Seu nome é formado pelas iniciais dos nomes dos seus autores, Rivest, Shamir e Adleman. É largamente empregado em programas de criptografia para correio eletrônico na Internet, como o *Pretty Good Privacy* (PGP) [17], além de ser usado em alguns dos navegadores da Internet, como o Netscape Navigator e o Microsoft Explorer, como segurança da camada de *sockets*.

O RSA usa a teoria dos números inteiros e a aritmética modular para transformar uma mensagem em um texto cifrado. Essas operações matemáticas são processadas por uma função do tipo *trap-door one-way function*.

3.1.1. Funcionamento do RSA

A chave pública consiste de dois números, n e e. O primeiro número n é estabelecido pelo produto de dois números primos distintos, p e q, escolhidos aleatoriamente. O número e pode ser qualquer inteiro positivo, desde que seja inversível módulo $\phi(n)$.

Em outras palavras,

$$\text{mdc}(e, \phi(n)) = 1 \qquad \text{Equação 1}$$

onde $\phi(n)$ é calculada como:

$$\phi(n) = (p - 1) \cdot (q - 1) \qquad \text{Equação 2}$$

Portanto, a chave pública do sistema RSA consiste no par (n, e).

3.1.1.1. Cálculo da chave privada

A chave privada é, também, formada por dois números, n e d, onde d é o inverso de e em $\phi(n)$. Assim:

$$e \cdot d \equiv 1 \bmod \phi(n) \qquad \text{Equação 3}$$

$$e \cdot d = 1 + k\,\phi(n), \text{ onde } k \in Z. \qquad \text{Equação 4}$$

3.1.1.2. Cifragem

A mensagem original, depois de pré-codificada, deve ser subdividida em blocos menores que n, $m < n$. O texto cifrado é calculado usando a exponencial modular:

$$c = m^e \bmod n \qquad \text{Equação 5}$$

3.1.1.3. Decifragem

Utiliza-se a chave privada para decifrar a mensagem:

$$m = c^d \bmod n \qquad \text{Equação 6}$$

É importante notar que a mesma função é usada tanto para cifrar quanto para decifrar. Sendo assim, o mais importante na implementação do sistema criptográfico RSA é a função exponencial modular. O algoritmo e a arquitetura dessa função será abordado nos próximos capítulos.

Para um mesmo valor de n, o tempo necessário para executar a função exponencial modular é rigorosamente proporcional ao número de *bits* do expoente. Por essa razão, é mais interessante escolher valores de e e d pequenos. Contudo, existem problemas de segurança para a escolha de pequenos valores para o expoente secreto d. Logo, o mais comum é usar valores de d aproximadamente do mesmo tamanho de n.

3.1.2. Segurança do RSA

O RSA é um sistema criptográfico de chave pública, sendo p e q seus parâmetros e $n = p \cdot q$. A chave de codificação corresponde à chave pública do sistema. Portanto, o par (n, e) é acessível a qualquer usuário. O RSA só será seguro se for difícil calcular d quando apenas n e e são conhecidos.

Na prática, só se sabe calcular d aplicando o algoritmo euclidiano estendido a $\phi(n)$ e e [39]. Por outro lado, para calcular $\phi(n)$ é necessário fatorar n para obter p e q. Logo, na prática, só será possível quebrar o código se for possível fatorar n. Contudo, para valores grandes de n, este problema é muito difícil de ser solucionado, uma vez que não se conhece algoritmo rápido de fatoração.

Sendo assim, é preciso lembrar que, se a escolha dos parâmetros p e q não for feita com cuidado, pode ser fácil quebrar o sistema RSA. Isso fica evidente se p e q forem muito pequenos. Todavia, mesmo para valores grandes de p e q pode haver problemas. Por exemplo, se p e q são muito grandes, mas se $| p - q |$ for pequeno, podemos achar p e q facilmente a partir do teorema de Fermat [39].

Felizmente, os sistemas criptográficos RSA podem utilizar números primos enormes; existem trabalhos publicados que prevêem números primos de até 2048 *bits* para aplicações militares e que exigem um elevado grau de segurança [39][40]. Neste trabalho optou-se por utilizar números primos de até 1024 *bits* de tamanho por existirem mais de 10^{200} números primos de até 512 *bits* (isto é bem mais que o número de átomos do universo) [17], garantindo uma segurança satisfatória para o cenário apresentado de emprego das redes sem fio.

Um dos objetivos deste trabalho é a implementação em *hardware* de sistema criptográfico, entre eles o RSA, de forma que o circuito integrado resultante atenda às necessidades de desempenho impostas pelas redes móveis.

Contudo, para atingir esse objetivo, optou-se pela implementação com uma chave de 1024 *bits* para o sistema criptográfico RSA, suficientemente robusta por não ser necessária a sua troca a cada transmissão de mensagem. Além disso, a complexidade das operações para realizar a geração das chaves do algoritmo criptográfico RSA justifica o emprego de *hardware* adicional.

3.1.3. Descrição comportamental

O RSA faz uso de exponenciação modular para cifrar, decifrar, gerar e verificar assinaturas. A exponenciação modular, contudo, é uma operação muito lenta, sobretudo quando operada por *software*. A implementação em *hardware* de circuitos dedicados a esse cálculo tem aumentado a velocidade do RSA, compatibilizando-a com as taxas de transmissão de dados das redes de computadores móveis.

Atualmente têm surgido diversas publicações sobre processadores dedicados ao algoritmo RSA, com variadas técnicas para melhorar seu desempenho. Os estudos com objetivo de aperfeiçoar a arquitetura e as variações dos algoritmos têm sido muito importantes para o aparecimento contínuo de novos circuitos integrados (CI) com um reduzido esforço computacional.

A exponenciação modular é geralmente implementada como somas de repetidas multiplicações em que é necessário o cálculo do módulo do resultado de cada produto. Todas essas operações precisam ser realizadas com números inteiros com no mínimo 1024 *bits*. Como quase todos os processadores de trabalho operam com palavras de no máximo 64 *bits*, os sistemas criptográficos desenvolvidos em *software* devem executá-las em operações de 64 *bits*. Fica clara a necessidade de se utilizar *hardware* dedicado para sistemas criptográficos de chave pública quando se deseja atingir altas taxas de processamento.

Uma forma alternativa de se calcular a exponencial modular é através do algoritmo H [15], que inicia o cálculo pelo *bit* mais significativo (MSB – *Most Significant Bit*), e que está descrito na Figura 5. Com ele é possível reduzir o complexo cálculo da exponenciação modular em seqüências de cálculo do módulo de produtos e quadrados.

Executa $P = M^e \mod N$

$P = 1$
$j = \log_2 e$
while $e > 0$ **do**
    if ($e \geq 2^j$) **then**
        $P = P \text{ x } M \mod N$
        $e = e - 2^j$
    end if
    $P = P \text{ x } P \mod N$
    $j = j - 1$
end while

Figura 5 – Algoritmo H (primeiro MSB)

Os parâmetros das chaves pública e privada (n, e) e (n, d) são representados pelo expoente E, que está descrito em binário por (e_{k-1}, e_{n-2}, …., e_1, e_0) e por N. A variável P denota o produto parcial do cálculo. Este algoritmo precisa de n iterações para realizar o cálculo, cada iteração composta de uma elevação ao quadrado e uma possível operação de multiplicação. Contudo, existe uma dependência entre o resultado da elevação ao quadrado e a sua utilização como operando do produto, e o resultado do produto serve como resultado parcial para a próxima iteração.

Outra forma de se computar o complexo cálculo da exponencial modular é por meio do algoritmo L [10], descrito na Figura 6. Em analogia ao H, o algoritmo L também utiliza sucessivos cálculos de multiplicações modulares para realizar o procedimento de cifragem e decifragem do RSA. Contudo, neste algoritmo, o cálculo é iniciado pelo *bit* menos significativo (LSB – *Least Significant Bit*).

```
Executa  P = M^e  mod N
P = 1
while e > 0 do
        if ( e is even ) then
                M = M x M  mod N
                e = e/2
        else  /*  e é impar */
                e = e –1
        end if
        P = P x M  mod N
end while
```

Figura 6 – Algoritmo L (primeiro LSB)

Os dois algoritmos executam, no pior caso, 2*k* operações, em que *k* é o número de *bits* do parâmetro *E*. Porém, como existem em média *k*/2 *bits* '1' no expoente *E*; o cálculo de uma simples exponencial modular necessita aproximadamente de 3/2*k* cálculos do módulo do produto nos dois algoritmos (L e H). Contudo, em *software*, é preferível o emprego do algoritmo H; uma vez que nele é necessário armazenar apenas um resultado parcial de variável longa, enquanto que no algoritmo L existe a necessidade de se armazenar o valor parcial de duas variáveis longas.

Todavia, foi possível modificar o algoritmo L [10], como descrito na Figura 7, para não existir dependência entre as operações de produto e quadrado, podendo ser executadas em paralelo. Dessa forma, seu desempenho em *hardware* é ainda melhor que na implementação do algoritmo H.

```
Executa  P = M^e  mod N
P = 1
For  j = 0 to n – 1
        if ( e(j) = '1' ) then
                P = P x M  mod N
        end if
        M = M x M  mod N
end loop
```

Figura 7 – Algoritmo L modificado para implementação em *hardware*

Cabe ressaltar que os algoritmos que executam o RSA reduzem a exponencial modular em iterações de produtos (multiplicação e quadrado) modulares. Sendo assim, a velocidade do sistema criptográfico é totalmente dependente da velocidade de execução dos circuitos que calculam o módulo do produto de grandes números inteiros. O ponto chave da implementação da exponenciação modular é o desenvolvimento de um algoritmo eficaz para a computação do módulo do produto de grande números inteiros com a menor quantidade de portas lógicas possível.

Existe uma variedade de algoritmos propostos para o cálculo do módulo do produto. Podem ser divididos em duas categorias: divisão depois da multiplicação e divisão durante a multiplicação. O primeiro requer mais espaço em memória, pois necessita dividir um número de 2*k bits* por outro de *n bits* para extrair o resto. O segundo método, em que cada passo do cálculo da divisão é feito por uma subtração durante a execução da multiplicação, requer mais operações que o anterior.

A Figura 8 mostra um algoritmo eficaz para o cálculo do módulo do produto [41], denotado por A x B mod N. Basicamente, este processo transforma o módulo do produto em simples operações de deslocamentos, somas e subtrações.

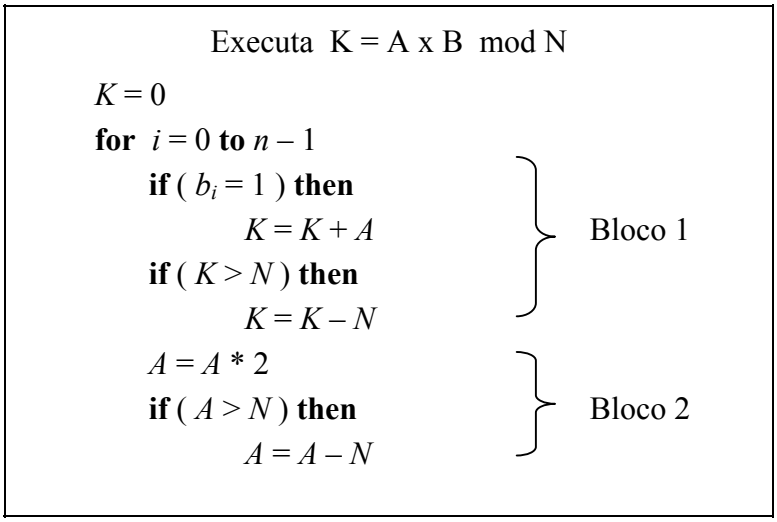


Figura 8 – Algoritmo módulo do produto (A . B mod N)

Neste algoritmo pode-se notar que há dois blocos distintos (bloco 1 e bloco 2) de processamento que podem ser executados em paralelo. Os dois blocos têm basicamente o mesmo número de ações. No primeiro (bloco 1), são executadas uma adição e uma subtração e, no segundo (bloco 2), são executados um deslocamento e uma subtração.

A idéia básica desse algoritmo é executar um deslocamento e uma subtração em paralelo com uma soma e uma subtração, como mostra a Figura 9. Uma vez que a subtração pode ser desenvolvida como soma, o mais crítico nesse algoritmo é a computação da soma de grandes números inteiros.

Como já foi visto anteriormente, o principal objetivo deste trabalho é implementar uma versão em *hardware* do sistema criptográfico RSA, de forma que o circuito integrado resultante atenda às necessidades de desempenho imposta pelas redes sem fio.

Contudo, para atingir esse objetivo, foi feito preliminarmente um estudo da divisão do trabalho de *hardware* e *software* para o sistema RSA para aplicações em segurança. Decidiu-se gerar por *software* as chaves de cifragem e decifragem. A principal razão para essa decisão é que as chaves não devem ser trocadas com freqüência para o tipo de aplicação empregada.

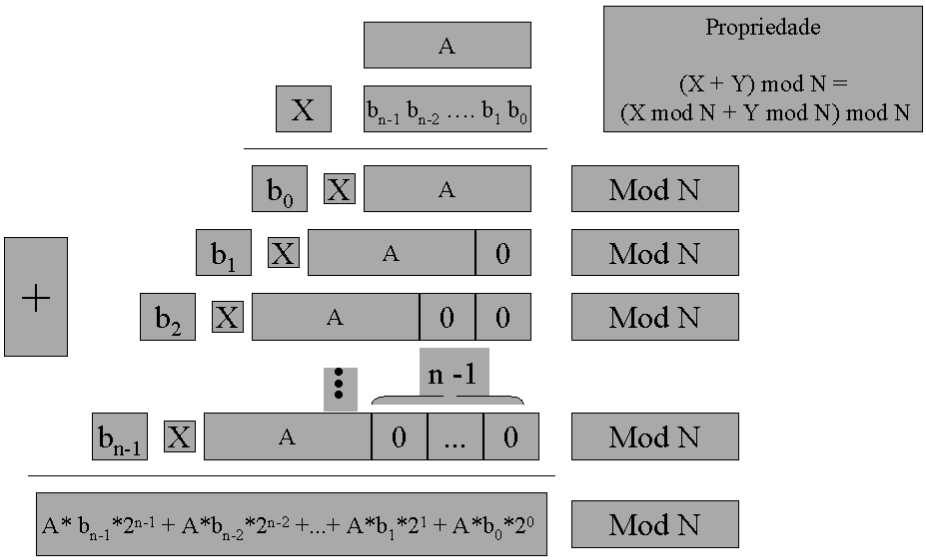


Figura 9 – Representação do módulo do produto (*A* . *B* mod *N*)

Com o compromisso de ter o melhor desempenho possível, dividiram-se as operações do sistema criptográfico RSA, como mostra a Figura 10, sem deixar de levar em consideração a área do circuito, a velocidade e o consumo de potência.

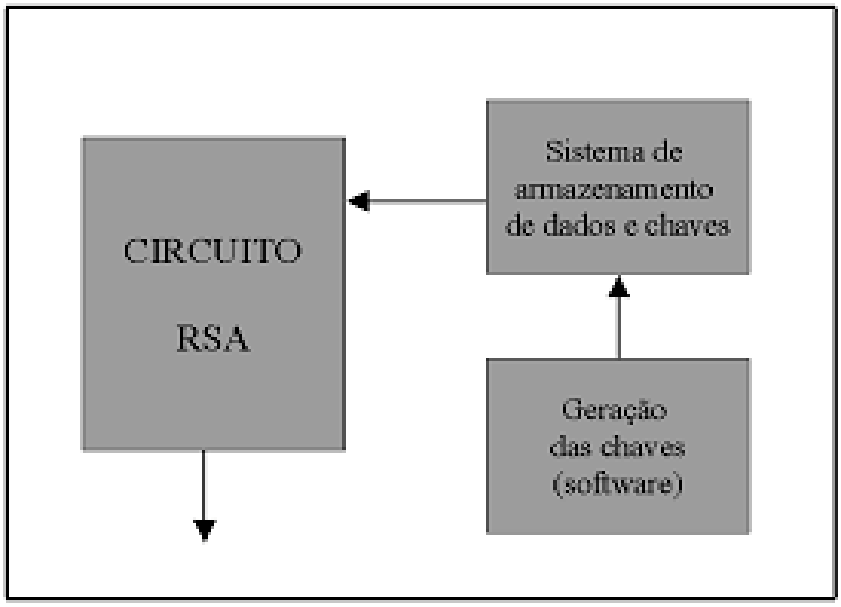


Figura 10 – Divisão entre *hardware* e *software*

Assim sendo, com base na definição do tamanho do módulo da chave em 1024 *bits*, foi definido o algoritmo, descrito na Figura 11, que implementa o RSA. Esse algoritmo é utilizado tanto na cifragem quanto na decifragem, uma vez que a função utilizada, descrita pelas equações 5 e 6 anteriormente definidas, é a mesma nesses dois casos.

As variáveis *A* e *B* representam os operandos da multiplicação modular de 1024 *bits* e *N* representa o módulo (chave), também de 1024 *bits*, dessa operação. As principais vantagens do desenvolvimento desse algoritmo são a possibilidade de realização do cálculo dos dois módulos do produto em paralelo, em função da não dependência dos seus resultados, e a realização de uma operação de multiplicação com operandos de 1024 *bits* sem a necessidade de haver resultados parciais de 2048 *bits*, o que aumentaria a área final do circuito.

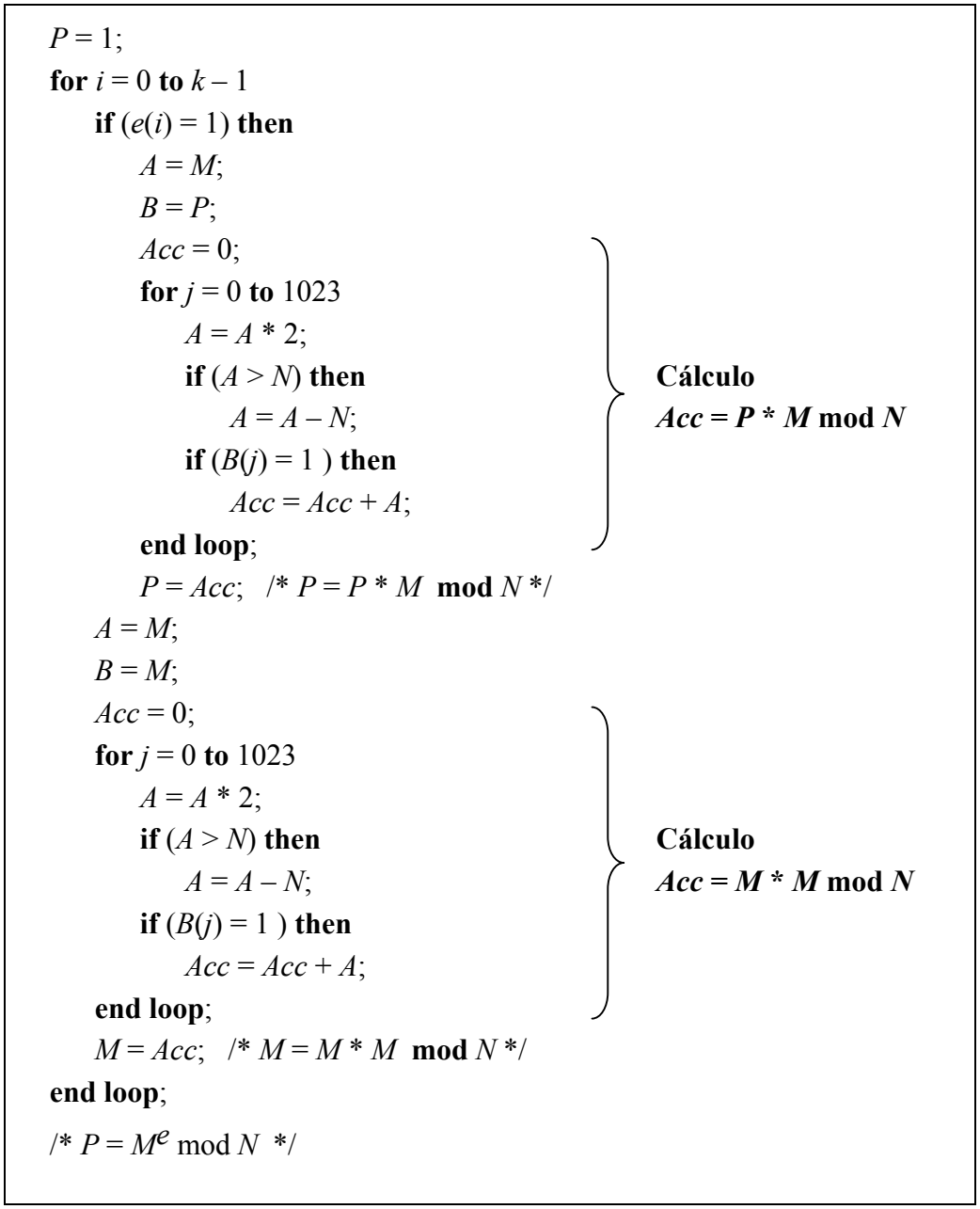


Figura 11 – Algoritmo RSA implementado

Contudo, esse algoritmo (Figura 11), que se apresenta eficiente para o cálculo da exponencial modular com base na computação do módulo do produto e no módulo do quadrado executados em paralelo, possui também uma desvantagem, que é a comparação de números de 1024 *bits* de magnitude. Esse problema foi resolvido neste trabalho por meio da execução da subtração por soma com complemento a menos 1. Com isso, o valor do *carry overflow* da operação informa se o minuendo é maior ou menor que o subtraendo. Colocando-se um multiplexador de duas entradas de 1024 *bits* cada uma e a sua seleção sendo realizada pelo *carry overflow*, obtém-se a comparação desejada.

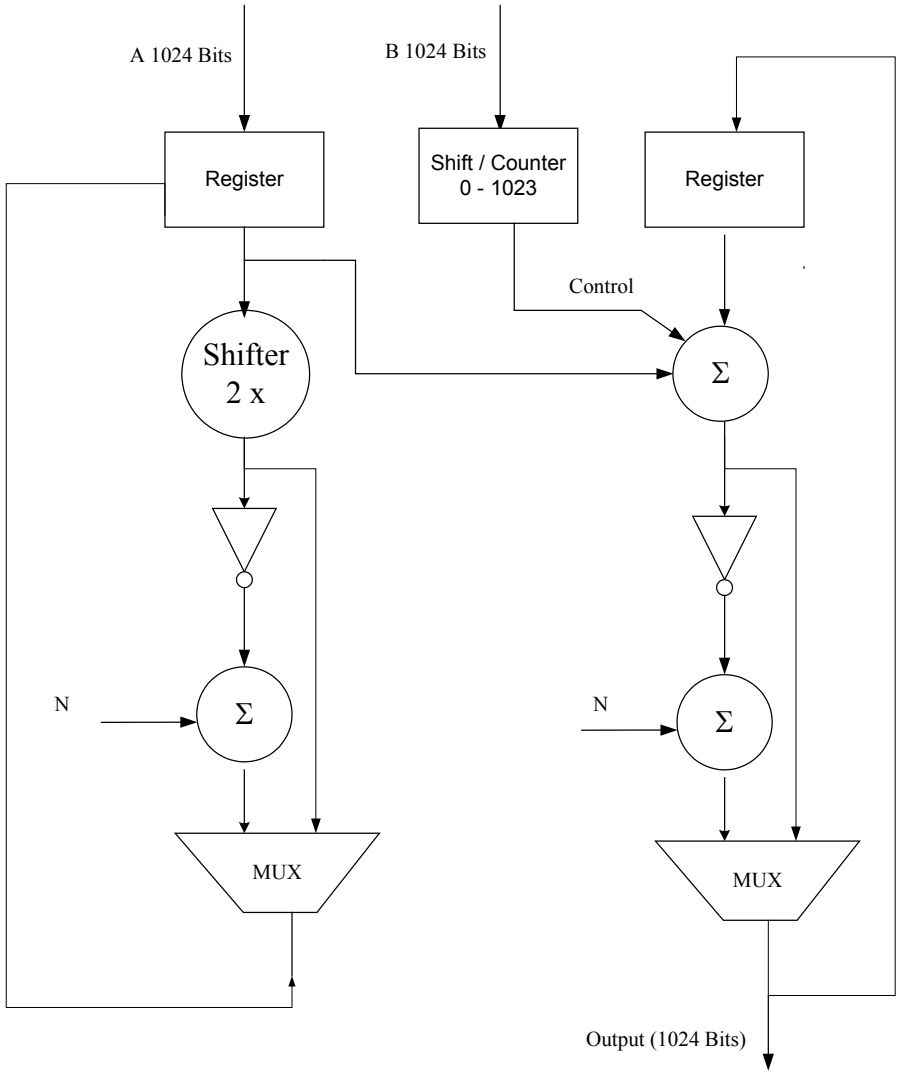


Figura 12 – Diagrama da multiplicação modular

A definição da arquitetura básica para o integrado RSA baseou-se no algoritmo definido na Figura 11. Contudo, analisando-se a complexidade da implementação direta desse algoritmo num modelo estrutural, optou-se por desenvolver o projeto em etapas, para facilitar sua conclusão. Esse fluxo de projeto foi fundamental na implementação deste trabalho. Para executar as operações dos algoritmos escritos anteriormente, a Figura 12 mostra o diagrama em blocos da multiplicação modular do RSA.

A primeira etapa é responsável pela definição das entradas e saídas do sistema. Esse passo forçou algumas definições e características para o sistema logo no início dos trabalhos. Fundamentado nessas características, fez-se a descrição do sistema em *software*, criando-se assim uma referência para testes futuros.

Depois de simulada a descrição em *software* e com as definições das

características e sinais de entrada e saída, criou-se a descrição comportamental e estrutural para o circuito usando a linguagem *Very High Speed Integrated Circuit Hardware Description Language* (VHDL). Nesta etapa foi fundamental a preocupação de se realizar descrições sintetizáveis. Isso exigiu maior cuidado, impedindo o uso de certas facilidades e recursos de programação que as linguagens de alto nível possuem.

Após o mapeamento na biblioteca de células alvo, foi necessária uma simulação do circuito. Essa simulação lógico-temporal forneceu apenas os atrasos das portas lógicas.

As etapas seguintes seriam o *placement*, o *routing* e o *back-annotation* do circuito. Contudo, essas etapas não foram realizadas por não haver interesse na fabricação do circuito final, mas apenas no seu projeto funcional.

3.1.4. Descrição estrutural

No desenvolvimento do projeto do circuito que implementa o RSA, escreveu-se o algoritmo escolhido para a sua implementação em linguagem de alto nível (utilizou-se a linguagem C). Essa etapa foi fundamental para ajudar na definição do modelo comportamental do RSA. Com base nesse programa, posteriormente foi escrito um programa em VHDL, ainda sem preocupação com a síntese do circuito, para desempenhar o papel de *golden device*, ou seja, a arquitetura que serviu de referência para o teste e validação do modelo final do projeto.

Após a finalização do *golden device*, implementou-se o teste lógico (*test bench)* relativo a esse modelo. Com a validação desse modelo, obtida por várias cifragens e decifragens correspondentes, de vários tipos e tamanhos de arquivos de dados e texto, obteve-se um padrão para posterior comparação com outras implementações em diferentes níveis de abstração.

A descrição do algoritmo escolhido foi realizada em linguagem C, ambiente Borland / DOS. Nesta fase do trabalho não houve preocupação com desempenho, mas apenas com a obtenção, ao seu final, de um algoritmo que fornecesse uma noção da complexidade computacional e que também pudesse ser usado em futuras comparações com o circuito final.

No desenvolvimento, foram utilizadas funções e rotinas (procedimentos), recurso muito comum no uso da linguagem C, adotado para tornar compatível com as próximas etapas de desenvolvimento (implementação comportamental e estrutural em VHDL), que tem uma estrutura hierárquica de componentes, em que cada um executa uma tarefa específica no conjunto final do sistema.

Com o programa em linguagem de alto nível bem consolidado e a arquitetura inicial bem definida, passa-se ao desenvolvimento do modelo comportamental do RSA em VHDL. Nesta fase, as entradas e saídas dos componentes passam a ser consideradas sinais elétricos, sendo essencial definir os tipos e funções de cada um deles. Todos esses sinais são materializados em pinos do componente principal RSA. A Figura 13 ilustra os pinos com os nomes dos sinais utilizados na descrição do circuito RSA.

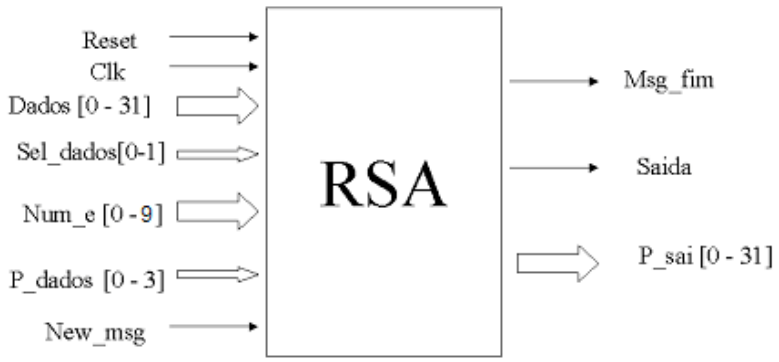


Figura 13 – Sinais de entrada e saída do circuito RSA

O total de pinos de sinais de entrada e saída é 85 (sem computar alimentação do circuito). Todos esses sinais são usados na interface entre o integrado RSA e o sistema ao qual ele estaria interligado. A Tabela 2 lista cada um desses sinais elétricos e explica resumidamente a funcionalidade dentro do componente.

Sinal Elétrico	Função	Entrada/Saída
Reset	*Reset* geral do sistema	Entrada
Clk	*Clock* do circuito	Entrada
Dados [0 – 31]	Vetor de 32 *bits* usado para carregar e montar o bloco de 1024 *bits* de mensagem, módulo da chave (*N*) e expoente da chave (*E* e *D*).	Entrada
Sel_dados[0 – 1]	Sinal de dois *bits*, para seleção de qual vetor será carregado (00 – mensagem, 01 – módulo da chave, 10 – Expoente da chave, 11 – fim do carregamento, dados prontos para o processamento).	Entrada
Num_e [0 – 9]	Vetor de 10 *bits* com a informação da posição do MSB do expoente *E* e *D*.	Entrada
P_dados [0 – 4]	Vetor de 5 *bits* que indica em que posição o bloco de 32 *bits* de dados lido entrará no vetor de 1024 *bits* correspondente.	Entrada
NEW_MSG	*Bit* indicativo para iniciar a carga de novo bloco de mensagem.	Entrada
MSG_FIM	*Bit* indicativo de fim de bloco de mensagem. Pronto para ser processado.	Saída
Saída	*Bit* indicativo de bloco processado, pronto para ser transferido para o sistema.	Saída
P_SAI [0 – 31]	Vetor de 32 *bits* usado para transferir o bloco de 512 *bits* da mensagem cifrada ou decifrada.	Saída

Tabela 2 – Sinais de entrada e saída do circuito RSA

A validação do algoritmo implementado se fez pela simulação lógico-temporal. Utilizaram-se os recursos do Synopsys, ferramenta de auxílio a projetos de circuitos, para executar a descrição do integrado em etapas pré-definidas com a finalidade de comparar os resultados parciais dessas etapas com os resultados equivalentes do programa em linguagem C.

Neste tipo de teste foram processadas várias seqüências de dados, em especial a seqüência: 10, 11, 12, 13, …, 1F, 20, 21, …, 2F, 30, 31, …, 3F, 40, 41, …, 4F, 50, 51, …, 5F, 60, 61, ..., 6F, 70, 71, …, 7F, 80, 81, …, 8F, 90, 91, ..., 9F, ..., F0, F1, ..., FF, representados em hexadecimal (tabela ASCII – *American Standard Code for Information Interchange*), sendo um total de 2048 *bits* divididos em dois blocos de 1024 *bits* cada. Por ser uma seqüência crescente de *bytes*, torna-se mais simples a percepção se a funcionalidade do algoritmo (cifrar e decifrar) for alcançada.

Depois de validada a descrição comportamental, passou-se ao desenvolvimento do modelo estrutural do integrado RSA. Com essa descrição pode-se gerar automaticamente o circuito, por meio de um processo de síntese.

Fundamentado no algoritmo proposto, a operação de exponenciação modular pode ser reduzida a quatro grandes blocos, como mostra a Figura 14. Dois desses blocos são muito semelhantes: o cálculo do módulo do quadrado e o cálculo do módulo do produto. Um terceiro bloco é responsável pela interface do sistema com o *chip* RSA; nele é feito todo o carregamento das chaves e dados que serão cifrados e decifrados, além dos controles entre sistema e circuito. O quarto bloco é o responsável pela contagem de zero até o *bit* mais significativo do expoente *E* (parte da chave), informando o fim da operação da exponenciação e, conseqüentemente, da cifragem ou decifragem do bloco de mensagens.

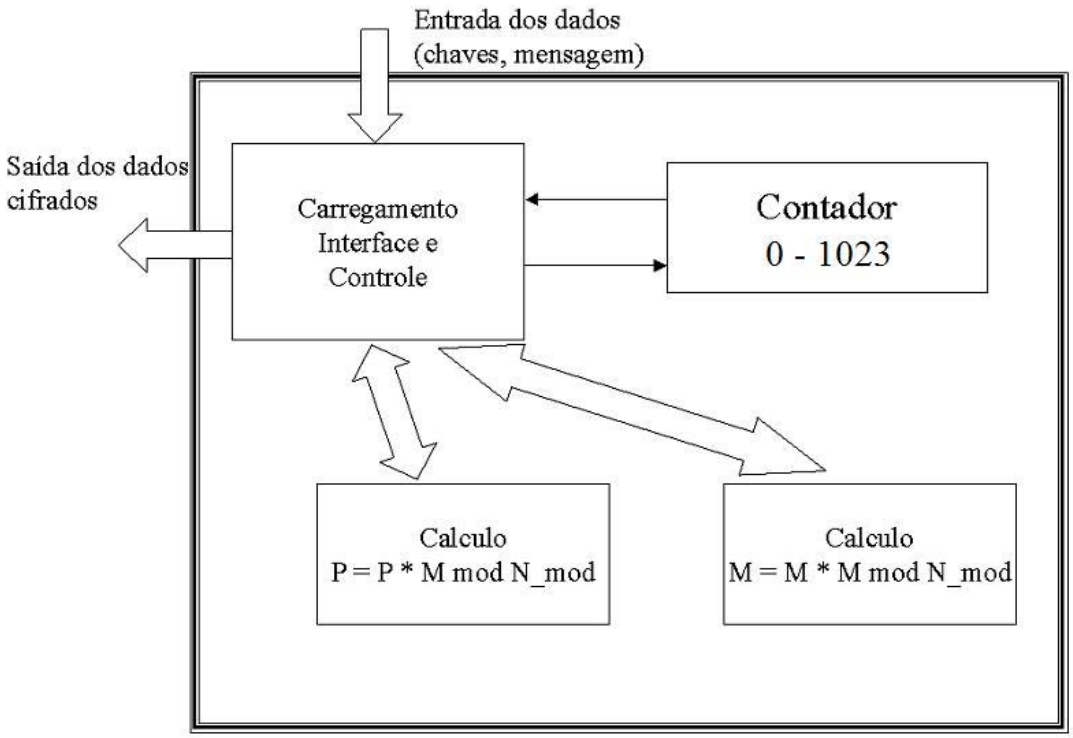


Figura 14 – Diagrama em blocos simplificado do circuito RSA

No final desta fase do projeto foi possível definir o funcionamento temporal do circuito. Contudo, cabe ressaltar que o funcionamento de todos os componentes do circuito são sincronizados pelo sinal *clock*. Na borda de subida do *clock* atualizam-se as entradas para que os dados possam ser processados. A partir desse momento, todo o

período do ciclo de *clock* serve para estabilizar a lógica dos componentes.

Deve-se notar que o período do ciclo de *clock* é delimitado pelo tempo necessário para a estabilização da lógica de todos os componentes do circuito, ou seja, o maior atraso esperado por um dos componentes do circuito do RSA é o período do sinal de *clock* e, conseqüentemente, determina a velocidade de processamento final do circuito.

3.1.5. Desempenho do circuito

Depois de concluída a etapa de testes funcionais e operacionais para o integrado RSA, passou-se a avaliar o desempenho desse integrado em termos de velocidade. A freqüência máxima de operação foi determinada pelo somador, que é a estrutura limitante em termos de velocidade. O somador, que é o bloco crítico, teve um atraso de 64 ns para 512 *bits* e um atraso de 172 ns para 1024 *bits*.

Sendo assim, a freqüência máxima de funcionamento foi de 15,6 MHz e 6 MHz. Com essas freqüências foram obtidas as seguintes velocidades de processamento para cifragem e decifragem, para chaves de 512 *bits e 1024 bits,* respectivamente:

- um bloco de 512 *bits* gasta cerca de 512 x 5 (número de componentes do cálculo do módulo do produto) x 64 ns (período *clock*) x 12 (número de *bits* típico para o expoente *E* da chave), o que resulta em uma taxa de 260 *kbits*/s.
- um bloco de 1024 *bits* gasta cerca de 1024 x 5 (número de componentes do cálculo do módulo do produto) x 172 ns (período *clock*) x 30 (número de *bits* típico para o expoente *E* da chave), o que resulta em uma taxa de 38 *kbits*/s.

Na Tabela 3 é possível avaliar as características do circuito RSA desenvolvido [41].

Características	Circuito (512 *bits*)	Circuito (1024 *bits*)
Módulo chave	512 *bits*	1024 *bits*
Taxa Cifragem	260 *kbits*	38 *kbits*
Número de portas	2×10^5	9×10^5
Metodologia de Projeto	*Standard Cell*	*Standard Cell*

Tabela 3 – Característica da implementação do circuito RSA

A Tabela 4 fornece uma comparação do RSA desenvolvido nesta tese (NOSSO) com outros trabalhos [11], [42], [43]. Nota-se que, em termos de desempenho, o circuito aqui projetado ficou no mesmo nível que esses trabalhos. Sendo inferior apenas ao RSA desenvolvido por [11].

Algoritmos	Nº de ciclos de *clock*
Chen *et al* [43]	4 *n*
Kornerup [42]	3 *n*
NOSSO	3 *n*
Wang *et al* [11]	$2n + \log(n + 1)$

Tabela 4 – Comparação de algoritmos de multiplicação modular

A comparação desta tabela é feita em relação ao tamanho da chave (*n*), ou seja, no caso do circuito RSA desenvolvido neste trabalho, são necessários 3 x 1024 ciclos de *clock* para realizar um cálculo de uma multiplicação modular de operandos de 1024 *bits*.

Pelos trabalhos apresentados na Tabela 4, é possível notar, também, que a parte central do circuito que executa o algoritmo criptográfico RSA é o cálculo do módulo do produto. Dessa forma, a maioria dos trabalhos publicados sobre este assunto compara o desempenho apenas desses multiplicadores.

Para executar essas multiplicações modulares, os circuitos que mais consomem área são os somadores. Contudo os circuitos RSA desenvolvidos que menos utilizam este tipo de componente [11], utilizam, em contrapartida, vetores de memória em forma de tabela, que pré-computam algumas operações e, conseqüentemente, aumentam

demasiadamente a área final do componente.

3.2. Algoritmo criptográfico assimétrico de curvas elípticas

O estudo das curvas elípticas é um importante ramo da Matemática, e a criptografia de curvas elípticas (ECC – *Elliptic Curve Cryptography*), proposta independentemente por Neil Koblitz em 1987 e por Victor Miller em 1986, é apenas uma aplicação dessa teoria [44].

Para a criptografia, torna-se particularmente interessante o estudo dessas curvas quando interceptam pontos de coordenadas (x, y) inteiras. As curvas elípticas podem oferecer algoritmos de chave pública que, em alguns casos, são mais rápidos e utilizam chaves bem menores, ao mesmo tempo em que oferecem altos níveis de segurança [45]. Os princípios da criptografia de curvas elípticas podem ser utilizados em adaptações de muitos algoritmos criptográficos, tais como Diffie-Hellman ou ElGamal [44].

Esta seção apresenta alguns conceitos importantes para aplicações em criptografia, como a definição da curva elíptica, do corpo finito e da base de operação.

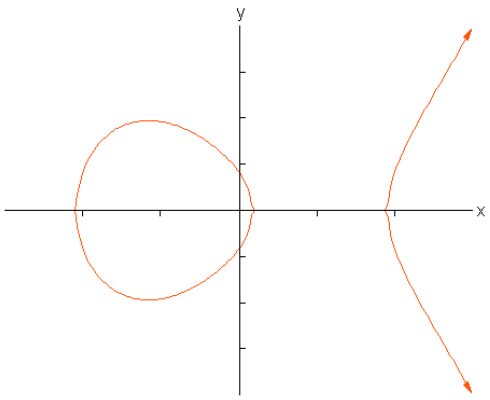
3.2.1. Funcionamento do ECC

3.2.1.1. Definição da curva elíptica

Curvas elípticas não são elipses. Elas têm esse nome porque são definidas como um objeto matemático (uma curva) descrito por uma equação cúbica [46], as mesmas que são usadas para calcular o comprimento de arco de uma elipse [47].

A equação simplificada de uma curva elíptica tem a forma da Equação 7. Um exemplo é mostrado na Figura 15, em que a curva elíptica é traçada em vermelho [48].

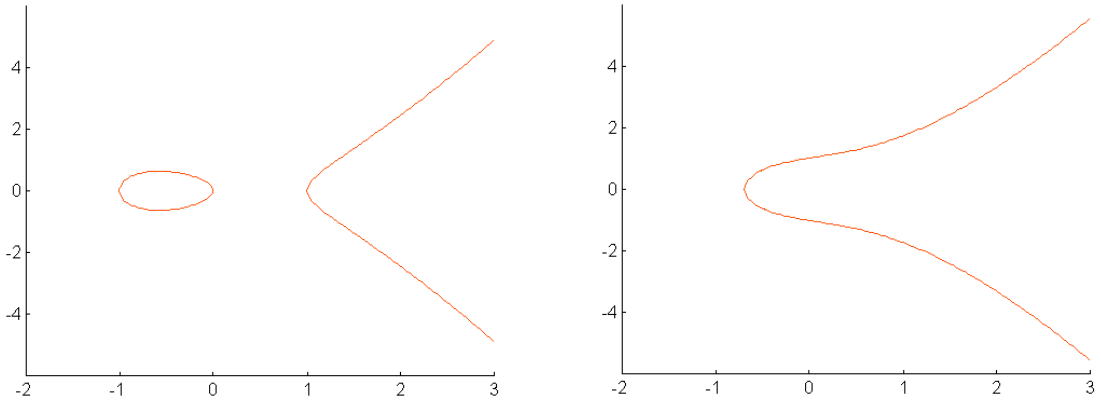
$$y^2 = x^3 + ax + b \qquad \text{Equação 7}$$



$$y^2 = x^3 - 4x + 0{,}67$$

Figura 15 – Exemplo de curva elíptica

Essas curvas podem assumir diferentes aspectos, a partir da variação dos parâmetros *a* e *b*. A Figura 16 permite visualizar variações de formatos de curvas elípticas decorrentes de alterações desses parâmetros [46].



(a) $y^2 = x^3 - x$ (b) $y^2 = x^3 + x + 1$

Figura 16 – Variações de curvas elípticas

Se uma curva elíptica como a apresentada na Equação 7 não contém fatores repetidos, ou, equivalentemente, se a desigualdade da Equação 8 é atendida, então a curva elíptica pode ser usada para formar um corpo ou campo, definido na seção 3.2.1.2 a seguir [44].

$$4a^3 + 27b^2 \neq 0 \qquad \text{Equação 8}$$

3.2.1.2. Definição de corpo finito e da base de operação

Subentende-se, neste trabalho, que corpo e campo possuem as mesmas propriedades matemáticas.

Um corpo ou campo é um conjunto de números para os quais as operações de adição, diferença, multiplicação e divisão – exceto o zero – estão definidos, e as leis comutativa, associativa e distributiva se aplicam [49]. Em outras palavras, um campo tem operações de adição, multiplicação e inversas aditiva e multiplicativa – exceto que zero não tem uma inversa multiplicativa. Essas operações sempre produzem um resultado que está no campo, exceção feita à divisão por zero, que é indefinida. Exemplos mais familiares de campos são os dos números reais, números complexos, números racionais (frações) e inteiros módulo um número primo, sendo o último um exemplo de corpo finito [50].

Um corpo finito consiste de um conjunto com uma quantidade finita de elementos, das operações de adição e multiplicação efetuadas sobre esses elementos, e das inversas aditiva e multiplicativa de cada elemento [49][51].

O número de elementos no corpo é a ordem do corpo finito. Existe um corpo finito de ordem q se e somente se q for um número primo. Um corpo finito é representado por F_q ou GF(q), onde q é o número de elementos. Se $q = p^n$, onde p é um primo e n é um inteiro positivo, representa-se por GF(p^n), sendo que p é a característica de GF(q) e n é o grau de extensão de GF(q) [51].

Um corpo finito pode ser primo, binário ou de extensão ótima.

Corpo finito primo é o corpo GF(p) que contém um número p primo de elementos. É também chamado corpo finito de característica p. Os elementos constituintes desse corpo são do tipo inteiro módulo p e as operações são realizadas sobre a aritmética de inteiros módulo p.

Corpo finito binário, também chamado corpo finito de característica 2, é o corpo GF(2^n) que contém 2^n elementos para algum grau de extensão n de GF(q). Os elementos desse corpo são uma cadeia de *bits* de tamanho n, e a aritmética realizada sobre esse corpo é a aritmética de operações sobre *bits*.

Corpo finito de extensão ótima é o corpo GF(p^n) onde p é um número primo de

Mersenne ($2^n \pm c$), para n e c positivos e arbitrários [49].

Definido o corpo de atuação, passa-se à representação da base de operações, onde se descreve a aritmética – especificando, primeiramente, como a cadeia de *bits* será interpretada. De forma similar à álgebra linear, define-se a base de um espaço vetorial como o conjunto de vetores independentes que o geram. Existem dois tipos de base: a base polinomial e a base normal.

Base polinomial é definida por um polinômio irredutível módulo 2, chamada de corpo polinomial. A seqüência de *bits* $a_{m-1}\ a_{m-2}\ ...\ a_2\ a_1$ representa o polinômio $a_{m-1}t^{m-1} + a_{m-2}t^{m-2} + ... + a_2t^2 + a_1t$ sobre o corpo GF(2). A aritmética é implementada como aritmética módulo $p(t)$, onde $p(t)$ é o corpo polinomial.

Já a base normal é a base da forma $\beta, \beta 2, \beta 2^2, ..., \beta 2^{2m-1}$, onde $\beta \in GF(2^n)$ [49].

3.2.1.3. Aplicações de curvas elípticas em criptografia

Ao se projetar um sistema de criptografia baseado em curvas elípticas, é necessário determinar, num primeiro momento, quais são as características gerais do sistema onde serão definidas todas as operações e parâmetros genéricos que todos os componentes utilizarão. Num segundo momento, cada um dos usuários desse sistema terá que definir seus parâmetros pessoais (chaves), de forma a viabilizar sua participação no sistema.

Ao se determinar as características gerais do sistema, é necessário realizar os seguintes passos [12][52]:

i) definir a natureza de seu campo finito F_q (F_p ou F_{2m});

ii) selecionar a representação para os elementos em F_q (polinomial, base ótima, subcampos etc.);

iii) implementar aritmética e operações em F_q;

iv) selecionar uma curva apropriada em F_q (quais parâmetros utilizar para a curva);

v) definir um ponto gerador em $E(F_q)$;

vi) definir o mapeamento da mensagem original em pontos de uma curva (*embedding*).

Os itens i, ii e iii citados acima já foram abordados durante a descrição de curvas elípticas sobre corpos finitos.

O item iv aborda o problema de como melhor escolher os parâmetros da curva elíptica, de forma a tornar seu sistema mais seguro. Existem técnicas que possuem o único propósito de auxiliar na escolha de parâmetros de curva apropriados. Entre essas técnicas incluem-se o método baseado no teorema de Hanssen, o método global, o método da multiplicação de complexos e o método randômico. A descrição desses métodos pode ser obtida em [12]. Discussões mais extensas sobre o que deve ser evitado e quais as *boas práticas* em termos de parametrização dessas curvas também podem ser vistas em [48].

Além da definição da própria curva $E(F_q)$ como parte dos parâmetros globais que devem ser mantidos públicos, está um ponto denominado ponto gerador [53] ou ponto base [48] $g \in E(F_q)$. Abordado no Anexo A, esse ponto é uma referência que permitirá a realização da criptografia. O ponto g é obtido a partir da escolha de um valor n primo grande tal que $ng = O$ (ponto no infinito) [48][53]. Todos os pontos $P_i \in E(Fq)$ têm uma ordem n_i, tal que, $n_iP_i = O$, dessa forma, o valor n é denominado ordem de g. Algo também importante é o fato de que n deve ser grande o suficiente de forma a inviabilizar a obtenção de todos os múltiplos de g: g, $2g$, $3g$, $4g$, ... , $(n-1)g$ [48].

Message embedding é um procedimento essencial ao uso da criptografia com curva elíptica, que consiste em uma forma de mapear a mensagem original (texto puro) em pontos de uma curva elíptica. Isso corresponde a colocar a mensagem original *sobre* a curva elíptica definida [54]. Esses pontos, depois de sofrerem operações parametrizadas pelas chaves individuais de um usuário, dão origem a um outro conjunto de pontos, que representa os pontos originais *cifrados*. Aqui, observa-se uma clara diferença entre os métodos de criptografia tradicional e a técnica baseada em curvas elípticas: ao invés de texto cifrado, são transmitidos um conjunto de *pontos cifrados* [46]. Esses *pontos cifrados*, ao serem recebidos pelo destinatário, são convertidos, através da chave correspondente, nos pontos originais. Nesse momento, ao aplicar a rotina de mapeamento invertida, recupera-se texto original e todo o *ciclo criptográfico* se fecha. O mapeamento de caracteres em pontos é citado em [46] e pode ser visto de

forma exemplificada em [54].

Uma vez estabelecidos todos os parâmetros e características gerais nas quais o sistema de criptografia com curvas elípticas deve se basear, basta que cada usuário determine seus parâmetros individuais, ou seja, seu par de chaves pública/privada, e os demais parâmetros locais necessários ao resto da implementação. Dessa forma, cada usuário possui um valor $n_A < n$ (conforme citado anteriormente, n é a ordem do ponto gerador g) como chave privada, que possui, como chave pública correspondente, um ponto $P_A = n_A g$. Observe que o ponto P_A é derivado de n_A e, assim como g, $P_A \in E(Fq)$.

Supondo que a mensagem original M (considera-se a mensagem como sendo de tamanho de 1 caracter) tenha sido mapeada em um ponto P_M, dentro do grupo finito de pontos de uma curva elíptica $E(F_q)$. A partir desse momento, quando um usuário A deseja enviar uma mensagem cifrada para B (usando a chave pública P_B), deve seguir o seguinte procedimento [46][53]:

A: Escolhe, aleatoriamente, um inteiro k.

A: Calcula, a partir de k, g, P_B e P_M, um par de pontos P_{C1} e P_{C2}:

$$P_{C1} = (kg)$$

$$P_{C2} = (P_M + kP_B)$$

A: Transmite para B o par de pontos cifrados P_C:

$$P_C = [P_{C1}, P_{C2}]$$

De outro lado, quando B recebe a mensagem (par de pontos P_C), recupera P_M, a partir do segundo ponto (P_{C2}), da seguinte forma:

B: Embora B não conheça k, sabendo que $kP_B = k\, n_B\, g$ (pois $P_B = n_B\, g$), calcula:

$$P_{C1}\, n_B = (kg)\, n_B = kP_B \;\Rightarrow\; kP_B \text{ } sai \text{ de } P_{C1}$$

B: Para extrair P_M de P_{C2}, basta calcular $P_{C2} - kP_B$, assim:

$$P_{C2} - P_{C1}\, n_B = (P_M + kP_B) - kP_B = (P_M + [(kg)\, n_B]\,) - [(kg)\, n_B]\,) = P_M$$

Esquemas similares para obtenção de texto cifrado, assim como assinatura digital em curvas elípticas, também podem ser encontrados em [12], [13], [46], [48] e [53].

3.2.1.4. Algoritmos criptográficos de curvas elípticas

Os sistemas de curvas elípticas estão sendo analisados em dois itens pelo *American National Standards Institute* (ANSI) ASC X9 (Financial Services): ANSI X9.62, *Elliptic Curve Digital Signature Algorithm* (ECDSA) e ANSI X9.63, *Elliptic Curve Key Agreement and Transport Protocols*.

Curvas elípticas estão também em análise pelo IEEE, padrão IEEE *P1363 (Standard Specifications for Public-Key Cryptography*), que inclui cifragem, assinatura e mecanismos de aceitação de chaves. Curvas elípticas sobre corpos finitos primos e sobre corpos finitos de característica dois são suportadas.

Os algoritmos criptográficos de curvas elípticas são assimétricos, pois utilizam chaves públicas e privadas. Entretanto, as mensagens são sempre cifradas com uma chave que é o produto da chave pública do destinatário pela chave privada do remetente, conforme descrito nas seções 3.2.1.3 e 3.2.1.4.2.

A geração de chaves é bastante simples, e é apresentada na seção 3.2.1.4.1.

Como todo algoritmo, também possui limitações, que estão apresentadas na seção 3.2.1.4.4, e sua segurança está na quantidade de operações necessárias para se obter o valor da chave privada.

3.2.1.4.1. Geração de chaves

Dois usuários *A* e *B* escolhem, de comum acordo, uma curva elíptica e um certo ponto F pertencente a esta curva. Isto pode ser feito publicamente.

Depois, em segredo, *A* escolhe como sua chave privada, um número aleatório A_S, que não precisa ser um ponto na curva, e calcula $A_P = A_S \cdot F$, onde A_P é sua chave pública.

Como mencionado no Anexo A, o ponto A_P pertence à curva, pois é um múltiplo de F, e é facilmente calculado.

B faz o mesmo, escolhendo um número aleatório B_S como sua chave privada e

obtendo sua chave pública B_P, calculada por $B_P = B_S \cdot F$ [44].

3.2.1.4.2. Cifragem e decifragem

Os usuários *A* e *B* divulgam suas chaves públicas. Eles dispõem agora de uma chave com a qual podem cifrar mensagens usando criptografia de chave privada. Esta chave é simplesmente o produto da chave pública de *A* pela chave privada de *B*, e equivale ao produto da chave privada de *A* pela chave pública de *B*.

Note-se que *A* e *B* podem calcular este produto depois de terem trocado suas chaves públicas, mas outras pessoas não podem, já que não conhecem nenhuma das chaves privadas [44].

3.2.1.4.3. Segurança da ECC

Para quebrar a chave, é necessário reconstruir uma das chaves privadas. Isto significa calcular A_S dados A_P e F, já que $A_P = A_S \cdot F$. E isto é muito difícil.

O número de pontos discretos na curva – pontos de coordenadas (x, y) inteiras – é chamado ordem da curva. Se a ordem do ponto F é um número primo de *n* bits, calcular A_S a partir do produto $A_S \cdot F$ e de F leva aproximadamente $2^{n/2}$ operações. Se F tem, por exemplo, o tamanho de 160 bits, serão necessárias cerca de 2^{80} operações. Se for possível executar um bilhão de operações por segundo, isto levaria cerca de 38 milhões de anos [44]. Este é um exemplo do problema do logaritmo discreto em curvas elípticas.

3.2.1.4.4. Limitações

Para violar os sistemas criptográficos, algoritmos específicos exploram alguns aspectos de fragilidade de cada sistema. Assim, tais aspectos devem ser evitados no projeto dos sistemas criptográficos.

Para os problemas da fatoração de inteiros e do logaritmo de números discretos, algoritmos específicos rápidos de quebra podem ser construídos para resolvê-los quando se utilizam fatores primos pequenos. De forma similar, para o problema de curvas elípticas, duas pequenas classes de curvas denominadas supersingulares e anômalas também apresentam aspectos de vulnerabilidade e, conseqüentemente, podem ser tratadas por algoritmos específicos [55].

O número de pontos de uma curva elíptica sobre um corpo finito é sempre um valor próximo ao número de elementos desse corpo [56]. Assim, uma curva elíptica E definida sobre GF(q) ou GF(p^n) de característica p tem aproximadamente q pontos sobre ela [57]. O número de pontos sobre E (número de soluções da equação da curva mais o ponto no infinito) é chamado ordem da curva, denotada #E [56], e satisfaz à seguinte desigualdade:

$$\#E \leq q + 1 - t \qquad \text{Equação 9}$$

onde $| t | \leq 2\sqrt{q}$ é o traço de Frobenius da curva. Uma curva é supersingular se o seu traço de Frobenius t é múltiplo da característica p do corpo sobre o qual a curva está definida, ou seja, se t divide p [56][57].

As propriedades das curvas supersingulares não as tornam indicadas para o uso em criptografia, pois o problema do logaritmo discreto pode ser reduzido ao problema do logaritmo discreto sobre um corpo de grau menor [54].

Uma curva elíptica E sobre um corpo finito GF(q) é chamada anômala se e somente se a ordem da curva é igual ao número de elementos que compõem o campo finito (tamanho do corpo sobre o qual está definida), ou seja [50][56][57]:

$$\#E = q \qquad \text{Equação 10}$$

O logaritmo discreto em curvas anômalas pode ser calculado em tempo polinomial, tornando-as curvas totalmente inadequadas para aplicações criptográficas [56][57].

O número de curvas supersingulares ou anômalas é extremamente pequeno em relação ao número total de curvas possíveis sobre o mesmo corpo finito [56]. Existem técnicas que auxiliam na escolha de parâmetros de curva apropriados [58].

Adicionalmente, como mencionado no Anexo A, se o polinômio $x^3 + ax + b$ não tiver raízes múltiplas, isto é, se o discriminante $4a^3 + 27b^2$ for diferente de zero, pode-se

definir uma lei de grupo [56], ou seja, a curva realmente forma um grupo.

Sobre curvas elípticas singulares ou degeneradas de equação geral $y^2 = x^3 - 3t^2x + 2t^3$, cujas raízes são t e $-2t$, onde t é raiz dupla se diferente de zero e raiz tripla se igual a zero, não se pode definir uma lei de grupo [56]. Estas curvas não podem ser usadas para criptografia.

3.2.1.5. Comparação com RSA

Segurança e eficiência são dois aspectos básicos utilizados para avaliação dos sistemas criptográficos de chave pública, como o RSA e o ECC. A Tabela 5 fornece algumas comparações entre esses dois sistemas.

A análise da segurança preocupa-se principalmente com a violação dos sistemas, que consiste em resolver o problema matemático em que se baseiam. Existem algoritmos genéricos que encontram a resposta para cada um dos problemas existentes [55]. A análise recai então no tempo necessário para chegar à solução, cuja ordem de grandeza permite avaliar o quanto um sistema de criptografia de chave pública é seguro [58]. Problemas de fatoração de inteiros e de logaritmos discretos admitem, em geral, algoritmos que executam em tempo subexponencial, enquanto que o problema dos logaritmos discretos em curvas elípticas executam em tempo puramente exponencial [55].

Característica	RSA	ECC
Sistema utilizado [58]	Sistema de fatoração de inteiros (IFS – *Integer Factorization Systems*)	Sistema de curva elíptica (ECDLS – *Elliptic Curve Discrete Logarithm System*)
Problema matemático em que se baseia [58]	Problema de fatoração de inteiros (IFP – *Integer Factorization Problem*)	Problema do logaritmo discreto em curvas elípticas (ECDLP – *Elliptic Curve Discrete Logarithm Problem*)
Segurança:		
Tempo para solução do problema com algoritmos genéricos [55]	Subexponencial	Puramente exponencial
Eficiência:		
Carga computacional [55]	10 vezes mais lento que o ECC	10 vezes mais rápido que o RSA
Tamanho da chave [55]	Pública = 1088 bits Privada = 2048 bits	Pública = 161 bits Privada = 160 bits
Tamanho de banda [55]	Maior	Menor
Parâmetros de sistema [59]	Nenhum	O campo finito, dois elementos que representam a curva, o ponto gerador *g* na curva e a ordem de *g*.
Chave pública [59]	O número primo *n* e o expoente *e*	O ponto P = *kg* na curva elíptica
Chave privada [59]	Expoente *d*	Um inteiro *k* onde 0 < *k* <

Tabela 5 – Comparação entre RSA e ECC

A Figura 17 apresenta o tempo necessário para violação de um sistema ECC em comparação com a aplicação RSA, onde o tempo é representado em número de anos que uma máquina, capaz de executar um milhão de instruções por segundo, leva para resolver o problema matemático. Atualmente, o tempo considerado como parâmetro de segurança razoável é de 10^{12} anos, alcançado pelo RSA com uma chave de 1024 *bits*, e pelo ECC, com uma chave de apenas 160 *bits* [52][55].

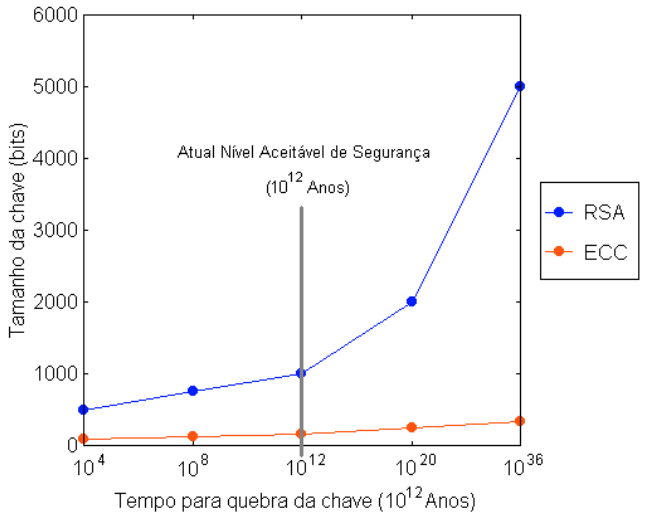


Figura 17 – Segurança e tamanho da chave

A análise da eficiência dos sistemas de criptografia considera carga computacional, tamanho de chave e tamanho de banda.

Considerando o mesmo nível de segurança para ambos os sistemas, o ECC executa aproximadamente 10 vezes mais rápido que o RSA.

Quanto ao tamanho da chave, verifica-se que o aumento do nível de segurança (tempo maior) necessita de um aumento bem mais significativo do tamanho da chave no sistema RSA, em comparação com o ECC. Analisando de outra forma, é possível utilizar chaves significativamente menores com o ECC para obter um nível de segurança equivalente com o RSA. O ECC oferece melhor segurança por bit que qualquer outro sistema conhecido de chave pública [4].

Largura de banda corresponde a quantos bits a mais devem ser transmitidos, além da mensagem original, após o processo de cifragem ou assinatura. Após cifrar uma mensagem de 100 bits, o ECC apresenta uma mensagem codificada de 321 bits, contra 1024 bits do RSA.

3.2.2. Implementação

A implementação do algoritmo criptográfico ECC assume que a mensagem é mapeada em pontos numa curva elíptica. Essas codificação e decodificação foram descritas anteriormente e podem ser melhor detalhadas em [60]. Contudo, neste

trabalho, a versão implementada do algoritmo criptográfico ECC não contém a implementação da codificação e decodificação de pontos mapeados nas curvas elípticas, pois essas codificações acrescentam um pequeno fator constante no desempenho global do sistema ao cifrar e decifrar.

Sendo assim, a implementação do algoritmo de curvas elípticas, neste trabalho, limitou-se a desenvolver circuitos que realizassem operações de multiplicações modulares de dois pontos distintos, aproveitando o êxito alcançado na implementação de alguns circuitos do algoritmo criptográfico RSA. As operações realizadas na implementação desses algoritmos foram multiplicações modulares, dobradores, inversores e comparadores.

De forma simplificada, a implementação do algoritmo ECC é apresentada como uma adição de pontos. Quando os pontos a serem somados são os mesmos, a operação de adição de ponto passa a ser substituída pela operação dobro de um ponto; quer dizer, em vez de se usar Q = P + Q a operação que deve ser usada é Q = 2P.

3.2.3. Resultados

O desempenho do algoritmo criptográfico ECC foi obtido a partir da definição do número de ciclos do *clock* exigido para execução das operações e do período do *clock*. O período mínimo do *clock* é função do atraso do caminho crítico do multiplicador, circuito mais complexo. Como essa implementação foi fundamentada nos circuitos desenvolvidos para o algoritmo RSA, adaptado para o algoritmo ECC com chave de 163 *bits*, obteve-se uma freqüência de *clock* de 38 MHz, com uma taxa de aproximadamente 300 kbps, para uma mensagem de 1024 *bytes*, com um atraso próximo a 19 ms. Certamente esses resultados serão melhorados quando for realizado um circuito específico para as operações do algoritmo criptográfico de curvas elípticas.

3.3. Considerações

Com base nas implementações dos algoritmos criptográficos assimétricos RSA e ECC obteve-se o desempenho e a área aproximada, em número de portas, de cada circuito. A comparação de desempenho entre esses algoritmos criptográficos torna-se difícil, uma vez que são diferentes nos níveis de segurança e no tamanho das chaves. Contudo, pode-se avaliar o desempenho de cada algoritmo na execução de um mesmo bloco de mensagem de 1024 *bytes* de tamanho. Dessa forma, os desempenhos puderam ser comparados, por meio dos tempos de processamento dessa mensagem comum a todos, através dos seus atrasos.

Pôde-se constatar, também, que as velocidades de processamento obtidas para cada algoritmo foram equivalentes às apresentadas em outros trabalhos anteriormente publicados [10][11][13][14][15][42][43][61][62]. Nos trabalhos [11] e [15], em suas implementações para o algoritmo RSA com chave de 1024 *bits*, foram obtidas taxas de desempenho totalmente compatíveis com as obtidas neste trabalho, de 38 *kbit*/s. As comparações dessas implementações foram feitas em relação ao tamanho da chave (n). Assim, são necessários, no caso do circuito RSA desenvolvido neste trabalho, 3 x 1024 ciclos de *clock* para realizar um cálculo de uma multiplicação modular de operandos, enquanto que, nos trabalhos [11] e [43], são necessários, $2n + \log(n + 1)$ e $3n$, respectivamente.

As implementações do algoritmo ECC realizadas em [14] e [62] atingiram uma *performance* um pouco melhor que a obtida neste trabalho, uma vez que não foram implementadas todas as etapas do algoritmo ECC, tendo sido adaptado do circuito RSA o cálculo da exponencial modular.

Capítulo 4

Implementação de algoritmos criptográficos simétricos

Neste Capítulo são apresentadas descrições do funcionamento dos algoritmos criptográficos Blowfish e Godzuk, com suas respectivas implementações em *hardware*. Considerando-se os aspectos referentes ao desempenho de tais algoritmos, permitiu-se proceder a um estudo comparativo entre esses dois algoritmos. Os resultados obtidos podem servir para auxiliar a escolha de algoritmos criptográficos em aplicações que necessitem, sobretudo, de aspectos críticos de desempenho, uma vez que, essa é a principal característica desses algoritmos quando comparados com os algoritmos assimétricos.

4.1. Algoritmo criptográfico simétrico Blowfish

O algoritmo criptográfico simétrico Blowfish é outra implementação relevante realizada neste trabalho. Esse algoritmo possui algumas vantagens quando comparado com outros semelhantes, como o DES e o IDEA, sendo as principais o fato de não ser patenteado, permitindo que sua licença esteja à disposição de toda a comunidade, e ser

considerado um algoritmo inquebrável, uma vez que ainda não são conhecidos ataques contra ele. Já foram examinadas chaves fracas no Blowfish [63], indicando que existe uma quantidade pequena de chaves que podem ser detectadas – mas não podem ser quebradas.

O projeto resultante foi chamado *Software Core for the Blowfish Cryptographic Algorithm* (SCOB) [64], que obteve desempenho compatível em termos de segurança e taxa de cifragem das redes de telecomunicações [27][46].

4.1.1. Funcionamento do Blowfish

O Blowfish utiliza um grande número de subchaves na execução do algoritmo. Estas podem ser computadas antes das operações de cifragem ou de decifragem. Cabe ressaltar que o algoritmo Blowfish normalmente é empregado em operações em que a chave não é modificada com freqüência [63] , como é desejável em redes sem fio.

As subchaves são da seguinte forma:

a) as subchaves *P* consistem em 18 subchaves de 32 *bits*, P_1, P_2,...., P_{18}.

b) existem quatro caixas *S* de 32 *bits* por 256 posições:

$S_{1,0}$; $S_{1,1}$; ... $S_{1,255}$;

$S_{2,0}$; $S_{2,1}$; ... $S_{2,255}$;

$S_{3,0}$; $S_{3,1}$; ... $S_{3,255}$;

$S_{4,0}$; $S_{4,1}$; ... $S_{4,255}$;

fornecendo, assim, 1042 subchaves de 32 *bits*.

O método exato para o cálculo destas subchaves pode ser encontrado em [27].

4.1.2. Cifragem e decifragem.

Como mencionado anteriormente, o algoritmo criptográfico Blowfish é baseado em redes de Feistel, constituindo 16 fases, opera na forma apresentada na Figura 18, e sua estrutura é apresentada na Figura 20 (a).

Seja *X* o bloco de dados de entrada de 16 *bits*:
Inicialmente divide-se o bloco *X* em dois sub-blocos de 32 *bits*: *XL* e *XR*;
Para $i = 1$ até 16;
$XL = XL$ XOR P_i;
$XR = F(XL)$ XOR XR
Troque *XL* com *XR;*
Se for a última fase, troque novamente *XL* com *XR*;
$XR = XR$ XOR P_{17};
$XL = XL$ XOR P_{18}; *XL* e *XR* são novamente combinados em um novo bloco de 64 *bits*.

Figura 18 – Algoritmo da rede de Feistel

A função F está definida na Figura 19, e opera da forma apresentada na Figura 20 (b).

Inicialmente divide-se *XL* em quatro sub-blocos de oito *bits*: *a*, *b*, *c* e *d*.
$F(XL) = ((S_{1,a} + S_{2,b} \bmod 2^{32})$ XOR $S_{3,c}) + S_{4,d} \bmod 2^{32}$.

Figura 19 – Algoritmo da função F

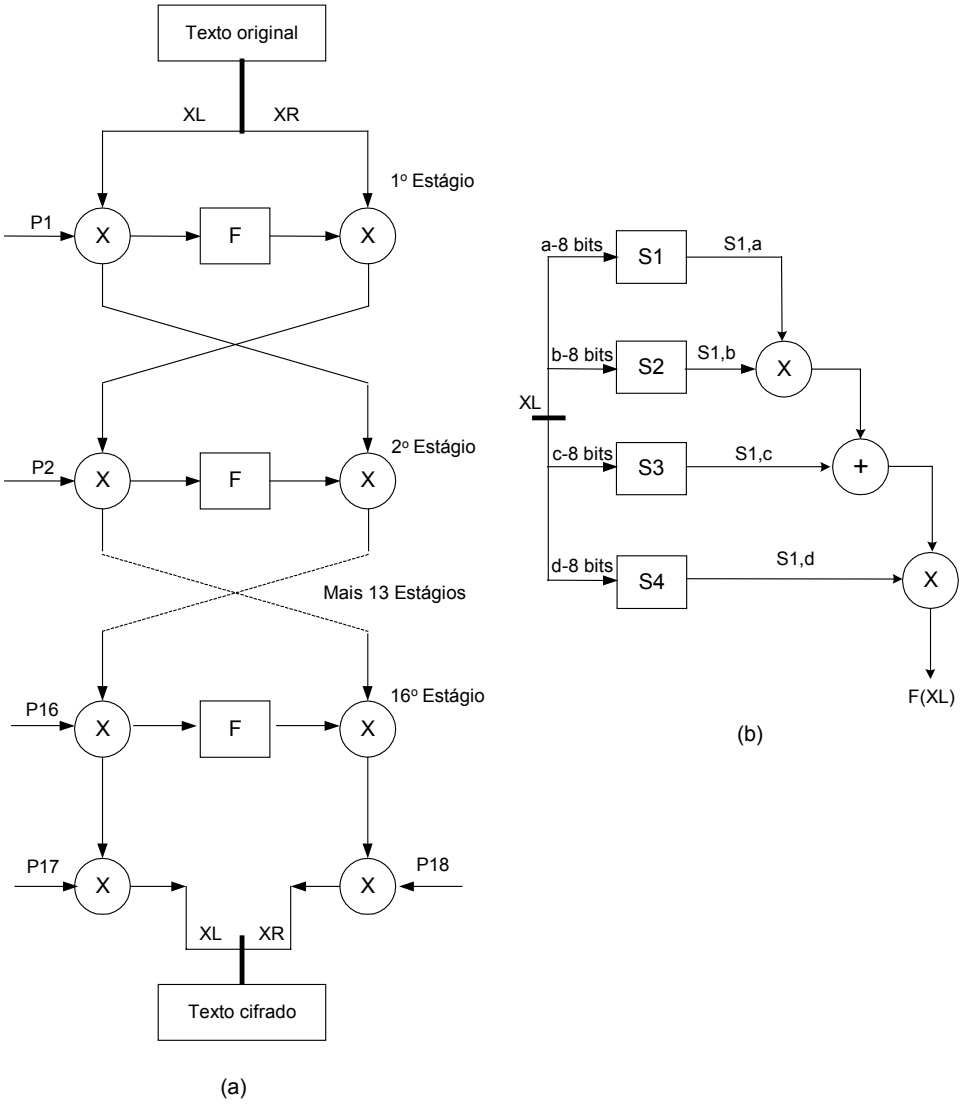


Figura 20 – Estrutura da rede de Feistel

A decifragem é feita da mesma forma que a cifragem, com exceção das subchaves P_1, P_2, ..., P_{18}, que são aplicadas em ordem reversa.

4.1.3. Implementação

Como o *Internet Engineering Task Force* (IETF) ainda não havia definido, à época da implementação desse algoritmo, as características dos subsistemas de segurança das redes sem fio, em termos de desempenho, área e consumo, este trabalho, então, teve a preocupação de buscar um desempenho maior do que o exigido por essas redes [65][66], com pequena área e, conseqüentemente, um baixo consumo de energia,

para atender aos possíveis requisitos dessas redes.

Dessa forma, procurou-se explorar os possíveis paralelismos temporais e espaciais existentes na estrutura do algoritmo, obtendo-se a estrutura *pipeline* mostrada na Figura 21.

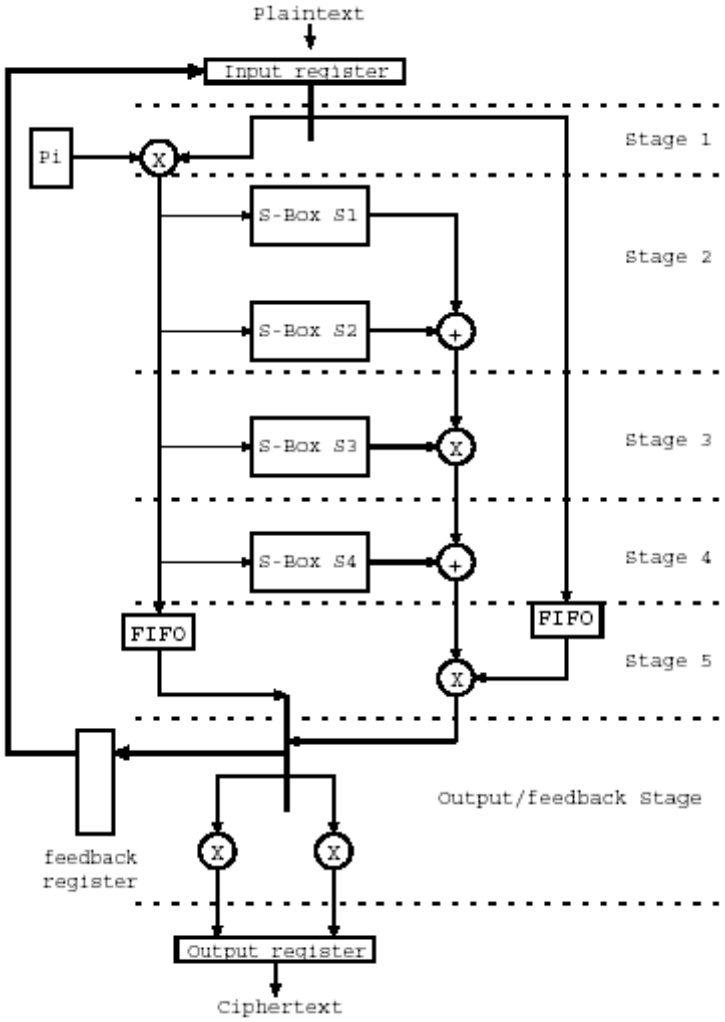


Figura 21 – Estrutura *pipeline*

As operações contidas em cada estágio do *pipeline* podem ser executadas em um único ciclo de máquina. As filas (estruturas *First In First Out* – FIFO) foram introduzidas de forma que pudessem sincronizar o funcionamento da estrutura em *pipeline*.

De maneira a facilitar a implementação do circuito em vários tipos de dispositivos de *hardware*, que normalmente apresentam limitações em termos de circuito, optou-se por implementar somente uma das fases do algoritmo em *hardware*, e

repeti-la por meio de iterações.

Para melhorar o desempenho do algoritmo Blowfish no SCOB, as subchaves de cifragem e decifragem são armazenadas em memórias e registradores.

Substituindo os blocos operacionais na estrutura *pipeline* por estruturas em *hardware* (somadores de 32 *bits*, ou-exclusivos, memórias etc.) e introduzindo na estrutura dispositivos de sincronismo como FIFO's, *tri-states* e *flip-flops*, obteve-se a arquitetura do SCOB, que é mostrada na Figura 22.

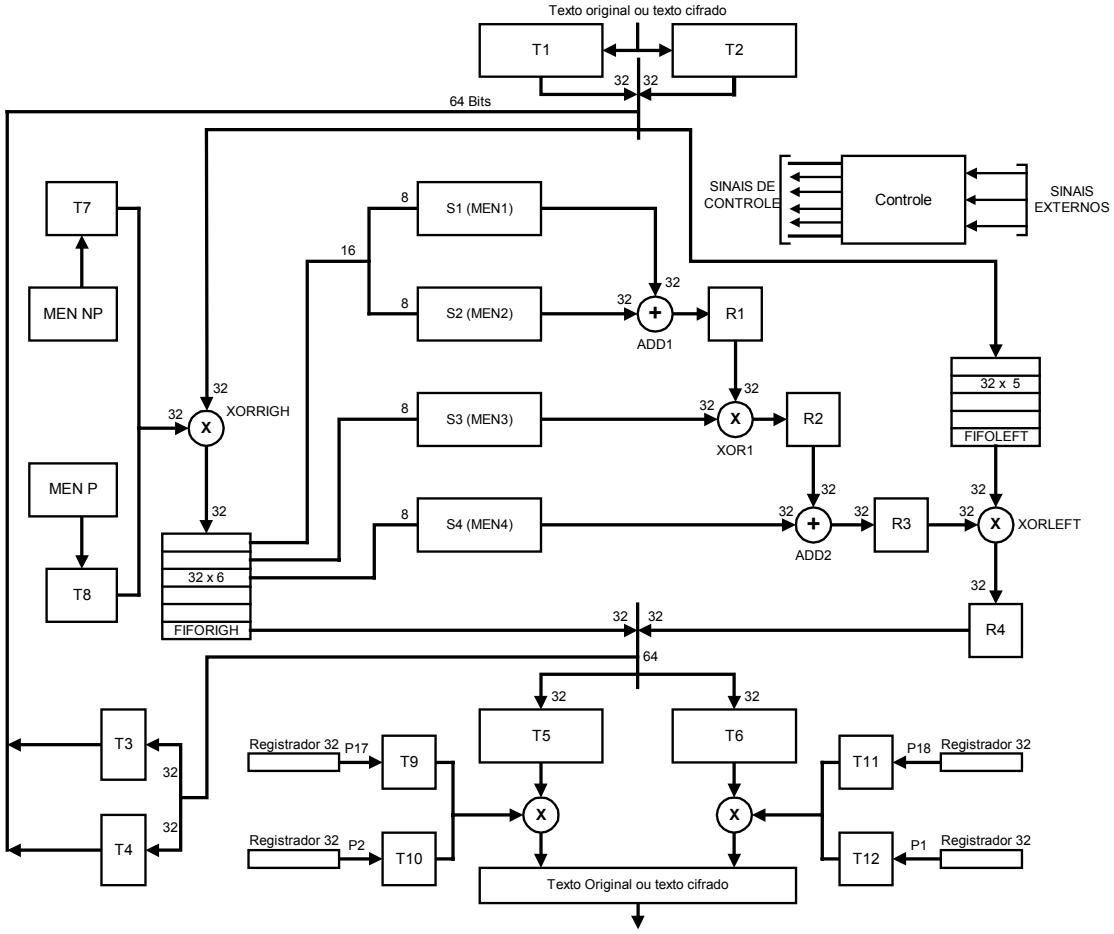


Figura 22 – Arquitetura do SCOB

A unidade de controle é responsável pelo gerenciamento do fluxo de dados na estrutura (preenchimento correto do *pipeline*, entrada e saída de dados, realimentação de dados etc.), pelo controle das memórias e pelo controle do processo (cifragem ou decifragem).

Essa arquitetura foi descrita em linguagem VHDL estrutural, com um código resultante de aproximadamente 1700 linhas e, posteriormente, pôde ser sintetizada em

FPGA e com células pré-caracterizadas ou em ASIC.

4.1.4. Resultados

A funcionalidade do SCOB foi validada a partir da comparação de resultados da simulação de sua descrição em VHDL estrutural, com os resultados do programa fonte em linguagem C do Blowfish. Para isso foram usados diversos blocos de dados diferentes e diversas chaves diferentes.

A implementação em forma de ASIC foi sintetizada sobre a biblioteca de células básicas 0,7 μm CMOS. Essa implementação resultou em uma freqüência de operação típica de 50 MHz. Com essa freqüência de operação, as dezesseis iterações necessárias para executar o algoritmo SCOB processando seis blocos de 64 *bits* por vez (*pipeline*) apresentaram uma taxa de cifragem de 200 *Mbits*/s. A implementação em forma de ASIC resultou em 4620 *Standard Cells*, ou seja, aproximadamente 43280 transistores, quatro memórias de acesso aleatório (RAM – *Randomic Access Memory*) de 32 *bits* por 256 posições e duas memórias RAM de 32 *bits* por 16 posições.

A descrição VHDL do SCOB foi igualmente mapeada em um *Erasable Programmable Logic Device* (EPLD) da família FLEX10K da Altera, mais especificamente no EPLD EPF10K250AGC599-1, devido às seis memórias RAM constantes da arquitetura. Essa implementação resultou em uma freqüência de operação de 10 MHz, com taxa de cifragem de 40 *Mbits*/s.

A Tabela 6 fornece um resumo das características dessas implementações.

Resultados	ASIC	FPGA
Máximo retardo	20 ns	100 ns
Tempo total para uma cifragem	240 ns	1600 ns
Freqüência de operação	50 MHz	10 MHz
Taxa de cifragem	200 *Mbits*/s	40 *Mbits*/s

Tabela 6 – Desempenho Blowfish

4.2. Algoritmo criptográfico simétrico Godzuk

O algoritmo Godzuk foi desenvolvido em conjunto no Laboratório de Projetos de Circuitos da COPPE – UFRJ [27] [60] com base no algoritmo Godzilla, para ser utilizado na terceira geração de celulares, e de acordo com as normas do 3GPP [1], uma vez que suas características de alto desempenho e reduzida área permitem seu emprego, sobretudo em telefonia celular, como um possível padrão de segurança [60]. Pode também ser implementado e aplicado em redes móveis.

Por ser uma versão do Godzilla, o algoritmo criptográfico Godzuk é também um algoritmo simétrico que opera com blocos de dados de 64 *bits* e chave de 128, 256, 512 ou 1024 *bits*. O algoritmo é executado em oito fases de rede de Feistel [67], executadas em três níveis de operações internas, sendo a última baseada em funções *Scheduling by Multiple Edge Reversal* (SMER) [68].

4.2.1. Funcionamento do Godzuk

4.2.1.1. Redes de Feistel

Vários algoritmos criptográficos são baseados no conceito de redes de Feistel [67]. Esse conceito data do início dos anos 70 e funciona da seguinte forma: supõe-se um bloco com *n bits* (*n* par) que sofrerá uma operação de cifragem. Esse bloco é dividido em dois sub-blocos de *n*/2 *bits*, bloco L e bloco R.

A estrutura básica das redes de Feistel está baseada na possibilidade de se definir uma operação de cifragem de blocos, onde a saída da *i*-ésima fase de iteração é dependente da saída da fase anterior, ou seja:

$$L_i = R_{i-1} \qquad \text{Equação 11}$$

$$R_i = L_{i-1} \text{ XOR } F(R_{i-1}, K_i) \qquad \text{Equação 12}$$

onde K_i é a subchave usada na *i*-ésima fase do algoritmo e F é uma função arbitrária utilizada em cada fase.

Para que as redes de Feistel funcionem, é necessário que exista a inversibilidade das operações, ou seja, que a função F apresente a seguinte característica:

$$L_{i-1} \text{ XOR } F(R_{i-1}, K_i) \text{ XOR } F(R_{i-1}, K_i) = L_{i-1} \qquad \text{Equação 13}$$

Esse conceito vem sendo largamente utilizado ao longo do tempo em diversos algoritmos criptográficos, tais como DES [23], IDEA [24], FEAL [25], RC2 [17] e Blowfish [26].

4.2.1.2. Escalonamento por reversão múltipla de arestas – SMER

O conceito de escalonamento por reversão múltipla de arestas (SMER – *Scheduling by Multiple Edges Reversal*) é baseado na teoria de grafos e é uma tentativa de generalização do *Scheduling by Edges Reversal* (SER) [69].

Para o caso do SER, a representação de um circuito (ou sistema) por meio de um grafo G(*N*, *E*), onde *N* representa o número de subsistemas e *E* representa o número de arestas que significa a topologia de interconexão, pressupõe que só exista uma aresta interligando dois nós quaisquer. Neste caso, um nó qualquer se torna ativo quando todas as arestas que partem do mesmo estão revertidas para ele. Contudo, a interligação entre este e outro nó só pode ser feita por uma única aresta.

A Figura 23 apresenta um grafo do tipo SER com sua topologia de interconexão, ou seja, dois nós quaisquer do grafo estão interligados por apenas uma única aresta, ilustrando o funcionamento descrito anteriormente. Os nós hachurados representam os nós ativos no momento, ou seja, os nós N_3 e N_5 possuem todas as suas arestas revertidas para si próprios.

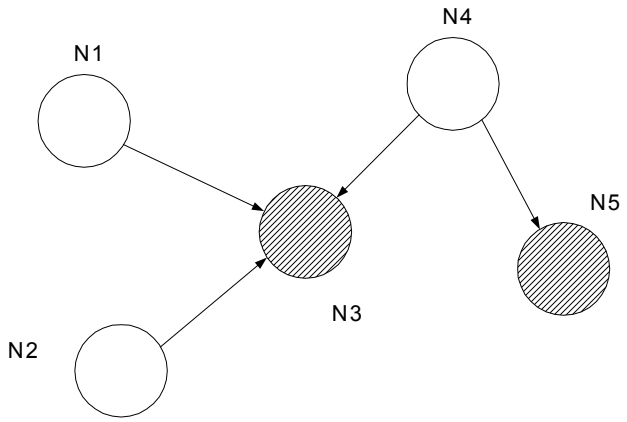


Figura 23 – Representação do grafo SER

O SMER é a generalização desse conceito. Para tanto, o grafo G(N, E) agora pode ter, entre dois nós, mais de uma aresta, ou seja, no SMER é possível ter diversas arestas interligando dois nós, como é ilustrado pela Figura 24, onde N_a significa o número de arestas que interligam os dois nós.

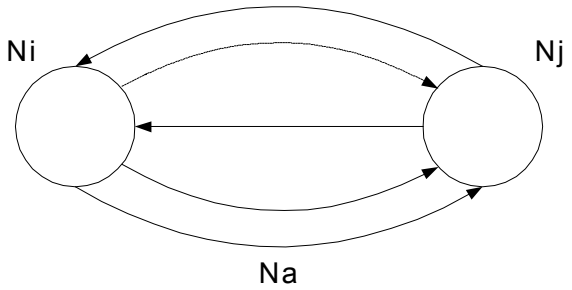


Figura 24 – Representação do SMER

4.2.1.2.1. Considerações iniciais

O funcionamento do SMER é bastante parecido com o do SER, porém deve-se atentar para algumas características peculiares do primeiro. Para facilitar o entendimento, percorre-se, passo a passo, o funcionamento de um exemplo inicial com dois nós. Consideram-se dois nós N_1 e N_2, interligados por duas arestas, conforme Figura 25. Para que o nó N_1 se torne ativo, é preciso que as duas arestas estejam revertidas para o mesmo. Para que o nó N_2 se torne ativo, apenas uma das arestas

precisa estar voltada para o mesmo. Quando ativo, N_1 executa suas operações internas e reverte as duas arestas logo em seguida. Já N_2, após a execução de suas operações internas, reverte apenas uma das arestas.

Supondo-se uma orientação inicial das duas arestas apontando para N_2, Figura 25 (a), o nó N_2 se apresenta ativo e, após a execução de suas tarefas, reverte uma de suas arestas, Figura 25 (b).

Apesar da reversão de uma das arestas, o nó N_2 continua ativo, já que a outra aresta continua apontando para o mesmo. Assim sendo, o nó N_2 executa novamente suas operações internas e reverte a outra aresta, conforme ilustrado na Figura 25 (c). A partir desse momento, N_1 passa a ser um nó ativo, uma vez que duas arestas apontam para o mesmo. N_1 reverte duas arestas após executar suas operações internas, Figura 25 (d), alcançando, então, a configuração inicial, a partir da qual executa-se um novo ciclo, Figura 25.

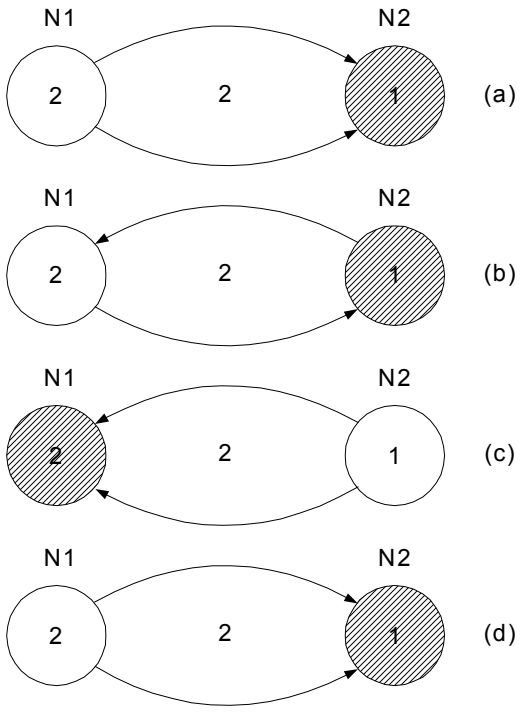


Figura 25 – Exemplo de funcionamento do SMER

É de suma importância observar que algumas condições de funcionamento devem ser analisadas. Antes disso, porém, são feitas algumas considerações sobre a nomenclatura a ser utilizada com o intuito de facilitar tal análise e a própria

representação gráfica.

Sejam dois nós quaisquer N_i e N_j interligados por N_a arestas. Define-se N_{ai} o número de arestas necessárias para tornar o nó N_i ativo, e N_{aj} o número de arestas necessárias para tornar o nó N_j ativo. N_{ai} e N_{aj} também significam respectivamente o número de arestas que os nós N_i e N_j reverterão após suas operações internas. Define-se A_i como o número de arestas que em um dado instante do processo estão revertidas para o nó N_i, e A_j como o número de arestas que em um dado instante do processo estão revertidas para o nó N_j. Obviamente, o número de arestas N_a é igual à soma de A_i e A_j.

Sendo assim, considera-se N_{ai} um número inteiro associado ao nó N_i, da mesma forma que N_{aj} é considerado um número inteiro associado ao nó N_j. A representação gráfica, utilizando a nomenclatura definida acima, está ilustrada na Figura 26.

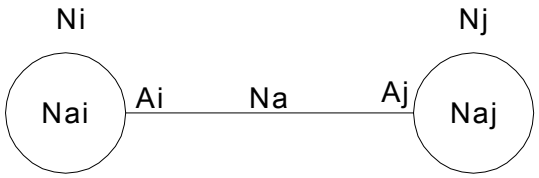


Figura 26 – Representação gráfica de uma malha SMER

Análise de funcionamento para dois nós:

Uma vez que o SMER é a generalização do SER, é necessário que a abordagem mais recente obedeça aos padrões de funcionamento do conceito original.

Uma das condições necessárias para o funcionamento do SER é que dois nós quaisquer interligados não podem estar ativos ou desativados ao mesmo tempo. Esta condição sugere que deve existir alguma interdependência entre N_{ai}, N_{aj} e N_a, de tal forma que este requisito seja atendido. Sendo assim:

Teorema 1:

Dados dois nós quaisquer N_i e N_j, interligados por N_a arestas e sendo N_{ai} o número associado ao nó N_i, e N_{aj}, o número associado ao nó N_j, tal configuração atenderá ao requisito anterior se e somente se

$$N_a = N_{ai} + N_{aj} - 1 \qquad \text{Equação 14}$$

Prova do teorema:

Considerando inicialmente que

$$N_{ai} + N_{aj} = N_a \qquad \text{Equação 15}$$

Neste caso, em algum instante do processo poder-se ia ter:

$$A_i = N_{ai} \text{ e } A_j = N_{aj} \qquad \text{Equação 16}$$

ou seja, o nó N_i e o nó N_j estariam ativos ao mesmo tempo. Com tal configuração, contrariando a condição de funcionamento proposta, chegar-se-ia à seguinte conclusão:

$$N_{ai} + N_{aj} > N_a \qquad \text{Equação 17}$$

Considerando outra possibilidade:

$$N_a = (N_{ai} - 1) + (N_{aj} - 1) \qquad \text{Equação 18}$$

Neste caso, em algum instante do processo haveria a seguinte possibilidade:

$$A_i = N_{ai} - 1 \text{ e } A_j = N_{aj} - 1 \qquad \text{Equação 19}$$

ou seja, como os nós N_i e N_j precisariam, respectivamente de no mínimo N_{ai} e N_{aj} arestas revertidas para os mesmos, de forma que se tornassem ativos, nota-se que o processo seria interrompido, uma vez que ambos os nós tornar-se-iam desativados. Tal configuração também contraria a condição proposta, podendo ser evitada obedecendo-se à seguinte expressão:

$$N_a > (N_{ai} - 1) + (N_{aj} - 1) \qquad \text{Equação 20}$$

Analisando as expressões 17 e 20 , conclui-se que:

$$N_a = N_{ai} + N_{aj} - 1 \qquad \text{Equação 21}$$

Sendo assim, ao definir-se o número de arestas interligando dois nós quaisquer, define-se automaticamente o valor de $N_{ai} + N_{aj}$, e vice-versa.

O exemplo abaixo ilustra bem esta relação:

Considerando-se dois nós N_i e N_j, interligados por sete arestas, e fazendo uso do teorema 1, chega-se à seguinte conclusão:

$$N_a = N_{ai} + N_{aj} - 1 \qquad \text{Equação 22}$$

sendo que $N_a = 7$, ou seja,

$$N_{ai} + N_{aj} = 8 \qquad \text{Equação 23}$$

Dessa forma, o número de combinações possíveis para os valores de N_{ai} e N_{aj} é finito e pode ser determinado.

Teorema 2:

O número de combinações possíveis para N_{ai} e N_{aj} é finito e igual a $N_{ai} + N_{aj} - 1$.

Prova do teorema:

Considerando um sistema SMER de dois nós N_i e N_j, interligados por N_a arestas, tem-se que a soma de N_{ai} e N_{aj} é constante e igual a $N_a + 1$. Sendo assim, N_{ai} pode variar progressivamente de 1 até N_a, com N_{aj} variando regressivamente, ao mesmo tempo, de N_a até 1, o que totaliza N_a combinações possíveis.

Operação Cíclica:

Considerando-se dois nós, o SMER foi definido de forma a operar de maneira cíclica, obedecendo a uma propriedade apresentada na teoria de algoritmos distribuídos, em [27] que define o número de ciclos (N_c) que o SMER deve executar para retornar à configuração inicial.

Definição: $N_c = (N_{ai} + N_{aj}) / \text{MDC}(N_{ai}, N_{aj})$, onde $\text{MDC}(N_{ai}, N_{aj})$ representa o máximo divisor comum entre N_{ai} e N_{aj}.

Assim sendo, um sistema SMER com dois nós N_i e N_j, onde o número de arestas N_a é igual a 5, tem como resultado da soma $N_{ai} + N_{aj}$ o valor 6, e o número de combinações possíveis para N_{ai} e N_{aj} igual a 5. A Tabela 7 apresenta as possíveis combinações e os respectivos números de ciclos executados.

A ilustração deste exemplo pode ser observada na Figura 27, onde são apresentadas as combinações 5-1, 4-2, 3-3 e seus respectivos ciclos.

N_{ai}	N_{aj}	Número de Ciclos
5	1	(5 + 1) / 1 = 5
4	2	(4 + 2) / 2 = 2
3	3	(3 + 3) / 3 = 2
2	4	(2 + 4) / 2 = 2
1	5	(1 + 5) / 1 = 5

Tabela 7 – SMER para N_a = 5

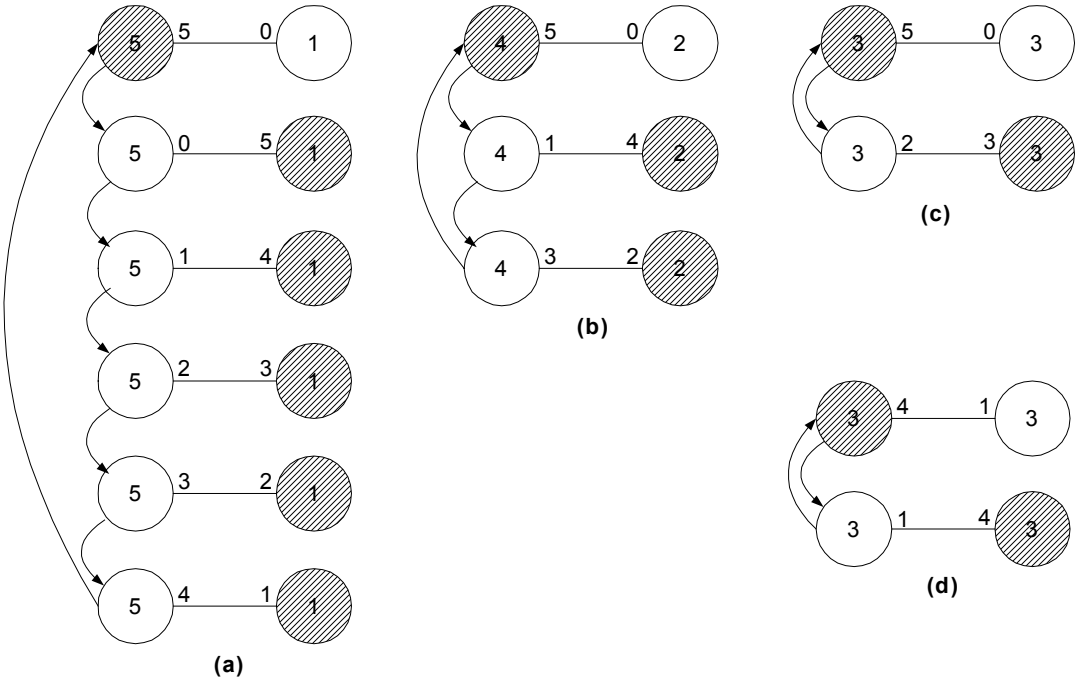


Figura 27 – SMER para Na = 5

Outra observação que merece destaque está relacionada ao número de ciclos, ou seja, quando a soma $N_{ai} + N_{aj}$ for um número primo ou N_{ai} e N_{aj} forem primos entre si, o número de ciclos para o SMER será o mesmo para todas as configurações adotadas e igual a:

$$N_c = N_{ai} + N_{aj} \qquad \text{Equação 24}$$

Tal propriedade possibilita a obtenção de configurações em que apenas as seqüências de reversões se alteram.

A Figura 28 ilustra esta característica apresentando as seqüências de reversões para $N_{ai} + N_{aj} = 5$, ou seja, $N_a = 4$ e $N_c = 5$.

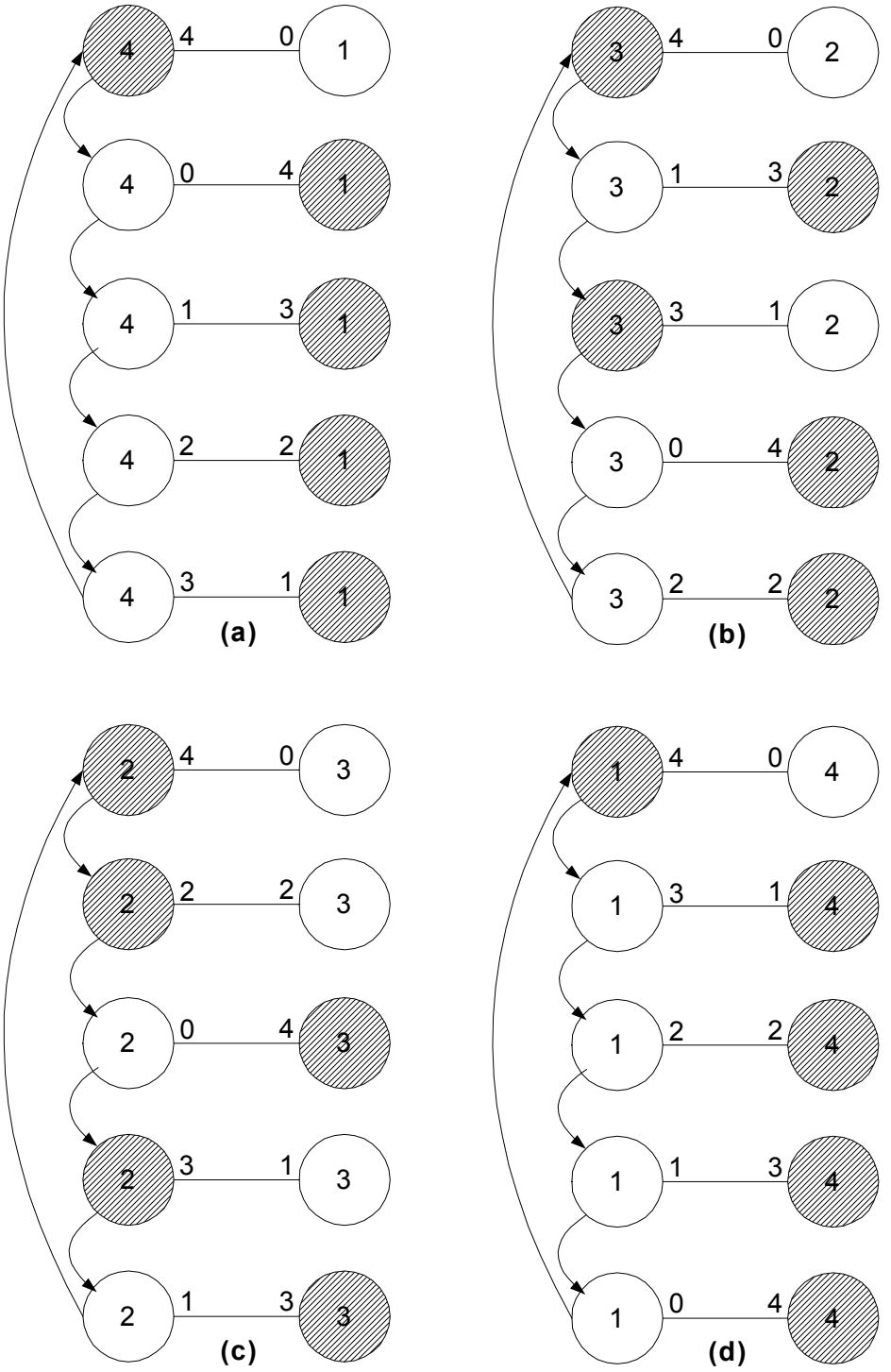


Figura 28 – SMER para $N_{ai} + N_{aj} = 5$

A teoria de escalonamento por reversões de arestas múltiplas pode ser utilizada em controle de processos síncronos e assíncronos, tendo ainda um futuro promissor na área de desenvolvimento de algoritmos criptográficos.

4.2.1.2.2. SMER em criptografia

Dentre as possíveis aplicações para o SMER está o seu promissor uso em criptografia. Para isso, o SMER pode ser utilizado como uma função randômica associada a operações de cifragem de *bits* ou para operações de geração de subchaves.

Esse tipo de função teria como principal aplicação o aumento da difusão da informação [17] em algoritmos criptográficos. Para tal, a função receberia *x bits* que, depois de interagirem com uma chave de *k bits*, gerariam uma saída de *y bits*, conforme ilustrado na Figura 29.

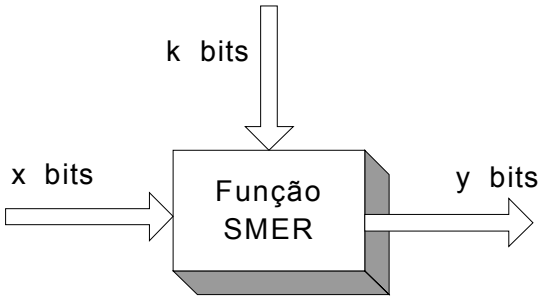


Figura 29 – Função genérica SMER

Dessa forma, torna-se necessário um estudo mais aprofundado das possibilidades de escolha para uma função SMER e o seu uso em criptografia, ou seja, da definição das configurações convenientes para tal função, assim como de sua aplicação em estruturas do tipo redes de Feistel ou em conjunto com funções lineares e não lineares, como no caso do algoritmo IDEA.

4.2.1.2.3. A função SMER

O estudo para a definição de uma função estruturada na técnica SMER, para o uso em criptografia, baseou-se nas seguintes condições:

i) a função deveria ser sobrejetora, ou seja, todos os resultados possíveis (contra-domínio) comporiam a imagem da mesma. Dessa maneira, para uma função com saída de oito *bits*, o número de saídas distintas e possíveis deveria ser $2^8 = 256$. Esta condição é desejável para que se possa operar com a máxima difusão, aumentando assim a segurança da função;

ii) a função deveria ser injetora, ou seja, cada elemento da imagem da função corresponderia a um e somente um elemento do domínio da mesma. Tal característica implica na inversibilidade da função, possibilitando sua aplicação em estruturas como redes de Feistel (inversibilidade não é necessária) ou compostas de interações entre funções lineares ou não (inversibilidade necessária), como é o caso dos algoritmos IDEA e *Modular Multiplication-based Block cipher* (MMB);

iii) a função estudada deveria operar com padrões de blocos de dados utilizados em algoritmos criptográficos e principalmente em plataformas de *hardware* e de processamento computacional, ou seja, potências de 2 tais como: 32 *bits*, 64 *bits*, 128 *bits* etc.;

iv) a função deveria apresentar um desempenho satisfatório, ou seja, depois de atender aos requisitos anteriores, deveria atingir velocidades de processamento em *software* e *hardware* compatíveis com as requeridas atualmente.

Ao atender-se às condições i e ii, tem-se que a função em questão deverá ser bijetora.

4.2.1.2.4. A função SMER adotada

Após um estudo baseado nas condições anteriores e na necessidade de altas velocidades de processamento, optou-se por adotar uma função SMER com as seguintes características:

- configuração de operação para dois nós. Esta decisão deve-se ao fato da necessidade da inversibilidade da função e da necessidade de um rápido processamento, uma vez que, para dois nós quaisquer, o modelo e as condições de funcionamento são inteiramente conhecidas;
- a função tem um número de arestas N_a igual a 15;
- a função indexa quatro *bits* cujo valor será representado pelo número de arestas revertidas para um dos nós em cada ciclo (será considerado o segundo nó).

Como $N_a = 15$, tem-se que $N_{ai} + N_{aj} = 16$. Uma vez que a função indexa quatro *bits* cobrindo todas as possibilidades, seria necessário que o número de ciclos fosse igual a 16, isto é:

$$N_c = (N_{ai} + N_{aj}) / \mathrm{MDC}(N_{ai}, N_{aj}) = 16 \qquad \text{Equação 25}$$

Para que essa expressão seja verdadeira é necessário que N_{ai} e N_{aj} sejam primos entre si. Dessa forma, existem apenas oito possíveis combinações para N_{ai} e N_{aj}. Essas combinações são apresentadas na Tabela 8.

Finalmente, define-se a função SMER como um processo que recebe quatro *bits* de entrada e gera uma saída, igualmente de quatro *bits*, após a interação da entrada com uma chave de sete *bits*, conforme ilustrado na Figura 30.

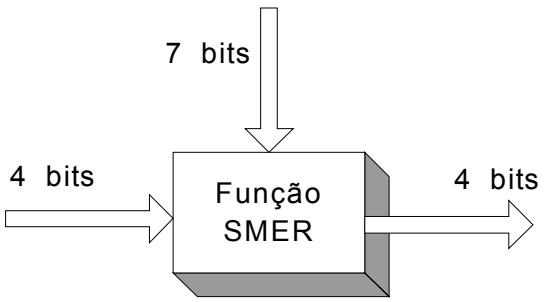


Figura 30 – Função SMER

Os três *bits* mais significativos da chave de sete *bits* determinam a configuração adotada, entre as oito possíveis para a função SMER, conforme apresentado na Tabela 8.

N_{ai}	N_{aj}	*Bits* de Configuração
1	15	000
3	13	001
5	11	010
7	9	011
9	7	100
11	5	101
13	3	110
15	1	111

Tabela 8 – *Bits* de configuração para possíveis combinações na função SMER

Os quatro *bits* menos significativos determinam o número de ciclos a serem executados para a geração do resultado. A Figura 31 ilustra a composição da chave de sete *bits*.

Para exemplificar o funcionamento da função SMER, supõe-se que o dado a ser cifrado seja 0110 e que a chave de cifragem seja 0010100. Dessa forma, a configuração a ser adotada é definida pelos três *bits* mais significativos da chave (001), ou seja, configuração 3 – 13. Os quatro *bits* menos significativos (0100) determinam o número de ciclos a serem executados, no caso, quatro ciclos.

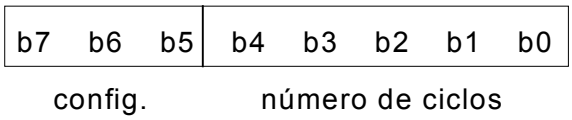


Figura 31 – Composição da chave de sete *bits*

Uma vez que o número inicial de arestas é determinado pelos quatro *bits* de entrada, a saída gerada após a execução de quatro ciclos na configuração 3 – 13 é 0010. Esse procedimento pode ser bem entendido com o auxílio da Figura 32.

Para a recuperação do dado inicial, operação de decifragem, a chave passa a ser o complemento da chave utilizada para cifragem, ou seja, os três *bits* mais significativos (configuração) são os mesmos e os quatro menos significativos passam a ser o complemento a 16 do número de ciclos usado na operação de cifragem.

No exemplo anterior, a chave de cifragem seria 0011100 e o novo dado de entrada, cifrado previamente, seria 0010, apresentando como saída o dado original.

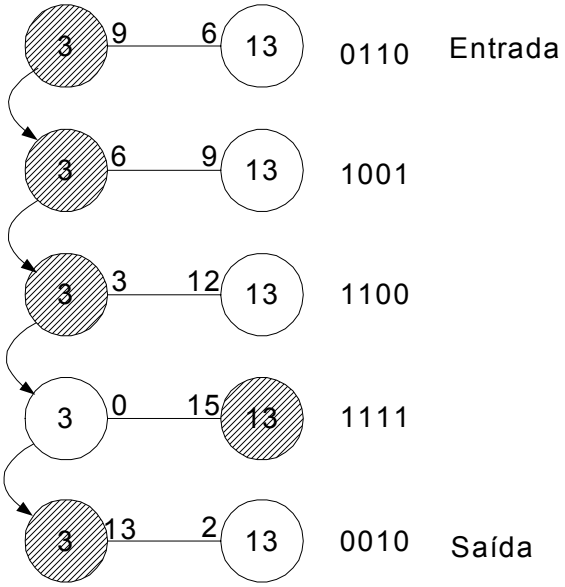


Figura 32 – Exemplo de funcionamento da função SMER adotada

4.2.1.2.5. Otimização da função SMER

No trabalho apresentado em [60] por Salomão, a implementação da função SMER foi executada com base no conceito de *look-up-table,* ou seja, os valores da entrada da função e da chave associada compõem um endereço que, aplicado a um segmento de memória, resultam em uma saída criptografada. A adoção desse procedimento proporcionou um alto desempenho ao circuito sintetizado para essa função. No entanto, foram observados elevados valores para área ocupada. A partir de então, tornou-se imperativo um estudo mais aprofundado do funcionamento do

algoritmo SMER, de maneira que se adotasse uma nova arquitetura para a função em tela, de forma a diminuir os valores da área ocupada sem comprometer a velocidade de operação da mesma.

Foi possível a identificação de uma lei de formação para a saída dos dados, em função da entrada e da chave associada.

Observações efetuadas:

i) as configurações são identificadas pelos três *bits* mais significativos da chave e, portanto, variam de 0 a 7;

ii) para cada configuração, a diferença entre os valores de dois estágios sucessivos é constante e varia de um número ímpar entre 1 e 15;

iii) a partir das duas observações anteriores, pode-se afirmar que cada configuração C tem um valor constante de diferença entre estágios, correspondente a um passo P cujo valor é $(2C + 1)$;

iv) o número de ciclos N determinado pelos quatro *bits* menos significativos da chave implica no número de vezes que o passo, determinado anteriormente, deve ser somado à entrada E da função;

v) a saída S da função tem tamanho fixo de quatro *bits* e, por isso, todo o resultado deverá ser truncado nesse valor, de forma a obedecer tal requisito.

De posse das observações anteriores pode-se afirmar que a expressão que define o funcionamento da função SMER é a seguinte:

$$S = (E + NP) \bmod (16), \text{ onde } P = (2C + 1) \qquad \text{Equação 26}$$

Para fins de implementação em *hardware*, pode-se, a partir de agora, visualizar as funções necessárias para síntese do circuito desejado: somador de quatro *bits*, multiplicador de quatro *bits* e a função módulo(16).

4.2.1.2.6. A função módulo(16)

Em termos de implementação em *hardware*, a função módulo(16) é de uma simplicidade extrema, uma vez que bastam ser desconsiderados os *bits* superiores a quatro em todas as operações. Sendo assim, esta função ficará implícita nas outras.

4.2.2. Implementação

O algoritmo Godzuk foi implementado com base na função SMER, descrita acima, e em operações lógicas ou-exclusivo.

A seqüência de funcionamento do Godzuk, executada em oito fases e descrita passo a passo no pseudocódigo da Figura 33, é retratada no diagrama em blocos da Figura 34.

```
Divide-se o bloco de entrada de 128 bits em dois sub-blocos de 64 bits, XE e XD;
Para i = 1 até 8 tem-se:
        XE = XE XOR P_i;
        XD = FI(XE) XOR XD;
        Se i < 8 então
                Troque XE e XD;
        Senão
                XE = XE XOR P_10;
                XD = XD XOR P_9;
Combinando-se XE e XD, obtém-se o dado cifrado de 64 bits.
```

Figura 33 – Descrição da seqüência de funcionamento do Godzuk

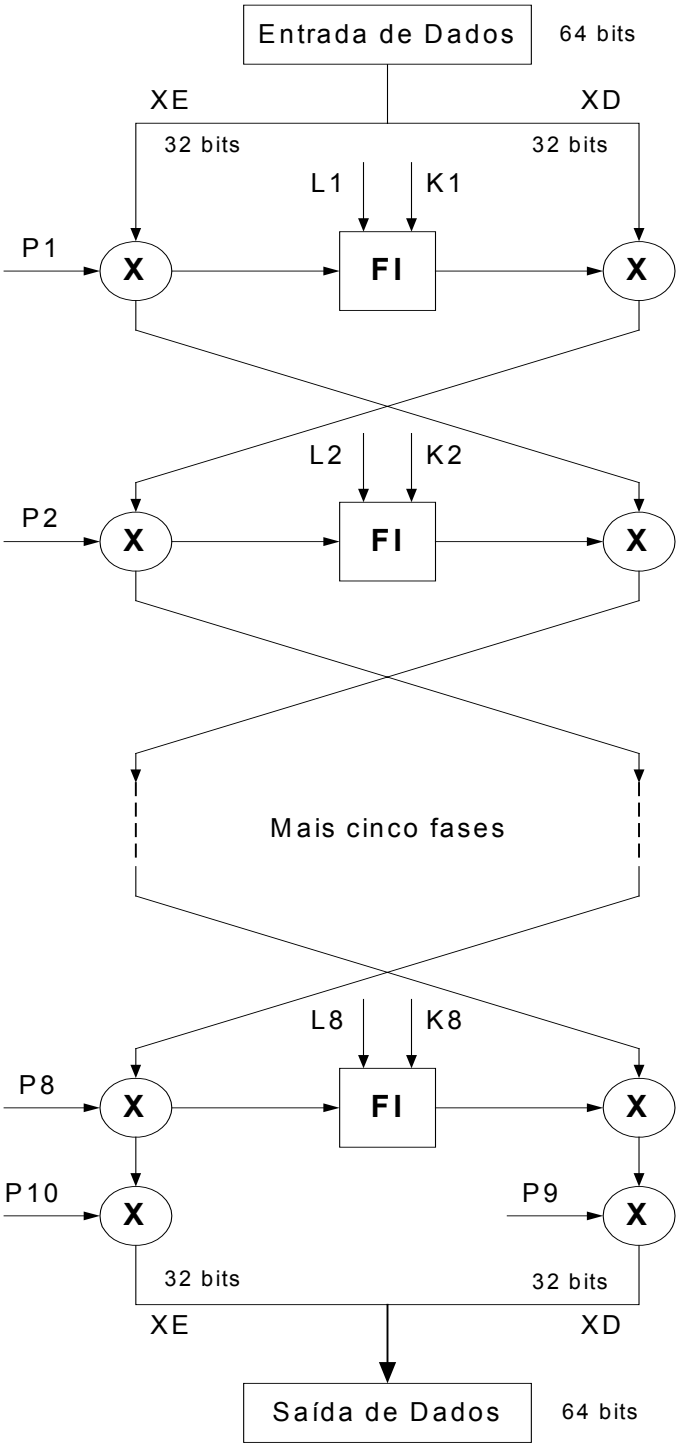


Figura 34 – Cifragem do Godzuk

As subchaves utilizadas são as seguintes: dez subchaves P_i de 64 *bits*, oito subchaves L_i de 48 *bits* e oito subchaves K_i de 84 *bits*. O método de geração dessas subchaves varia com as arquiteturas adotadas e por isso serão apresentadas durante as descrições das implementações.

A função FI:

A função FI utilizada no Godzuk está baseada na rede de Feistel, tendo três fases. A saída desta função, que anteriormente era o resultado de uma operação de ou-exclusivo entre os ramos da rede e as subchaves do sistema, passa a ser simplesmente a saída do último estágio. O pseudocódigo que descreve esta função é apresentado na Figura 35 e representada na Figura 36.

Divide-se o bloco de entrada de 32 *bits* em dois sub-blocos de 16 *bits*, XE e XD;
Para k = 1 até 3 tem-se:
    XE = XE XOR $L_{i,j}$;
    XD = FS(XE) XOR XD;
    Troque XE e XD;
Combinando-se XE e XD, obtém-se o dado cifrado de 32 *bits*.

Figura 35 – Descrição da função FI

As subchaves utilizadas na execução da função FI são as seguintes: três subchaves $L_{i,j}$ de 16 *bits*, obtidas a partir da divisão de cada subchave L_i (48 *bits*) em três novas subchaves $L_{i,j}$ (16 *bits*); três subchaves $K_{i,j}$ de 28 *bits*, obtidas a partir da divisão da subchave K_i (84 *bits*) em três novas subchaves $K_{i,j}$ (28 *bits*).

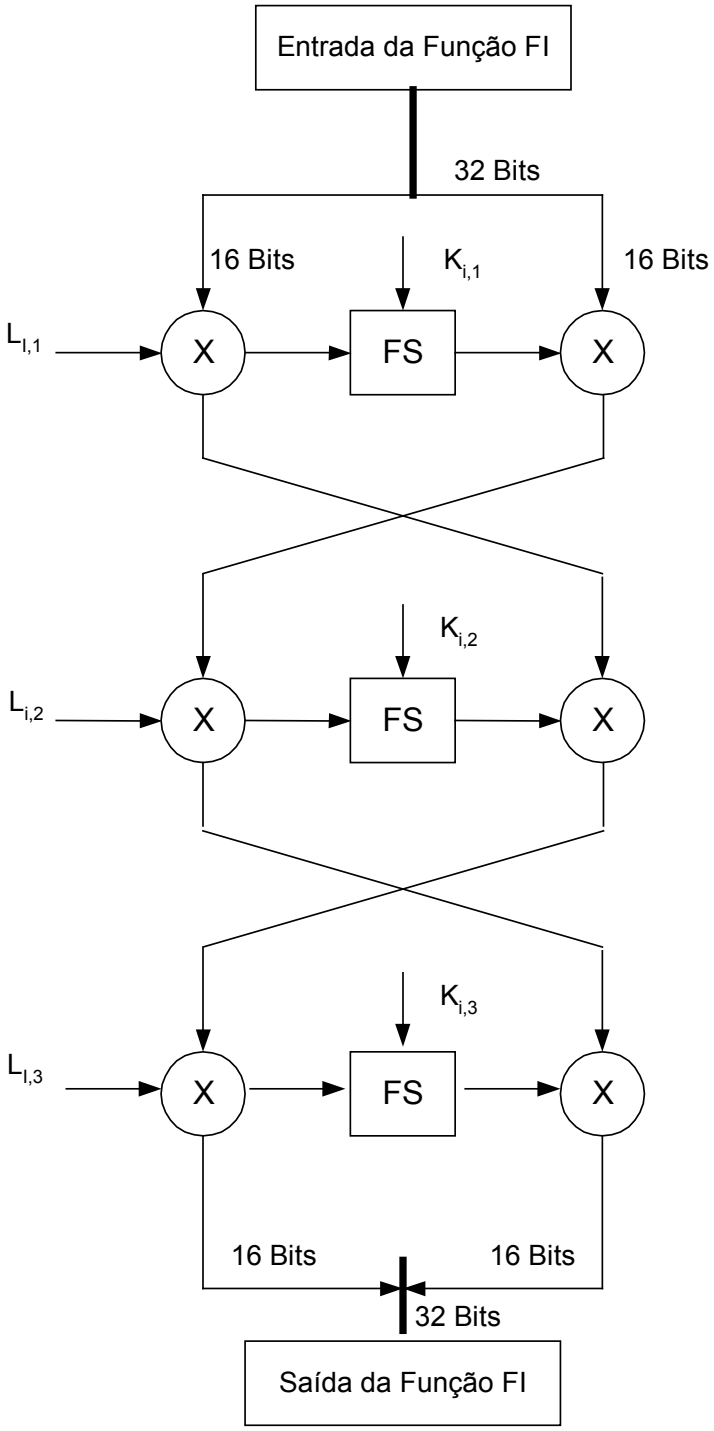


Figura 36 – Função FI do Godzuk

A função FS:

A função FS, apresentada na Figura 37, possui quatro funções SMER trabalhando em paralelo, de maneira que FS passe a operar com blocos de 16 *bits*, adequando-se à função FI.

As subchaves $K_{i,j,1}$, $K_{i,j,2}$, $K_{i,j,3}$ e $K_{i,j,4}$ possuem sete *bits* cada e são derivadas a partir da divisão da subchave $K_{i,j}$ (28 *bits*) em quatro novas subchaves.

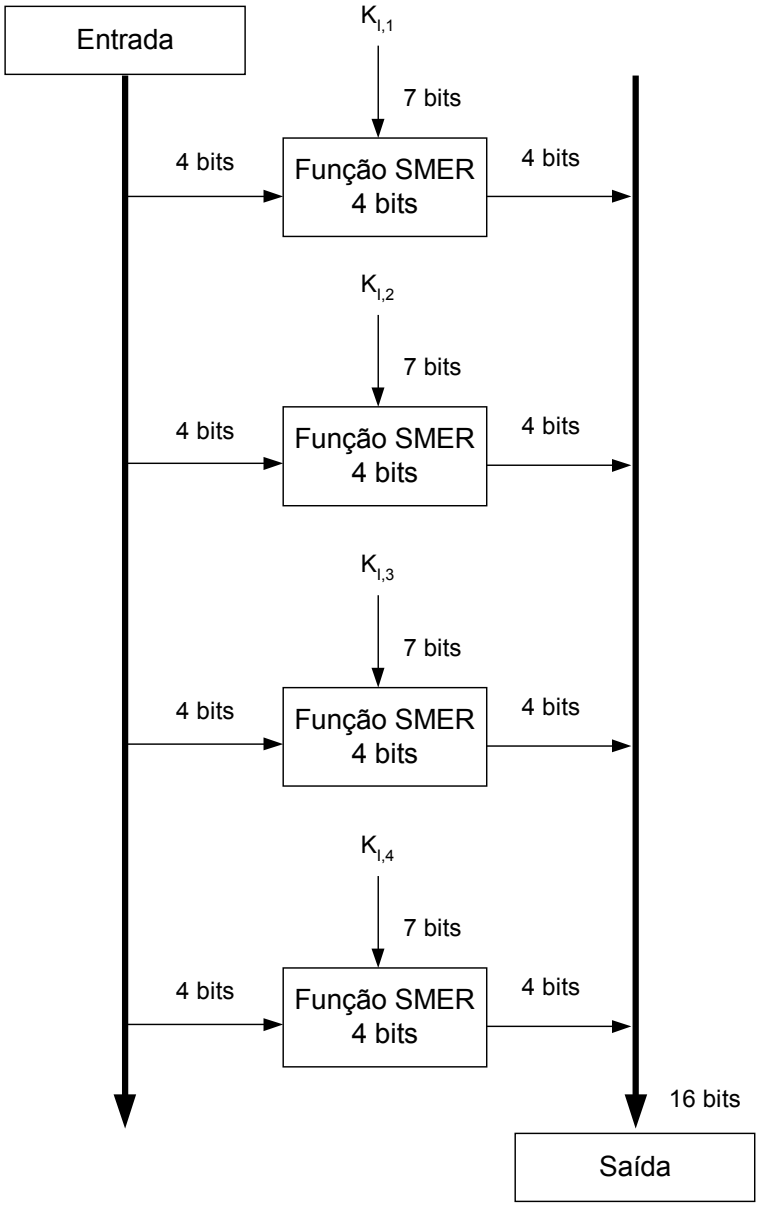


Figura 37 – Função FS do Godzuk

Segurança do algoritmo Godzuk:

A segurança do algoritmo criptográfico Godzuk está presente na sua concepção, que está baseada na teoria da provável segurança. Matsui prova em [67] que, para blocos de dados de tamanho *n*, sendo processados por uma função par, tanto a probabilidade diferencial média, quanto a probabilidade linear média, apresentarão um valor mínimo, ou seja, no pior caso, de 2^{-n+2}.

Operação de decifragem:

A operação de decifragem é efetuada da mesma forma que a cifragem, com uma pequena diferença: as chaves precisam ser invertidas na entrada do corpo principal.

Implementações do Godzuk:

Da mesma maneira que foi feito para os outros algoritmos criptográficos, a implementação do algoritmo Godzuk foi executada com ênfase em uma descrição comportamental, de forma que o principal objetivo fosse validar o seu funcionamento.

A entidade que descreveu a mais alta hierarquia do algoritmo apresentou seis sinais: cinco entradas e uma saída. As entradas podem ser divididas em sinais de temporização (*clk* e *reset*), sinais de controle (*crip_dec* e *data_enable*), e sinais de entrada e saída de dados (*GOD_in* e *GOD_out*). Os sinais de temporização e controle têm largura de um *bit*. Os sinais de entrada e saída de dados têm largura de 64 *bits*. A Figura 38 ilustra com precisão os sinais que acabam de ser definidos.

Figura 38 – Sinais da implementação do algoritmo Godzuk

Com relação à temporização, a função dos sinais apresentados é bastante intuitiva: o *clk* representa o relógio do sistema, sendo a borda de subida definida como referência temporal. O *reset* sinaliza o início ou reinício do processo, toda vez que o processo notar a mudança do estado lógico ‘1’para o estado ‘0’. Os sinais de controle têm papel importante no correto funcionamento do algoritmo Godzuk: o *crip_dec* sinaliza se a operação é de cifragem ou decifragem, conforme seu estado lógico seja ‘1’ ou ‘0’, respectivamente. O sinal *data_enable* tem a função de sinalizar ao algoritmo se o dado de entrada é um dado a ser cifrado ou se é a chave de cifragem, conforme seu estado lógico seja ‘1’ ou ‘0’, respectivamente.

De posse dessas informações pode-se, a partir de agora, analisar com maior profundidade algumas funcionalidades relevantes do Godzuk.

Captura de dados:

A entidade que controla este processo tem as mesmas entradas do corpo principal do algoritmo Godzuk, exceto pelo sinal *crip_dec*. São definidas duas saídas, uma com largura de 64 *bits*, representando o dado criptografado, e a outra representando a chave de operação, com 128 *bits*. O funcionamento deste circuito é extremamente simples: a cada borda de subida do *clk*, dependendo do estado do sinal *data_enable*, é capturado, por um dos registradores de saída, ou dado ou a chave, sendo que no caso desta última, serão necessárias duas bordas de subida sucessivas para que a chave do sistema seja capturada.

Geração das subchaves:

A geração de subchaves para esta implementação tem ligação direta com a captura de dados, ou seja, é a partir da porta de saída *key_in*, da entidade captura, que as subchaves são geradas. O método adotado para a geração é o seguinte: uma vez capturada a chave do sistema, esta é expandida ou compactada, de forma a adequar-se ao tamanho da chave em questão, passando a ser denominada *fonte*. A seguir, esse sinal é segmentado em quatro trechos de tamanhos iguais que, depois de combinados em ordem inversa, darão origem às subchaves do algoritmo. É importante salientar que na composição das novas subchaves, alguns desses trechos podem ter o estado lógico de seus *bits* invertidos, em função, ou não, do estado lógico do sinal *crip_dec*,

possibilitando assim que as subchaves sejam invertidas corretamente, de forma que se possa realizar a decifragem. Vale notar também que, ao se gerar metade das subchaves, basta gerar o complemento dessas para que se obtenha a outra metade.

a) As subchaves *P*

Para a geração das dez subchaves *P*, dividiu-se a chave do sistema em quatro segmentos *seg1*, *seg2*, *seg3* e *seg4*, do menos significativo para o mais significativo, e efetuaram-se duas operações de ou-exclusivo, uma entre os dois primeiros segmentos e outra para os dois últimos. Finalmente executou-se um *and* lógico entre o resultado das operações descritas anteriormente, formando-se o sinal *fonte*. A partir de então, o processo de formação das novas subchaves é todo baseado neste sinal. A Figura 39 ilustra com precisão tal processo, sendo de suma importância observar que os blocos de cor azul têm o estado lógico de seus *bits* invertido quando o sinal *crip_dec* assume valor '0'; o mesmo acontecendo com os blocos de cor branca quando o sinal *crip_dec* assume valor '1'; os blocos de cor amarela e verde independem do estado de *crip_dec*, sendo que os verdes representam a inversão do estado lógico de seus *bits*. Com isso, ao alterar-se o estado lógico de *crip_dec*, as dez subchaves se inverterão automaticamente no corpo do algoritmo.

b) As subchaves *L*

Para a geração das oito subchaves *L*, efetuou-se uma operação de *and* lógico entre o resultado de uma operação de ou-exclusivo e uma concatenação. A operação de ou-exclusivo foi feita entre os 48 *bits* mais e menos significativos da chave do sistema, e a concatenação efetuou-se com os seguintes segmentos: *not* (*key* (127 : 112)) e *key*(79 : 48). Dessa maneira, compôs-se um sinal *fonte* com 48 *bits*, adequado para a geração das subchaves *L*. A Figura 40 apresenta o processo de formação das subchaves.

c) As subchaves *K*

De maneira semelhante às anteriores, a geração das oito subchaves *K* inicia-se com a formação do sinal *fonte*, que é o resultado de uma operação de ou-exclusivo entre os seguintes segmentos da chave do sistema: *key*(127 : 44) e *key*(83 : 0). A Figura 41 apresenta com clareza o processo de formação das subchaves *K*.

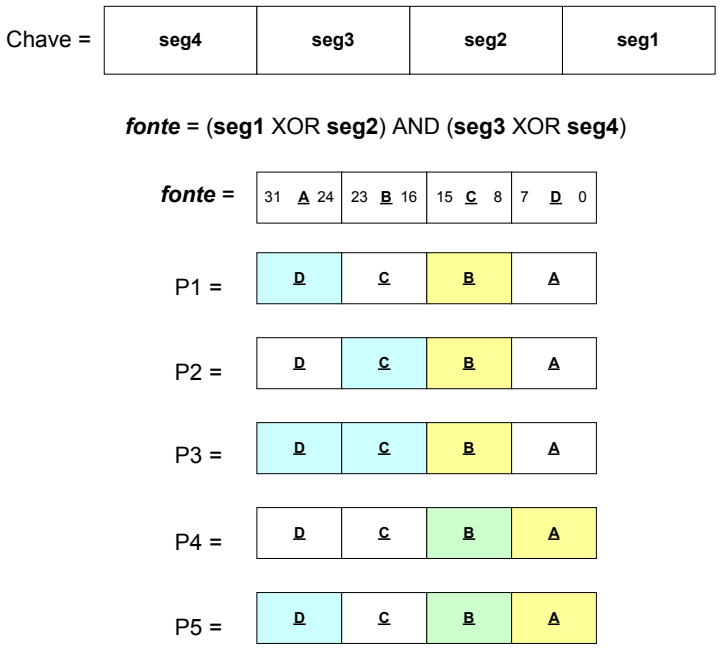


Figura 39 – Geração das subchaves *P* (Godzuk)

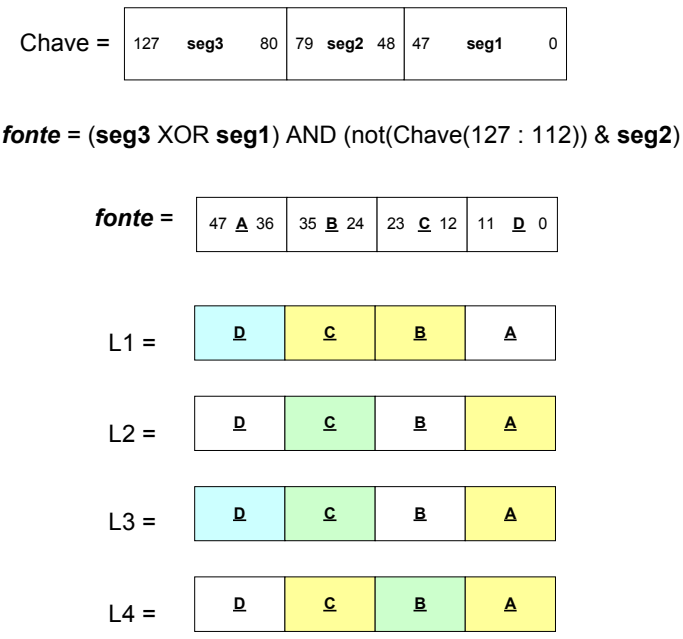


Figura 40 – Geração das subchaves *L* (Godzuk)

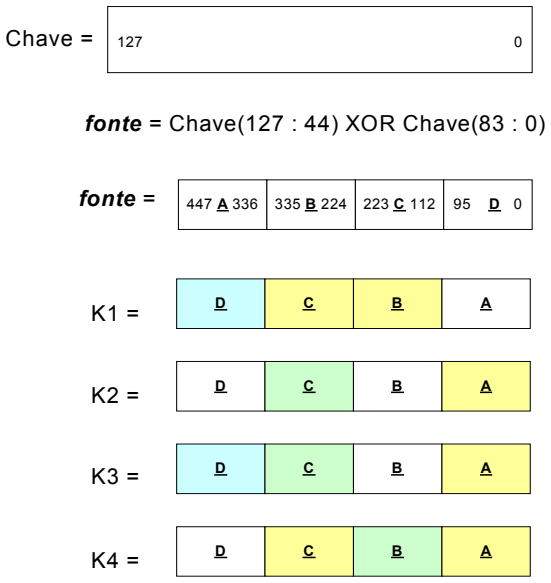


Figura 41 – Geração das subchaves *K* (Godzuk)

No desenvolvimento da arquitetura do algoritmo Godzuk optou-se por implementar uma arquitetura otimizada em relação aos conceitos de recorrência, ou seja, a presença de uma estrutura regular que se repete em uma seqüência de oito estágios compostos por duas funções lógicas de ou-exclusivo e uma função FI.

Desta forma, a arquitetura proposta contempla a idéia de recorrência e, em conseqüência disso, foi desenvolvida uma entidade que controlará a temporização do algoritmo, comandando a geração dinâmica de subchaves, além da realimentação, entrada e saída de dados. A Figura 42 apresenta com detalhes o funcionamento do algoritmo em questão.

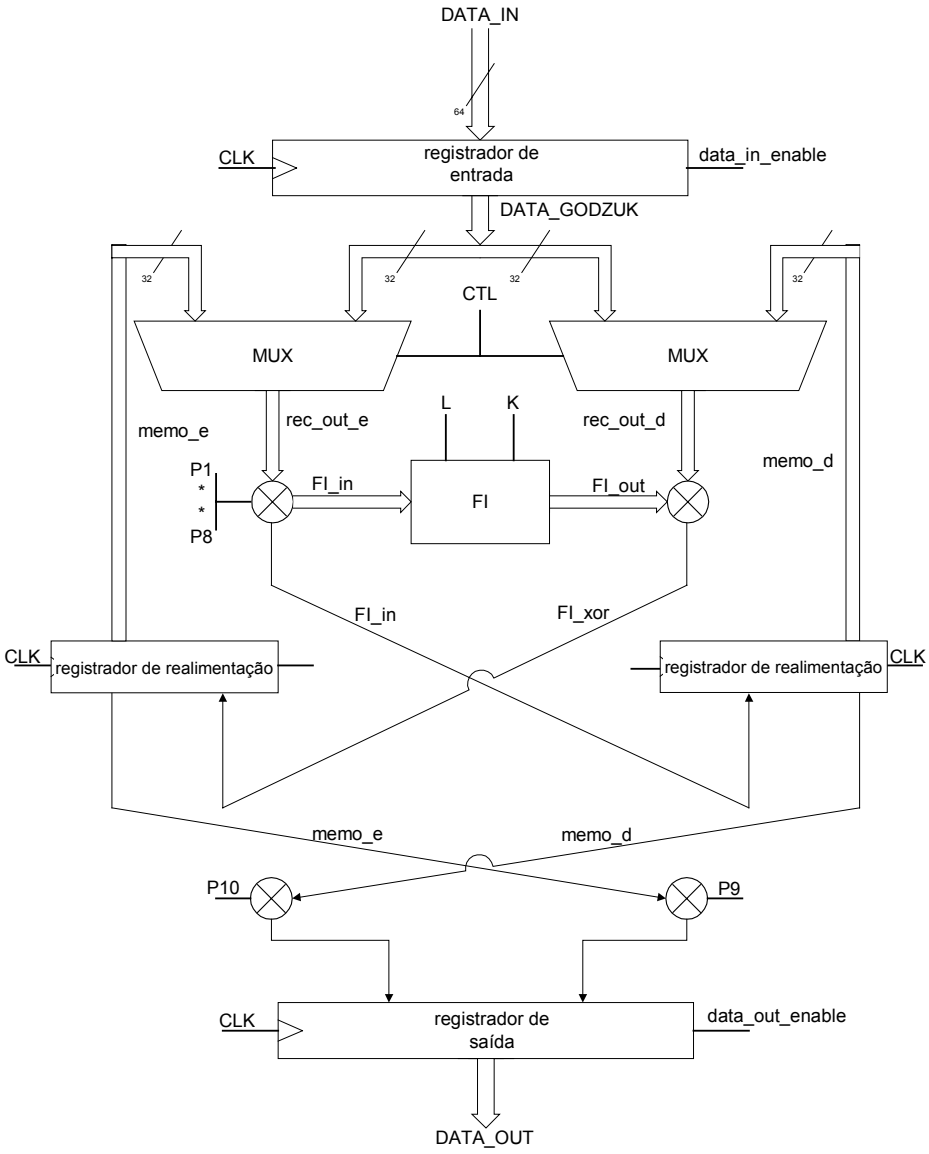


Figura 42 – Diagrama em blocos do algoritmo Godzuk

4.2.3. Resultados

Depois da descrição do Godzuk em VHDL, o algoritmo foi validado com a simulação comportamental efetuada com a ferramenta *ModelSim* da *Xilinx* e o código foi sintetizado para ASIC, utilizando-se a ferramenta Synopsys configurada com a biblioteca de células básicas 0,7 μm CMOS ES2. Os resultados podem ser observados na Tabela 9 [60].

Resultados	ASIC
Freqüência de operação	78 MHz
Taxa de cifragem	255 *Mbits*/s
Tempo total para uma cifragem para mensagem de 1024 *bytes*	39 μs

Tabela 9 – Resultados da implementação de Godzuk

4.3. Considerações

Com base nas implementações dos algoritmos criptográficos simétricos Blowfish e Godzuk, obteve-se o desempenho e a área aproximada, em número de portas, de cada circuito. Da mesma forma que com os algoritmos assimétricos, apresentados no capítulo anterior, a comparação de desempenho entre esses algoritmos criptográficos foi estabelecida por meio da avaliação do desempenho de cada algoritmo na execução de um mesmo bloco de mensagem de 1024 *bytes* de tamanho. Dessa forma, os desempenhos puderam ser comparados, pelos tempos de processamento dessa mensagem comum a todos, através dos seus atrasos.

Pôde-se constatar, também, que o desempenho obtido para cada um dos algoritmos simétricos foi equivalente às implementações de algoritmos simétricos semelhantes anteriormente publicados [70]. Para o algoritmo criptográfico simétrico Blowfish não há muitas implementações em *hardware*, apenas [15] demonstra que o desempenho obtido neste trabalho, de 200 *Mbits*/s, mais uma vez é adequado e compatível com as demais implementações existentes.

Para o algoritmo Godzuk não há outras implementações para comparação, sendo 255 *Mbits*/s uma taxa de desempenho compatível com outras implementações de algoritmos criptográficos simétricos [70].

No próximo Capítulo será apresentada uma técnica de avaliação do consumo de potência de circuitos descritos em VHDL. Com essa técnica será possível comparar as implementações dos algoritmos criptográficos, não apenas em termos de desempenho, mas principalmente quanto à potência dissipada, característica fundamental nas redes sem fio.

Capítulo 5

Estimativa de consumo de potência dos algoritmos

A redução dos terminais das redes sem fios, onde o tamanho e o peso das baterias são relevantes e a exigência desses dispositivos de cada vez mais executar tarefas que exijam maior desempenho computacional, torna o baixo consumo de energia uma das mais importantes e desejáveis características dos sistemas de comunicação sem fios [1].

Existem várias técnicas empregadas nos protocolos para reduzir o consumo de energia nas redes sem fio sem que ocorram perdas significativas na sua capacidade e na sua conectividade [71]. A inclusão de um *hardware* com a função de prover segurança a essas redes provoca uma preocupação extra com o impacto dessa inserção, não apenas no desempenho, mas também no novo consumo de potência dos dispositivos. O novo circuito agregado não deve provocar uma perda significativa na eficiência global do sistema, proporcionando um impacto mínimo no consumo de potência.

Os principais fatores que orientam o desenvolvimento de circuitos aplicados a dispositivos portáteis (PDA's, *notebooks* e telefones celulares) não são apenas área e velocidade. Atualmente o consumo de potência desponta como uma grande exigência nos subsistemas que integram esses dispositivos.

A avaliação do consumo de um circuito na fase de desenvolvimento é realizada por meio de ferramentas de auxílio ao projeto de circuitos, as quais possuem recursos para estimar o consumo de potência do circuito, quando efetivamente implementado. Uma forma menos precisa e não mais utilizada baseia-se na densidade de integração (número de portas) e no desempenho (freqüência do *clock*) do circuito. Para a elaboração deste trabalho, não foi possível contar com os recursos de avaliação da métrica consumo de potência, da ferramenta de desenvolvimento Synopsys, tampouco da avaliação por meio do número de portas lógicas, que se tornou muito imprecisa. Sendo assim, este trabalho utilizou uma técnica para avaliar o consumo de potência de circuitos, de forma comparativa, a partir de suas descrições em VHDL, proposta em conjunto com [72].

5.1. Metodologia

A complexidade crescente dos circuitos integrados *Very Large Scale Integration* (VLSI) para aplicações móveis tornou o consumo de potência uma característica fundamental no projeto de circuitos, sobretudo com tecnologia CMOS (*Complementary Metal-Oxide Semiconductor*), uma vez que esta será a tecnologia dominante nos próximos anos [71]. Existem algumas equações que modelam os compromissos de desempenho e consumo dos circuitos CMOS. A Equação 27 define o consumo de potência total de um circuito digital [73].

$$P_{Total} = P_{Estática} + P_{Dinâmica} \qquad \text{Equação 27}$$

A potência estática não depende da atividade do circuito e está definida pela tecnologia empregada no projeto, não sendo considerada neste estudo. Contudo, a potência dinâmica pode ser definida como:

$$P_{Dinâmica} = P_{Chaveamento} + P_{Curto\text{-}circuito} \qquad \text{Equação 28}$$

Na Figura 43 são ilustradas todas as correntes envolvidas na estimativa do consumo de potência.

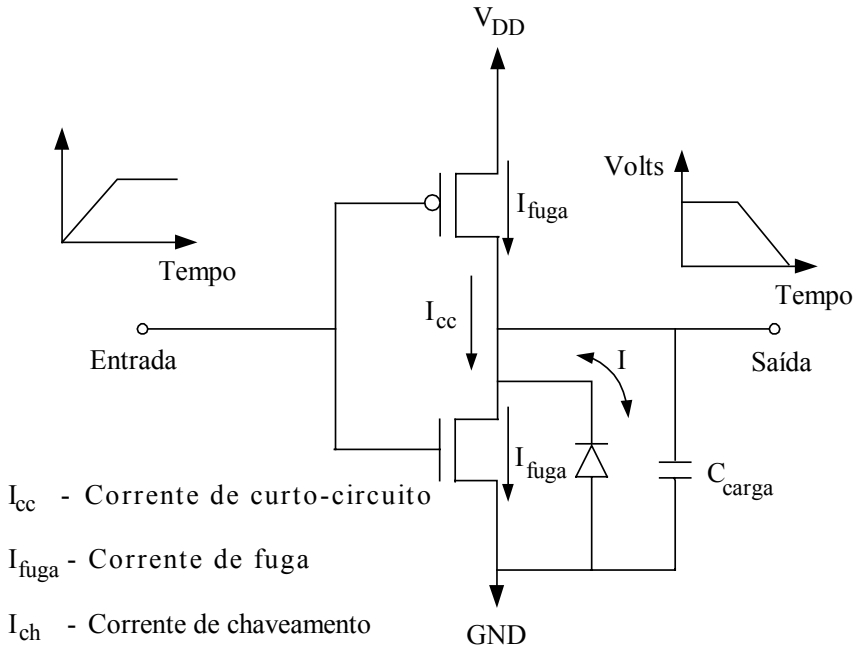


Figura 43 – Correntes na célula CMOS

$$P = ACV^2 f + \tau AVI_{cc} f + VI_{fuga} \qquad \text{Equação 29}$$

A Equação 29 é composta por três parcelas distintas.

A primeira parcela é responsável pela avaliação do consumo dinâmico do circuito provocado pela carga e descarga da capacitância formada na saída de cada porta. Esta parcela é proporcional à freqüência *f*, à atividade das portas no circuito *A* (representa a transição do sinal de saída em cada porta a cada ciclo do relógio), à capacitância vista pelas portas de saída *C*, e ao quadrado da tensão de alimentação *V*.

O segundo termo descreve a potência gasta devido às correntes de curto-circuito I_{CC} que, num intervalo de tempo τ , fluem entre tensão de alimentação e o terra, quando a saída de uma porta lógica CMOS muda de estado.

O terceiro termo mede a potência perdida em função da corrente de fuga que está presente, independente do estado da porta lógica.

Atualmente, o primeiro termo domina nos circuitos [71], sugerindo que o modo mais eficiente de diminuir o consumo de potência é reduzir a tensão de alimentação (V), uma vez que a redução da freqüência provoca um impacto negativo no desempenho

global do circuito. Infelizmente a redução da tensão também diminui a freqüência máxima do circuito, como mostra a Equação 30.

$$f_{máx} \propto (V - V_{limiar})^2 / V \qquad \text{Equação 30}$$

Sendo assim, como a maior parte do consumo de potência em circuitos CMOS está relacionada à dissipação dinâmica devido às correntes transientes de chaveamento e à carga e descarga das capacitâncias do circuito, se fosse aplicado a um circuito, num determinado intervalo de tempo, um certo número de vetores de entrada, e se fosse acumulado o número de transições do circuito, seria possível estimar o consumo de potência desse circuito para aquele conjunto de vetores de entrada.

Um fator importante relacionado ao funcionamento do VHDL é o modo de ativação das entidades para que uma descrição seja simulada. Como mostra a Figura 44, quando se descreve um circuito, declara-se uma entidade, no caso entidade *and* (e lógico), e cria-se uma ou mais arquiteturas. Essas arquiteturas descrevem o funcionamento da entidade.

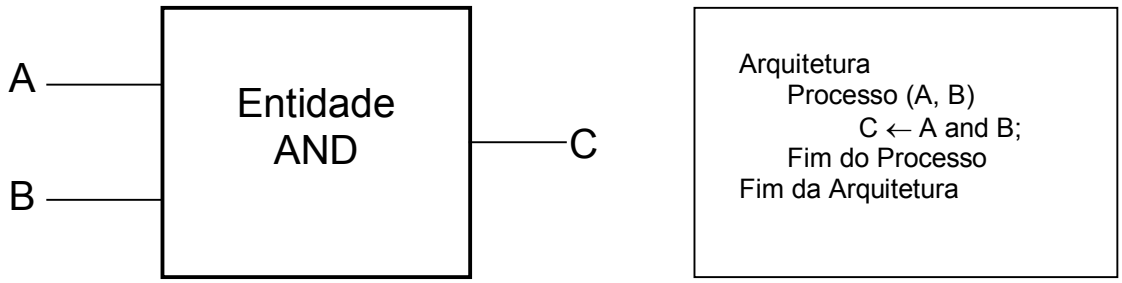


Figura 44 – Entidade e sua descrição da arquitetura

A arquitetura pode conter trechos de código denominados processos, como exemplifica a Figura 44. Esses processos são ativados quando existe uma mudança de estado em pelo menos um sinal que conste da lista de processo (neste exemplo, sinais A e B). Isso nos garante que, sempre que há uma transição de um sinal de entrada de entidade, o código existente no processo é executado.

Como mencionado acima, para que se possa estimar o consumo de potência de um circuito é necessário somar todas as transições do circuito sob teste. Como as descrições dos processos permitem definir seu funcionamento por meio das transições de um grupo de sinais, pode-se construir, sem muita dificuldade, um programa em VHDL que estime o consumo de potência proporcional às transições dos sinais.

A maior limitação dessa técnica está relacionada à falta de informações das descrições comportamentais dos sistemas, por não terem os componentes que definem sua arquitetura. Assim, não se pode determinar com exatidão o peso da transição de um sinal de entrada de um processo em relação a outro, pois a transição do sinal ativa um circuito em que se desconhecem as portas lógicas (consumo de potência) a ele associadas.

A solução é sintetizar a descrição comportamental do circuito e salvar o circuito gerado pela síntese. Desta forma, obtém-se uma descrição *Register Transfer Level* (RTL) com informações das portas lógicas geradas para o circuito. É evidente que esta síntese é feita em relação a uma biblioteca específica de células, neste caso, a biblioteca de células padrão (CMOS *standard cell library*) 0,7 μm ES2.

Após a síntese desse código VHDL, salva-se o circuito também em VHDL, como ilustra a Figura 45. Como abordado acima, cada porta lógica está associada à biblioteca ES2. Para se estimar o consumo, criou-se uma nova biblioteca com todas as portas lógicas existentes nessa biblioteca. Na Figura 45, a biblioteca ES2_pot representa essa modificação, que foi realizada em [72]. As portas lógicas, além de possuírem a descrição do funcionamento, têm também a chamada a um procedimento que escreve em um arquivo o valor característico de consumo de potência da porta lógica que possui uma transição na saída. No final da simulação, basta somar os valores existentes nesse arquivo para se obter uma estimativa do consumo de potência desse circuito.

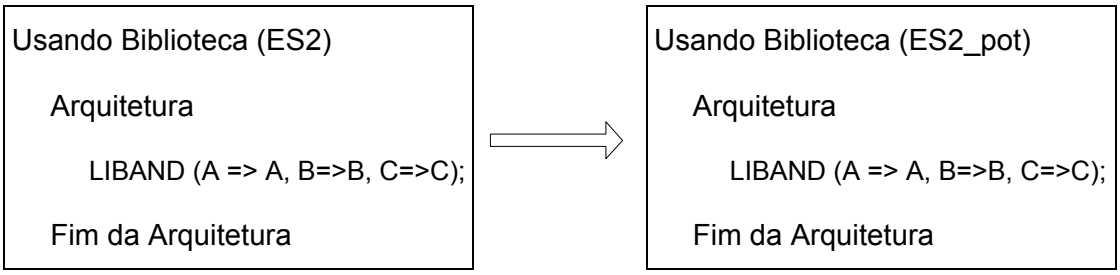


Figura 45 – Ilustração da modificação da biblioteca no VHDL do circuito

Sendo assim, a forma como é feita a estimativa de potência pode ser ilustrada em um exemplo simples por meio da descrição em VHDL, como na Figura 46.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
entity test is
port (Y : out std_logic;
        A, B : in std_logic);
end test;
architecture arch of test is
begin
      process (A, B)
      begin
          Y <= ((not(A) and B) or (A and not(B)));
      End process;
end arch;
```

Figura 46 – Exemplo de descrição comportamental

Após obter a descrição comportamental logicamente correta, obtém-se a síntese dessa descrição e o correspondente esquemático com os circuitos (células da biblioteca), que são salvos em uma descrição VHDL, como o código apresentado na Figura 47.

```
library IEEE;
library ES2;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use ES2.components.all;
entity test is
port (Y : out std_logic;
     A, B : in std_logic);
End test;
architecture arch of test is
begin
     U1: LIBXOR port map (Y => Y, A=> A, B=> B);
End arch;
```

Figura 47 – Esquemático do circuito descrito em VHDL

Para que se possa estimar o consumo de potência, o código do circuito é modificado, substituindo-se os componentes da biblioteca, no caso ES2, pelos

componentes criados para estimar o consumo. A Figura 48 mostra a representação da porta XOR (ou-exclusivo) na biblioteca criada para estimar o consumo. Essa biblioteca deve conter todas as células da biblioteca na qual o circuito foi sintetizado, para que se evite problemas de compatibilidade.

O procedimento *counter_power*, apresentado na Figura 48, é responsável pelo armazenamento do valor do consumo de potência da porta em um arquivo. A necessidade de gravar em arquivo se deve ao fato de não ser possível criar uma variável global para acumular diretamente esses valores. Ou seja, depois de gravar em arquivo todos os valores relacionados ao consumo, executa-se um programa que lê esse arquivo e soma todos os valores nele existentes.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use WORK.pack_power.all;

entity LIBXOR is
port ( Y : out std_logic;
       A, B : in std_logic);
end LIBXOR;

architecture power of LIBXOR is
Begin
       process (A, B)
       variable power, acum: real;
       variable now: std_logic := '0';
       variable before: std_logic;
       begin
              now:= (not(A) AND B) OR(A AND not(B));
              if (now /= before) then
                 power:= 6.01;
              else
                 power:= 0.601;
              end if;
              counter_power(power);
              before:= now;
       end process;
       Y  <=  (NOT (A) AND B) OR (A AND NOT(B)) after 1.4ns;
end power;
```

Figura 48 – Descrição de uma porta XOR para estimar o consumo de potência

A potência estimada em cada componente é a média do consumo por MHz do dispositivo sob teste. O valor dessa potência é computada como função da carga do capacitor de saída para uma determinada freqüência de referência. Esse valor é atualizado na biblioteca, nesse caso ES2, que é expresso em μW/MHz.

Os sinais de teste e o arquivo de saída com os valores para cada transição são mostrados na Figura 49. Pode-se notar que o valor nominal de consumo da célula é de 6,01 μW/MHz. Além disso, nota-se que o quarto valor do arquivo é menor que os demais, exatamente 10% do valor, que é a representação de transição na entrada que não altera a saída e, conseqüentemente, não consome a mesma potência dinâmica. Esse valor é definido pelas especificações fornecidas pelo fabricante (ATMEL ES2 ECPD07 Library Databook, 1996), que associa o valor máximo de 10% para transições que não alteram o valor da saída.

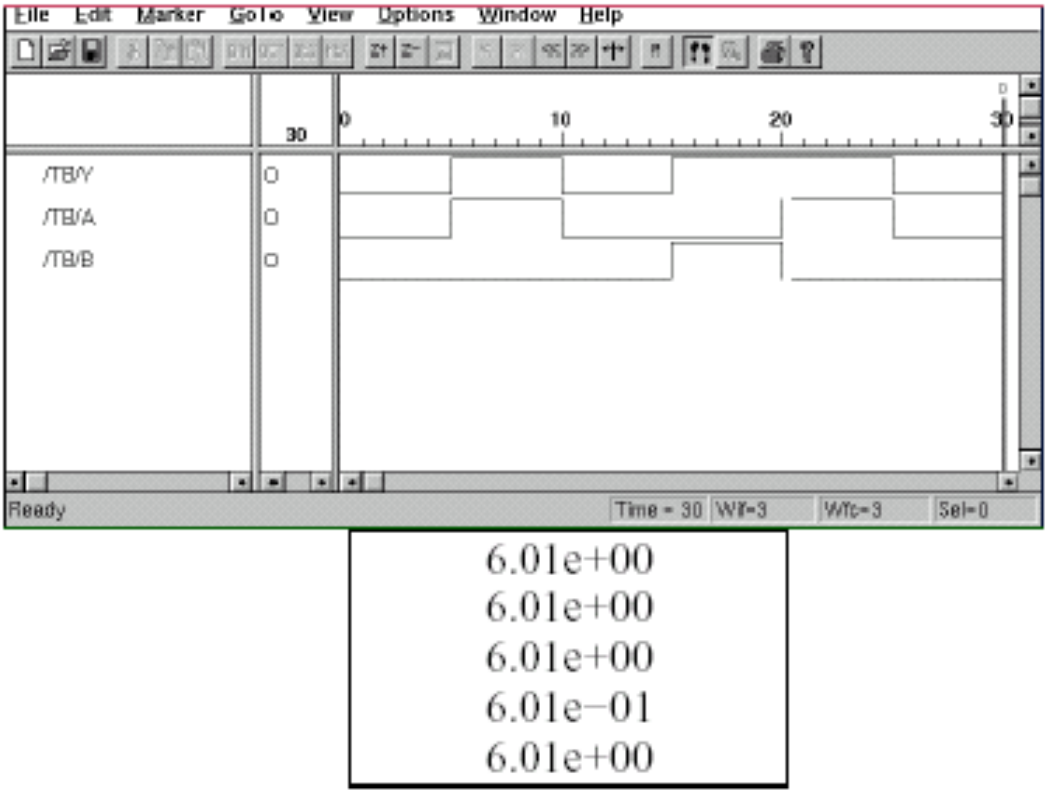


Figura 49 – Estímulos de entrada e o arquivo de consumo

Esse método simples permite comparar implementações em *hardware* de circuitos em relação ao consumo de potência, fazendo com que as simulações de esquemático possam estimar essa importante característica do circuito, apesar da biblioteca empregada para a síntese (ES2 0,7 μm) não apresentar informações a respeito

do consumo. Contudo, essa biblioteca foi modificada para emprego nesta tese e em [72][74] para estimativa do consumo de potência, a partir da alteração dos códigos de suas 76 células.

Com a aplicação desse processo, verificou-se que as especificações da biblioteca de células continham atrasos diferenciados, ou seja, se a transição fosse na porta de entrada A ou B, os atrasos eram diferentes e, além disso, se a transição fosse de subida ($0 \rightarrow 1$) ou de descida ($1 \rightarrow 0$), os atrasos também eram diferentes. Essas diferenças de atraso são muito importantes para o cálculo do consumo de potência, já que ocasionam *glitches* – pulsos momentâneos indesejáveis e inesperados nas entradas ou saídas dos circuitos – nos blocos com funções combinacionais, o que aumenta significativamente o consumo de potência. A Figura 50 mostra o código modificado para a porta XOR levando em conta essas características.

Com base nas versões modificadas da biblioteca para estimativa de potência, foram realizadas experiências na implementação de processadores [72] que demonstraram que o consumo praticamente dobra levando em conta esses diferentes atrasos. Além de comprovar a importância dos atrasos diferenciados das portas para a estimativa de consumo, este trabalho propiciou a confirmação da funcionalidade das entidades, uma vez que os circuitos foram simulados utilizando essas técnicas e não apresentaram alterações em relação à biblioteca ES2 original sem as alterações.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use WORK.pack_power.all;
entity LIBXOR is
port ( Y : out std_logic;
       A, B : in std_logic);
end LIBXOR;
Architecture power of LIBXOR is
Begin
       process (A, B)
       variable power, acum: real;
       variable now: std_logic := '0';
       variable before: std_logic;
       begin
              now:= (not(A) AND B) OR(A AND not(B));
       if (A´event AND now =´0´ AND before=´1´ ) then
              Y<= now after 1.319 ns;
       elsif (B´event AND now = ´0´ AND before =´1´) then
              Y<= now after 1.314 ns;
       elsif (B´event AND now = ´1´ AND before =´0´) then
              Y<= now after 1.073 ns;
       elsif (A´event AND now = ´1´ AND before =´0´) then
              Y<= now after 0.956 ns;
       Else
              Y<= now after 0.956 ns;
       end if;
       if now /= before then
              power:= 6.01;
       else
              power:= 0.601;
       end if;
       counter_power(power);
       before:= now;
end process;
end power;
```

Figura 50 – Descrição de uma porta XOR com atrasos diferenciados nas entradas

5.2. Resultados

Por meio das ferramentas do Synopsys, vhdlan, que compila os arquivos VHDL, e vhdldbx, que simula esses arquivos, foi obtida a estimativa dos valores de consumo de potência para a implementação das arquiteturas apresentadas no Capítulo 3.

São comparadas as diferentes implementações em *hardware* dos algoritmos criptográficos RSA, ECC, Blowfish e Godzuk. O desempenho global de cada implementação, em termos de consumo de potência, foi avaliado para um mesmo conjunto de vetores de teste (com blocos de 1024 *bytes*) [75].

A estimativa de consumo de potência e energia das quatro arquiteturas é apresentada na Tabela 10.

Algoritmo	**Atraso 1024 *bytes***	**Energia (mW/Hz)**	**Potência (mW)**
RSA 1024	215 ms	384,5	1788
ECC 163	19 ms	5,13	270
Blowfish	41 μs	$61{,}2 \times 10^{-3}$	1493
Godzuk	39 μs	$83{,}3 \times 10^{-3}$	2136

Tabela 10 – Medições comparativas de energia e potência

Para avaliar o consumo de potência dos algoritmos criptográficos, foram usados vetores de teste com mensagens de tamanhos múltiplos de 1024 *bytes*. Cabe ressaltar que, apesar dos consumos de potência das implementações serem muito próximos, o gasto de energia para cifrar e decifrar as mensagens são diferentes. Para os algoritmos simétricos, a energia é de algumas dezenas de μJ, enquanto que, para as implementações dos algoritmos assimétricos, 100 a 10.000 vezes maior.

5.3. Considerações

Essa análise permitiu concluir que existe mais uma vantagem dos algoritmos criptográficos simétricos em relação aos algoritmos assimétricos. Não apenas o desempenho, como abordado anteriormente, mas também o consumo de potência, tão determinante no desempenho das redes móveis, estabelecem que os algoritmos simétricos têm uma melhor aplicação nesse tipo de rede, pois têm um impacto menor e permitem uma maior autonomia.

A vantagem, contudo, da criptografia assimétrica não necessitar de um tipo de gerenciamento de chaves tão complexo quanto os algoritmos simétricos é antes uma necessidade das redes móveis, pois possibilita uma maior segurança no estabelecimento das chaves.

A solução ideal é combinar as duas técnicas de tal forma que se possa usufruir da vantagem da segurança do algoritmo assimétrico e da vantagem da rapidez de execução, além do baixo consumo de energia do algoritmo simétrico.

Um método híbrido, que será apresentado no próximo Capítulo, baseia-se em usar criptografia assimétrica para a troca de uma chave secreta temporária – como essa etapa é única e fundamental para a confidencialidade total do intercâmbio, o tempo adicional gasto na criptografia é compensado pela segurança do sigilo oferecido – e em usar criptografia simétrica para proteger as outras mensagens – essa etapa é repetida inúmeras vezes, devendo necessariamente usar um método rápido de criptografia. Como a chave criptográfica é intercambiada de uma maneira sigilosa e é temporária, o algoritmo simétrico fornecerá uma boa segurança.

Capítulo 6

Simulação e resultados

Existem vários métodos e ferramentas capazes de ajudar o projetista na tarefa de gerar um ambiente próximo ao real, com um custo mais baixo, e que possam exigir do sistema em teste as mesmas respostas de um cenário verdadeiro.

Este Capítulo aborda, de forma objetiva, as ferramentas de auxílio no desenvolvimento e simulação deste trabalho. O Synopsys, instalado em estação de trabalho SUN Ultra 10 no Laboratório de Projeto de Circuito – LPC/UFRJ, é usado não apenas na fase de projeto dos circuitos eletrônicos, mas também na etapa de testes para validar a implementação, por meio de simulação lógico-temporal.

Com base no circuito digital que implementa os algoritmos criptográficos, é desenvolvido o sistema de segurança para as redes sem fio. Para testar esse sistema, é necessário criar o ambiente semelhante ao real no qual será aplicado. O NS-2 é a ferramenta capaz de avaliar o desempenho de um sistema de segurança em uma rede sem fio, pois é capaz de implantar uma rede e definir quais protocolos serão utilizados nas suas mais diversas camadas. Os resultados obtidos através desses experimentos foram precisos e forneceram um grande respaldo na avaliação do sistema, pois puderam ser testados em todas as situações possíveis, fossem elas favoráveis ou desfavoráveis aos sistemas de segurança apresentados.

6.1. Ambiente de simulação

A principal diferença entre os Circuitos Integrados de Aplicação Específica (ASIC) e os circuitos integrados regulares é a inclusão do *software* como parte da produção. O desenvolvimento desse tipo de *hardware*, normalmente, é feito por ferramentas de edição ou por linguagens de especificação de *hardware*, cujos esquemáticos são obtidos após as sínteses dessa descrição.

As ferramentas de edição de esquemático existentes no mercado são, normalmente, acompanhadas de bibliotecas (analógicas/digitais), de forma a prover meios para o projeto de qualquer circuito integrado VLSI em tecnologia CMOS. Os programas Cadence e Synopsys são os dois principais programas profissionais encontrados no mercado e aceitam como linguagem de *hardware* o VHDL e o Verilog.

Os circuitos digitais desenvolvidos neste trabalho, que implementam algoritmos criptográficos, foram descritos com base no domínio de representação estrutural. A ferramenta usada para a obtenção da síntese lógica desses circuitos foi o Synopsys, que ofereceu redução de tempo de projeto e teste, por possuir um ambiente de validação integrado à ferramenta.

Os algoritmos criptográficos projetados em *hardware* podem suportar um desempenho superior à implementação equivalente em *software*. Há duas formas principais de implementação de circuitos integrados, que podem ser desenvolvidas por meio dessas ferramentas de projeto: com os circuitos integrados de aplicações específicas (ASIC) e com os dispositivos lógicos programáveis EPLD e FPGA.

Os circuitos ASIC têm algumas vantagens sobre os EPLD e FPGA. Primeiramente, eles são normalmente mais rápidos, pois são projetados especificamente para o problema, ao passo que os dispositivos programáveis são componentes de lógica genéricos, representados por matrizes de chaves, programados com o uso de elementos de lógica. Além disso, os ASIC são normalmente menores e consomem menos potência, pois os dispositivos programáveis têm lógica extra para manter a vantagem da reprogramação.

A simulação desempenha um papel importante nas redes de computadores. Uma das vantagens da simulação é o custo/benefício proporcionado. Geralmente, uma

simulação tem um custo relativamente pequeno, se comparado a uma implementação, e o resultado pode chegar bem próximo ou igualar-se ao obtido pela implementação. Todavia, a simulação apresenta vários tipos de limitações que dependem do simulador utilizado.

Vários são os simuladores utilizados para projetar e avaliar redes de computadores. O NS-2 é um simulador de rede orientado a objeto, escrito em C++ e tem como ambiente de programação a linguagem interpretada Otcl, que simula uma variedade de redes fixas ou móveis [76].

Neste trabalho optou-se pelo NS-2, não apenas pela flexibilidade acima descrita, mas também por preencher os requisitos esperados de robustez, agilidade, simplicidade na criação e iniciação dos componentes da rede e facilidade na obtenção dos resultados. Outro ponto importante na escolha desse simulador foi o conhecimento de seu funcionamento pelos integrantes do Grupo de Teleinformática e Automação do Departamento de Engenharia Elétrica da Universidade Federal do Rio de Janeiro, onde o trabalho se desenvolveu, que já possuem experiência no seu emprego, proveniente de várias pesquisas desenvolvidas.

O principal objetivo dessa simulação é obter e avaliar as conseqüências da inserção de serviços de segurança nas redes sem fio e determinar o seu impacto no desempenho global da rede. Com a simulação da inserção desses *hardwares* de segurança nos protocolos das redes sem fio, foram obtidos resultados apresentados em gráficos que mostram a variação da taxa de entrega dos pacotes em relação a diferentes graus de mobilidade e em relação ao número de fontes (nós) geradoras de tráfego. A partir desses resultados, foi possível avaliar o desempenho dos algoritmos criptográficos nesse tipo de rede.

Em resumo, um dos maiores desafios do modelo de segurança é definir em qual camada o *hardware* deve estar presente para garantir melhor desempenho. Existem duas propostas: camada fim-a-fim (aplicação) e camada de acesso à rede (enlace).

Na proposta de inclusão na camada fim-a-fim (aplicação), a segurança da aplicação ocorre em um sistema no qual os dados são cifrados (*off-line*) e depois transmitidos. Todas as camadas intermediárias processam somente dados cifrados. Embora garanta uma segurança fim-a-fim, esta configuração oferece um nível de segurança apenas aos dados da aplicação, mantendo o roteamento da rede sem proteção,

além de ter um sistema mais complexo de gerenciamento de chaves.

Contudo, existe maior transparência na camada de acesso à rede, pois todos os dados de controle (roteamento) e de usuário são cifrados (*on-line*) antes de serem transmitidos. As maiores vantagens são uma maior segurança no tráfego IP, uma gestão de chaves mais simples e um melhor desempenho, uma vez que os dados são cifrados *on-line* e entregues à camada física para serem transmitidos. Contudo, como desvantagem, consegue-se autenticar apenas os terminais, e não o usuário / aplicação.

Sendo assim, optou-se neste trabalho pela integração do *hardware* nas camadas de acesso à rede, particularmente entre as camadas de rede e *data link*, não apenas por imprimir maior segurança ao tráfego, mas sobretudo por apresentar um sistema de segurança mais completo e com melhor desempenho.

Para investigar como os modelos de segurança propostos neste trabalho se comportam, e qual o seu impacto nos protocolos das redes em ambientes reais, definiu-se um cenário de aplicações com um padrão para os nós.

Embora exista uma variedade de características para criação de vários cenários de aplicações das redes sem fio, optou-se por um cenário simplificado, onde o principal objetivo foi a validação dos resultados da inserção dos algoritmos criptográficos (RSA, ECC, Blowfish e Godzuk), determinando seu impacto no desempenho global da rede, pela verificação da variação da taxa de entrega dos pacotes em relação a diferentes graus de mobilidade e em relação ao número de fontes (nós) geradoras de tráfego.

O objetivo básico do modelo de segurança proposto é fornecer segurança para esse tipo de cenário padrão apresentado, comprometendo minimamente o desempenho global da rede; permitindo ainda, com os resultados obtidos, adaptar esse modelo de segurança para os cenários específicos apresentados que têm características próprias, ajudando a criar outras soluções específicas.

Os sistemas criptográficos foram adaptados como proteção aos mecanismos de operação das redes. Por razões de eficiência, a melhor solução foi usar os algoritmos RSA e ECC para autenticação e garantia da identidade real dos usuários remotos que desejassem se comunicar e ter acesso à rede (*login)*, e os algoritmos Blowfish e Godzuk para estabelecer uma comunicação segura com privacidade entre os nós, após o estabelecimento do enlace.

6.2. Representação do cenário

A nova arquitetura, obtida com a integração dos sistemas de segurança desenvolvidos em *hardware* nas aplicações das redes sem fio, foi validada no NS-2 versão 2.1b8a, que possui uma extensão atualizada para redes móveis, contendo vários protocolos de roteamento.

Uma aplicação das redes sem fio encontra-se no domínio militar, pois permite que unidades no campo de batalha comuniquem-se em qualquer lugar, a qualquer hora, sem a exigência de qualquer infra-estrutura fixa. A perda de qualquer unidade não romperá a rede, desde que outras unidades possam dela participar.

Esse cenário exige uma grande adaptação das redes, pois, nesse ambiente, os nós estão espalhados em grandes áreas, alguns com baixa mobilidade e, a qualquer momento, pode ocorrer a perda de um nó. Existe um tráfego bem peculiar entre os nós, em que aproximadamente 20% deles são grandes fornecedores de informações e existe uma formação natural de grupos com interesses de informações comuns (pelotões, companhias, batalhões).

A mobilidade relativa entre os nós desses grupos (pelotões, companhias) podem ser muito baixa, sobretudo se for considerada uma situação de defesa do terreno, em que os nós permanecem em manutenção da posição. A Figura 51 mostra uma representação simplificada desse cenário, onde os grupos de nós envolvidos pelo tracejado entre eles têm mobilidade relativa zero. Nesse contexto, foi escolhido esse cenário para avaliar a eficiência das arquiteturas dos algoritmos implementados nesse trabalho e obter o impacto do seu uso.

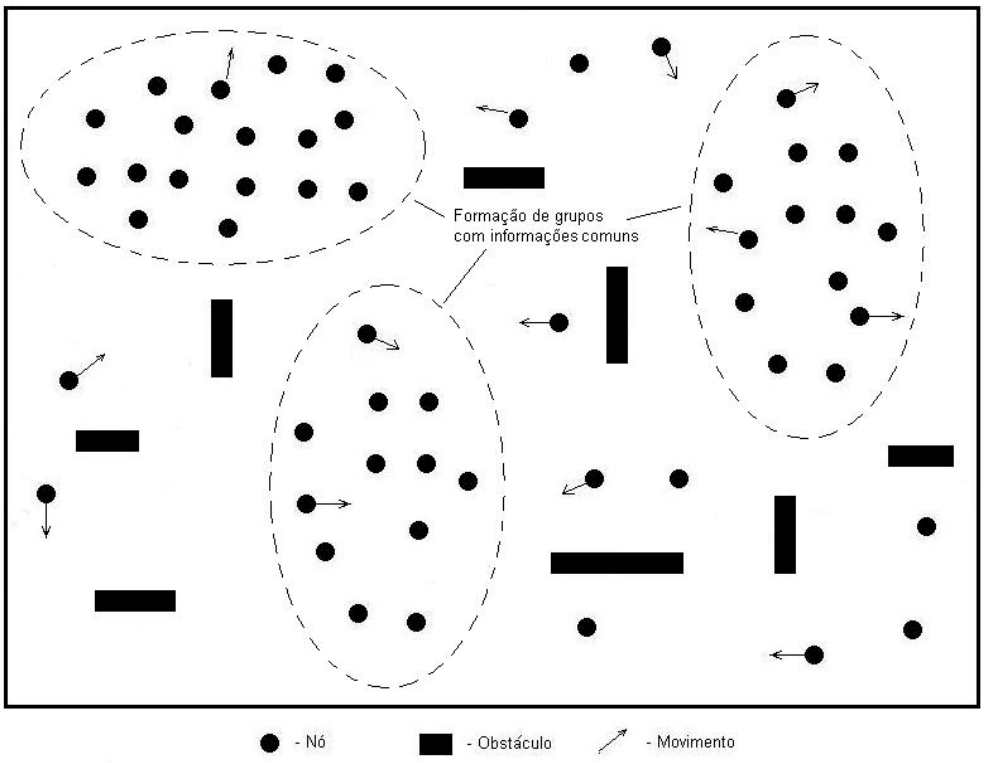


Figura 51 – Cenário campo de batalha

O estudo de simulação, para esse cenário, foi executado no ambiente (NS-2). Os cenários criados contêm um número aleatório de nós e cada nó é posicionado aleatoriamente em um ambiente retangular maior que 1 km^2. As simulações desses cenários são semelhantes às aproximações empregadas em [77], adaptadas do cenário reunião, que possui nenhuma mobilidade entre os nós ou uma mobilidade baixa, em que o nó não sai do raio de alcance do rádio do nó transmissor. O *software* de simulação foi modificado para modelar a inclusão dos circuitos de criptografia em cada nó. Essa simulação levou em conta os atrasos provocados pelos efeitos dessa inclusão de *hardware*.

Com base nos resultados das simulações será proposta a integração, aos protocolos de comunicações, dos mecanismos de segurança desenvolvidos neste trabalho.

O simulador utiliza o cenário definido como entrada para variar a inserção dos nós (entre 6 e 30 nós), além da definição do padrão de comunicação, obtendo valores aleatórios de posição numa área de 1500 m x 1500 m. Assim, cada nó possui uma determinada posição, na qual se mantém durante o tempo de estabelecimento da comunicação. Depois desse tempo, acrescenta-se um maior número de nós ao cenário,

escolhendo-se um ponto aleatório para essa inserção.

Para as diferentes quantidades de nós no cenário foram obtidos diversos posicionamentos distintos. Dessa forma, podem ser utilizados vários padrões de testes diferentes.

O tráfego escolhido para a comparação entre os sistemas foram implementados utilizado-se o padrão IEEE 802.11, que define a camada física e de enlace da rede. A taxa do canal é de 11 Mbps. O protocolo de roteamento usado é o *Ad Hoc On-Demand Distance Vector* (AODV), utilizado em redes *Ad Hoc*. O protocolo de transporte é o *User Datagram Protocol* (UDP), sendo o tráfego gerado pela fonte do tipo *Constant Bit Rate* (CBR) de 1 Mbps.

Além disso, foram utilizadas diversas quantidades de fontes de tráfego CBR (de cinco a trinta fontes), permitindo uma avaliação por meio de dois cenários. O primeiro, uma cadeia de alguns nós com uma comunicação saindo do primeiro nó e sendo dirigida ao último, após ser encaminhada pelos demais, que funcionam apenas como roteadores. O número de nós foi variado de cinco a dez. A distância entre cada elemento da cadeia e seu vizinho foi fixada em 150 m. O objetivo desse cenário era verificar o impacto da inserção dos circuitos dos algoritmos criptográficos em ambientes de múltiplos saltos.

O segundo cenário foi definido a partir da alocação de números variáveis de pares de nós em localizações também variáveis, para verificar o impacto da inserção dos algoritmos criptográficos na vazão da rede.

A métrica escolhida para análise dos protocolos foi a taxa de entrega de pacotes. Essa taxa foi definida como a relação entre o número de pacotes originados pelas fontes CBR do nó de origem e o número de pacotes recebidos pela fonte CBR do nó de destino. A taxa de entrega de pacotes é uma métrica importante, porque descreve a taxa de perda dos pacotes, o que afeta a vazão máxima que a rede pode suportar. Essa métrica caracteriza a eficácia do protocolo da rede.

A modelagem do novo sistema, protocolo em *software* com segurança em *hardware*, tem que ser representada de forma mais próxima da realidade para ser validada. Para isso, optou-se por uma representação do *hardware* através dos seus atrasos provocados em cada tipo de mensagem processada. Esse atraso foi colocado entre a camada MAC e a camada física. Esse posicionamento permite que não apenas os dados sejam criptografados, mas também os pacotes de controle e cabeçalhos do

protocolo MAC 802.11.

Uma vez que a mensagem chega a um nó depois de atravessar um modelo de canal, entra no *hardware*-modelo de segurança (algoritmos criptográficos desenvolvidos) localizado entre a ligação de dados e a camada de controle de acesso ao meio. Esse *hardware*-modelo de segurança é representado pelo atraso do processamento do circuito digital que implementa os algoritmos criptográficos, incluindo os tempos gastos com o controle.

A Tabela 11 apresenta os atrasos de cada algoritmo criptográfico. Como o *hardware* foi colocado na camada enlace, cada um desses atrasos será implementado no NS-2 como um tempo de espera nesta camada. Dessa forma, tem-se o impacto da inclusão desses circuitos nos protocolos de roteamento.

Característica	**Blowfish (256 *bits*)**	**Godzuk (128 *bits*)**	**RSA (512 *bits*)**	**RSA (1024 *bits*)**	**ECC 163**
Freqüência do *clock*	50 MHz	78 MHz	15,6 MHz	6 MHz	38 MHz
Taxa de cifragem	200 *Mbits*/s	255 *Mbits*/s	260 *kbits*/s	38 *kbits*/s	300 *kbits*/s
Mensagem 1024 *bytes*	41μs	39 μs	52 ms	215 ms	19 ms

Tabela 11 – Atrasos das implementações dos algoritmos criptográficos

Cabe ressaltar que os tempos necessários para estabelecimento das chaves também devem ser considerados, sobretudo em cenários em que não exista qualquer prévio contato entre os nós, pois neste ambiente exige-se inicialmente um volume intenso de mensagem para que cada nó obtenha suas chaves, aumentando assim o tempo de processamento de mensagens nos sistemas com segurança, em relação aos sistemas sem segurança.

6.3. Resultados

A segurança em redes cabeadas está amadurecida em diversas pesquisas e ambientes comerciais. Contudo, o desenvolvimento da segurança em redes sem fios é ainda um campo em evolução. Assim, foi realizado um estudo para definir os principais

aspectos e requisitos de *hardware*, como um estudo inicial e uma alternativa de padrão para as redes sem fio [30].

Constatou-se que os algoritmos criptográficos implementados em *hardware* podem servir como um padrão para as redes sem fio por duas razões principais. A primeira é o alto desempenho em operações de cifragem e decifragem de dados. Uma vez que todas as informações (dados e controle) usadas em redes móveis têm que ser protegidas com criptografia, verificou-se que é aconselhável o emprego de algoritmos com desempenho maior que 1 *Mbits*/s, pois esta é a especificação de rádio para essas redes [5][78]. A segunda razão é o consumo de potência do circuito requerido para essas aplicações, pois quanto maior o consumo de potência, menor a autonomia do sistema. Muitos outros algoritmos de criptografia são mais rápidos que os apresentados neste trabalho, valores já mencionados no Capítulo 3. Porém, por registrarem taxas mais altas, eles precisam de mais área e maior consumo de potência [27].

Dessa forma, a dificuldade para se definir um algoritmo de criptografia como padrão para as redes sem fio reside no compromisso entre segurança, consumo de potência e desempenho.

O Blowfish e o Godzuk são exemplos de algoritmos criptográficos simétricos que resolvem esse problema, pois foi mostrado que são algoritmos simétricos com 64 *bits* de dados e podem operar com 256 e 128 *bits* de chave, respectivamente. Os algoritmos ECC e RSA, exemplos de algoritmos criptográficos assimétricos, também atendem às necessidades das redes sem fio em relação aos critérios de desempenho e consumo de potência, sobretudo quando empregados em conjunto com um dos algoritmos simétricos.

O desempenho dos circuitos que executam os algoritmos criptográficos desenvolvidos e apresentados no Capítulo 3 são definidos de acordo com o *clock* e com o tamanho em *bits* dos blocos de dados processados simultaneamente. A Tabela 11 resume esses resultados para cada algoritmo criptográfico.

A implementação de Blowfish que usa 64 *bits* de dados com um *clock* de 50 MHz tem um processamento de 200 *Mbits*/s. Dessa forma, executa os 1024 *bytes* de dados em um tempo aproximado de 41 μs [64].

O algoritmo simétrico Godzuk, que também usa 64 *bits* de dados, mas com uma chave de 128 *bits*, obteve um *clock* de 78 MHz e um processamento de 255 *Mbits*/s.

Dessa forma, executa os 1024 *bytes* de dados em um tempo aproximado de 39 μs.

A freqüência final do circuito que implementa o algoritmo RSA é de aproximadamente 15 MHz para chaves de 512 *bits* e 9 MHz para chaves de 1024 *bits*, o que é bem aceitável para um circuito que executa operações matemáticas com números de grande comprimento. Sua implementação obteve as médias de 260 *kbits*/s e 38 *kbits*/s de taxa de transmissão que processa os 1024 *bytes* de uma mensagem em aproximadamente 52 ms e 215 ms, respectivamente.

O desempenho do algoritmo criptográfico ECC com chave de 163 *bits* foi obtido a partir dos circuitos desenvolvidos para o algoritmo RSA. Obteve-se uma freqüência de *clock* de 38 MHz, com uma taxa de aproximadamente 300 kbps, para uma mensagem de 1024 *bytes*, com um atraso próximo a 19 ms.

Como já mencionado, a utilização dos algoritmos assimétricos torna-se ineficiente para cifrar textos grandes, devido à baixa velocidade do processo de cifragem e decifragem, além de exigirem chaves muito grandes para serem considerados seguros.

Verifica-se ainda que ambos os tipos de algoritmos criptográficos oferecem vantagens e desvantagens. Nenhum dos sistemas simétrico e assimétrico, separadamente, oferece total eficiência no processo de cifragem e decifragem, no que se refere a número de chaves, velocidade de processamento e distribuição de chaves.

Um sistema criptográfico híbrido combina o uso das potencialidades que os algoritmos de criptografia simétrica e assimétrica possuem para tornar a comunicação segura, de forma a obter as vantagens que cada sistema oferece [17][79]. Um exemplo de sistema criptográfico híbrido de sucesso em *software* é o *Pretty Good Privacy* (PGP), que significa "privacidade muito boa", tendo sido inventado em 1991 [79].

Sendo assim, este trabalho aproveitou o êxito da criptografia híbrida em *software* e testou várias implementações de segurança com base nos algoritmos desenvolvidos em *hardware*, empregando-os tanto sozinhos, mas sobretudo em conjunto (um simétrico e outro assimétrico). O objetivo do sistema é estabelecer uma sessão de troca de dados segura com um conjunto de terminais em um meio sem fio desdobrado como uma rede *Ad Hoc*. No início de uma sessão, todos os terminais têm um algoritmo simétrico (neste caso, Blowfish ou Godzuk) com suas respectivas chaves (privadas), e também um dos dois algoritmos assimétricos (RSA ou ECC) e suas chaves

privada e pública. Dois terminais que queiram se comunicar trocam a chave do algoritmo simétrico, esta cifrada pelo algoritmo assimétrico. Por exemplo, as chaves do algoritmo RSA são usadas para codificar a chave privada do algoritmo Blowfish. Assim, os dois terminais codificam mensagem usando o algoritmo Blowfish. Quando a sessão de comunicações termina, todas as chaves podem ser descartadas.

Nesse exemplo, não importa se alguém está escutando às escondidas a conversação inteira. Sem as chaves privadas do algoritmo RSA, não há como se obter as chaves do algoritmo Blowfish e, dessa forma, não se pode decifrar as mensagens passadas entre os dois terminais. Outra conseqüência importante é o desempenho global. Como são usadas apenas chaves do algoritmo RSA para cifrar as chaves de 256 a 448 *bits* do algoritmo Blowfish, e como os dados da mensagem são cifrados pelo algoritmo Blowfish, que é sensívelmente mais rápido que o RSA, o desempenho global é quase tão rápido quanto o desempenho de um sistema que emprega apenas o algoritmo Blowfish. Para uma mensagem de 1024 *bytes* de dados, o processo dura aproximadamente 53 ms. Porém, se forem transmitidos 10 pacotes de dados de 1024 *bytes*, durará menos que 54 ms, como mostrado na Figura 52, pois a sessão que codifica a chave privada do Blowfish é realizada apenas uma vez com o algoritmo RSA.

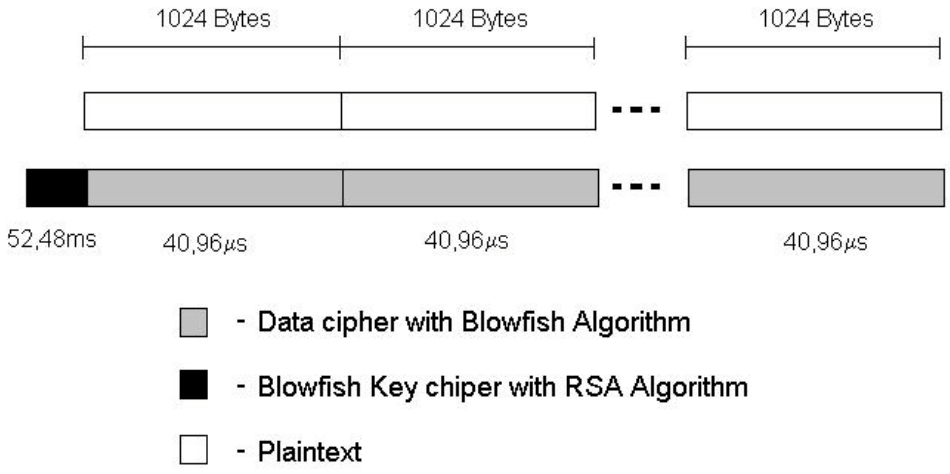


Figura 52 – Tempo de cifragem dos blocos

6.3.1. Resultados de simulação para o primeiro cenário

O primeiro cenário, definido por um conjunto de nós dispostos em cadeia, foi criado para avaliar o impacto da inserção de *hardware* criptográfico em um ambiente com múltiplos saltos. Optou-se por este cenário por representar, de forma simplificada, a disposição dos terminais no cenário campo de batalha na situação de defesa. Para a área de simulação, utilizou-se um campo retangular de 2000 m x 1000 m, com uma distribuição de um conjunto de nós variando entre seis e dez grupos. Cada nó tem um raio de alcance de 250 m e essa área quadrada limita o número de saltos. O tempo de simulação foi de 500 segundos, que é o tempo médio para um volume de mensagens realista. As métricas utilizadas para comparar o desempenho dos protocolos foram:

- atraso de pacotes de dados – inclui todos os possíveis atrasos causados pela latência da descoberta de rotas, propagação, atrasos devido a retransmissões do MAC e tempos de transferência;
- taxa média de *bits* por segundo de controle e dados transmitidos e entregues na rede.

Os resultados obtidos com as simulações mostram o atraso dos pacotes e a taxa de transmissão de dados, demonstrando que o desempenho da rede é prejudicado apenas quando ocorrem múltiplos saltos (mais de 10 saltos). Para esse cenário, foram executados três conjuntos de simulações [80]. No primeiro, foi utilizado apenas o algoritmo RSA para cifrar todos os dados. No segundo, a simulação empregou o algoritmo Blowfish para cifrar todos os dados. Finalmente, o último usou ambos os algoritmos, sendo o RSA para cifrar a chave do Blowfish protegida, e o Blowfish para cifrar os dados. Para comparar com o *hardware* criptográfico, foram executadas simulações com o desempenho da execução dos algoritmos criptográficos, também desenvolvidas em *software*. A Figura 53 mostra os resultados do atraso dos pacotes em função do número de nós para o algoritmo criptográfico RSA (1024 *bits*) em *software*, usando e não usando criptografia.

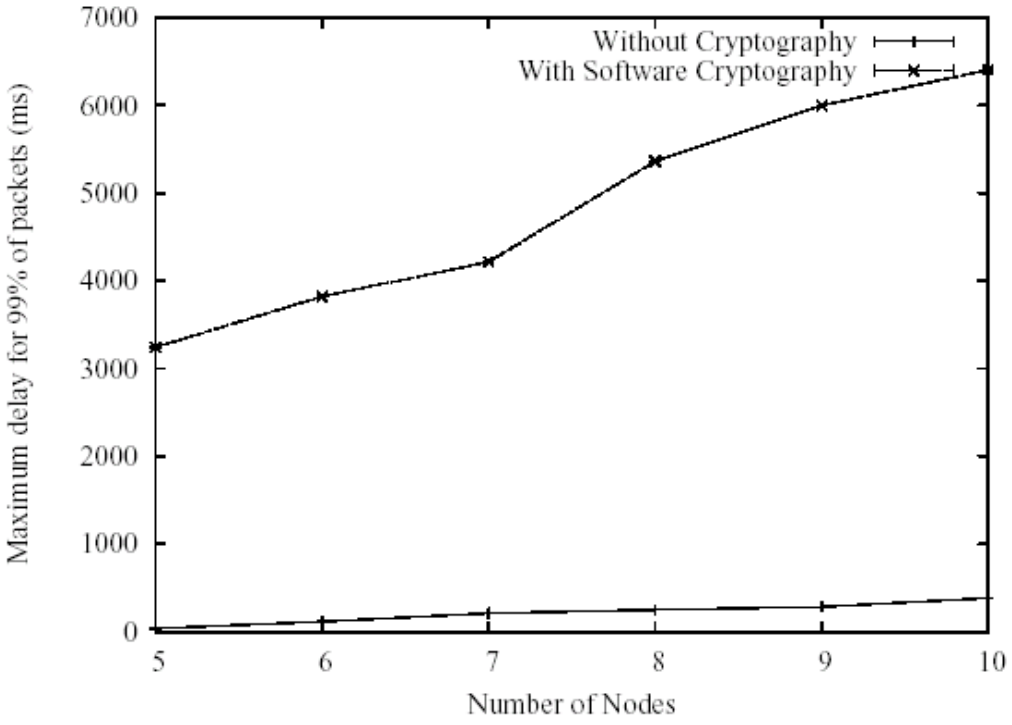


Figura 53 – Atraso do algoritmo criptográfico RSA em *software*

A Figura 54 mostra o desempenho do algoritmo criptográfico Blowfish, também desenvolvido em *software*, comparando-o com a rede sem os recursos de segurança.

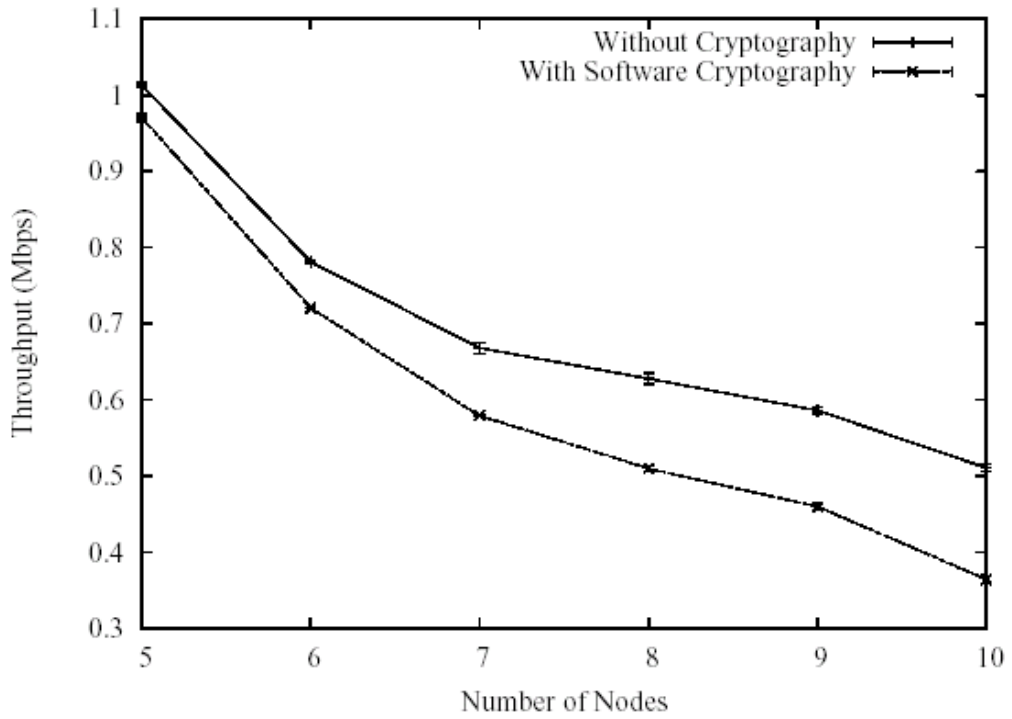


Figura 54 – Taxa de transmissão do algoritmo Blowfish em *software*

A Figura 55 compara as taxas de transmissão de uma rede sem criptografia e de outra com o algoritmo criptográfico RSA. Na Figura 56 são apresentadas as curvas correspondentes aos atrasos dos pacotes, também para uma rede com e sem a implementação do algoritmo RSA. Pode-se observar que o desempenho em ambos os casos é praticamente o mesmo, mas apenas para um número reduzido de nós (cinco ou seis nós).

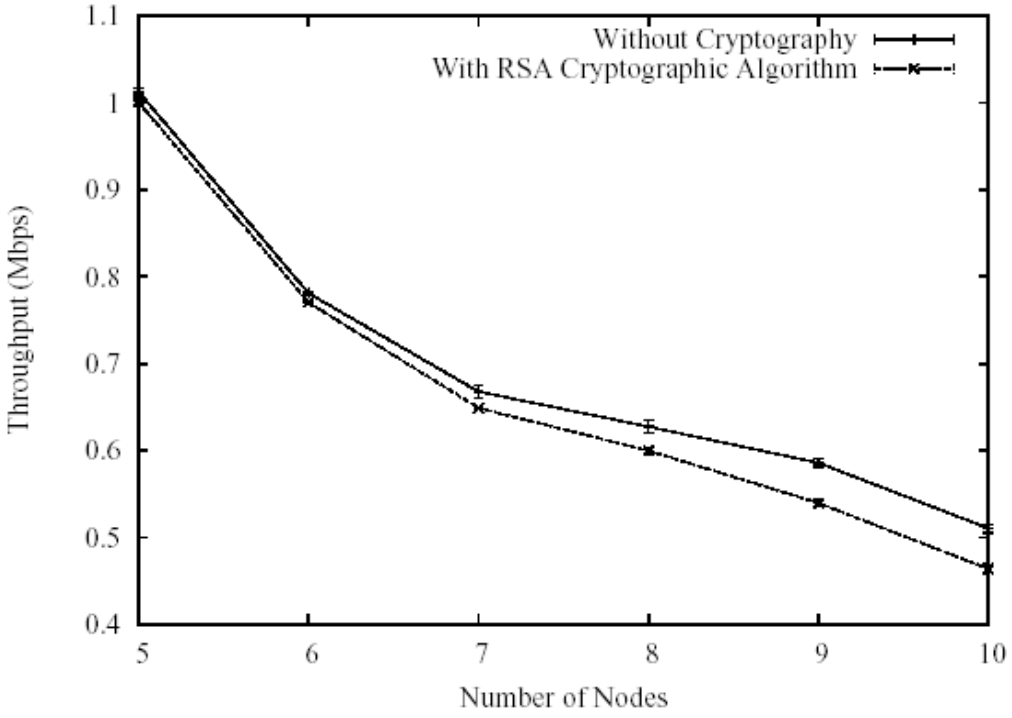


Figura 55 – Taxa de transmissão do algoritmo RSA em *hardware*

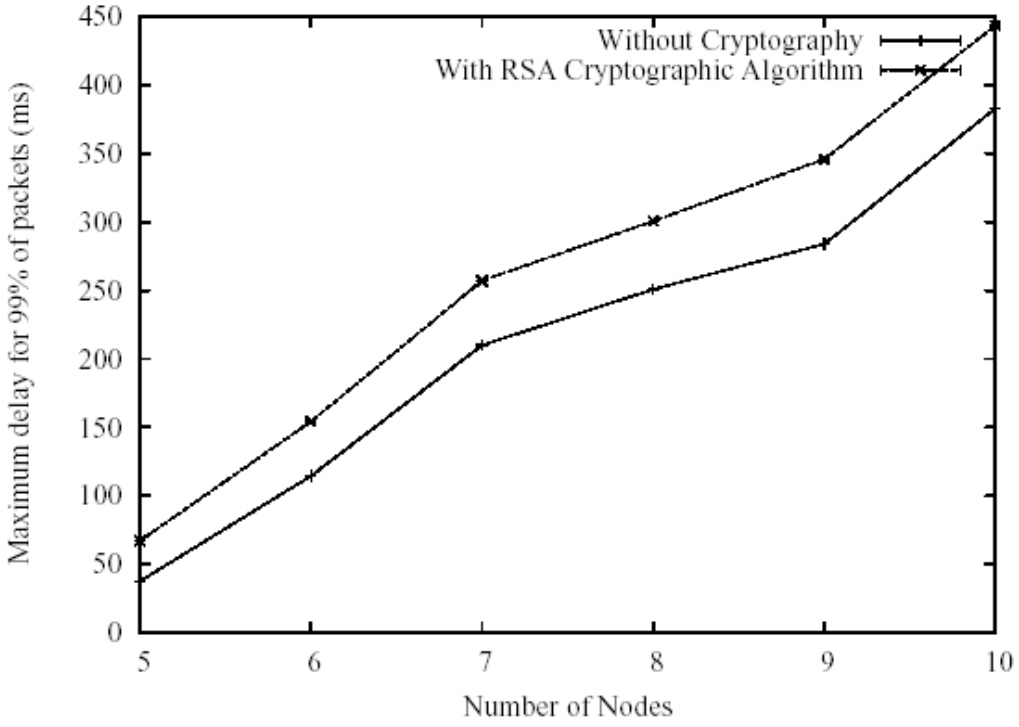


Figura 56 – Atraso do algoritmo RSA em *hardware*

As Figuras 57 e 58 apresentam, respectivamente, os atrasos dos pacotes e a taxa de desempenho para uma rede com a implementação do algoritmo criptográfico Blowfish nos seus nós, sendo cada um dos gráficos comparado com as respectivas redes sem a implementação desse algoritmo. Pode-se constatar que, dado o alto desempenho do algoritmo Blowfish, seu impacto no desempenho da rede é mínimo e imperceptível, mesmo quando a rede possui um número elevado de nós. Contudo, suas características exigem um complexo sistema de estabelecimento das chaves criptográficas, dificultando seu emprego.

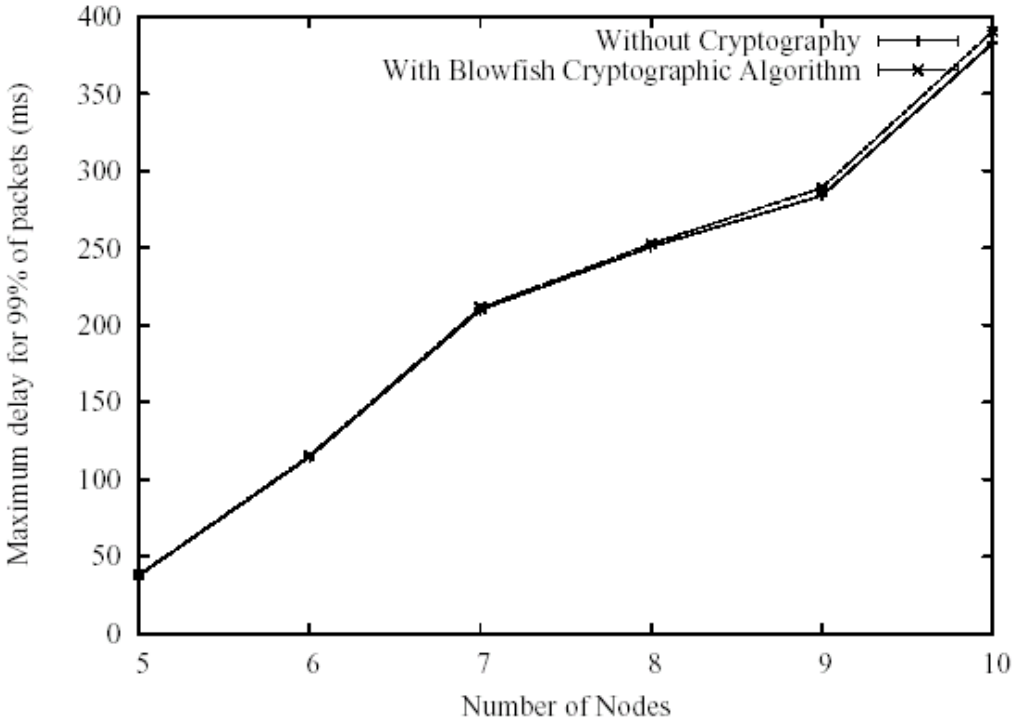


Figura 57 – Atraso do algoritmo Blowfish em *hardware*

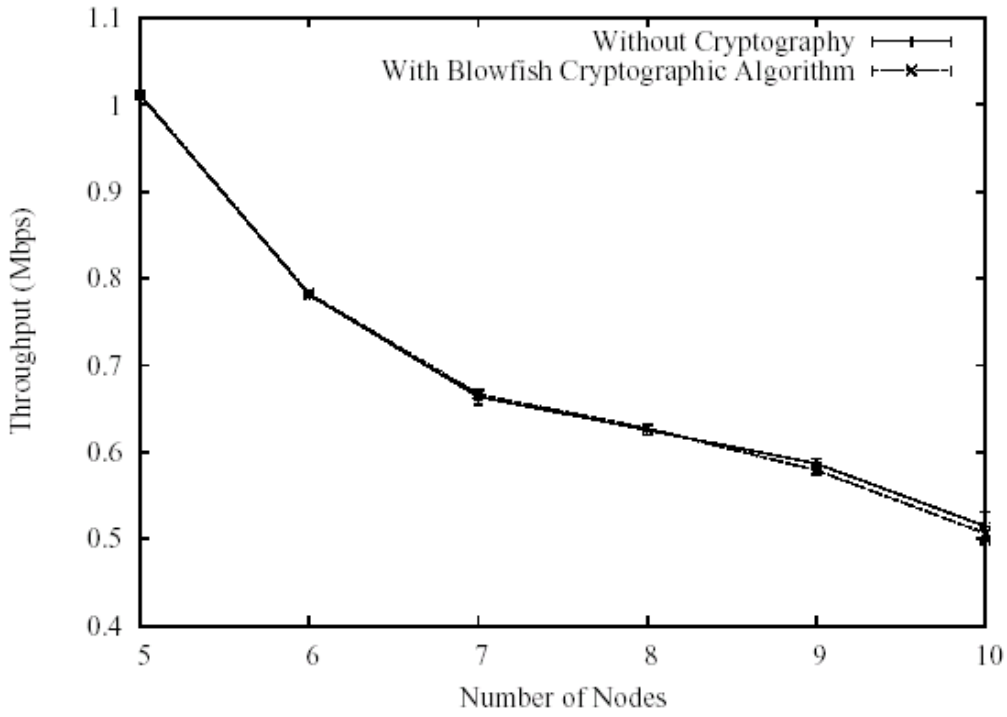


Figura 58 – Taxa de transmissão do algoritmo Blowfish em *hardware*

As Figuras 59 e 60 apresentam, respectivamente, os atrasos dos pacotes e a taxa de desempenho para uma rede com a implementação dos algoritmos criptográficos RSA

e Blowfish nos seus nós, sendo o RSA para cifrar a chave do Blowfish, que é transmitida protegida, e o Blowfish para cifrar os dados. Também nesses gráficos a comparação é feita com as respectivas redes sem a implementação desses algoritmos. Pode-se constatar que praticamente não existe diferença entre os gráficos obtidos apenas com o uso do Blowfish. Como esse algoritmo, que possui alto desempenho, é empregado para a cifragem dos dados, e o RSA é usado apenas uma vez para cifrar a chave do Blowfish, o desempenho conjunto é praticamente o mesmo que usar apenas o algoritmo Blowfish, com a vantagem de tornar o estabelecimento das chaves criptográficas mais simples para a rede.

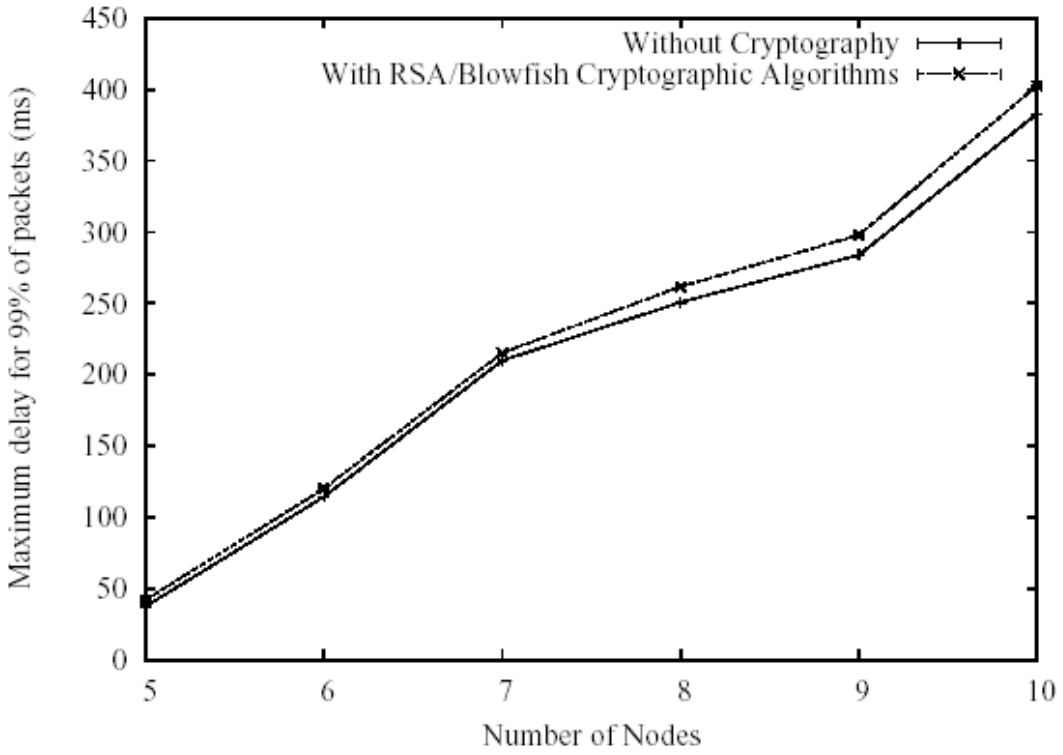


Figura 59 – Atraso no emprego conjunto dos algoritmos RSA e Blowfish

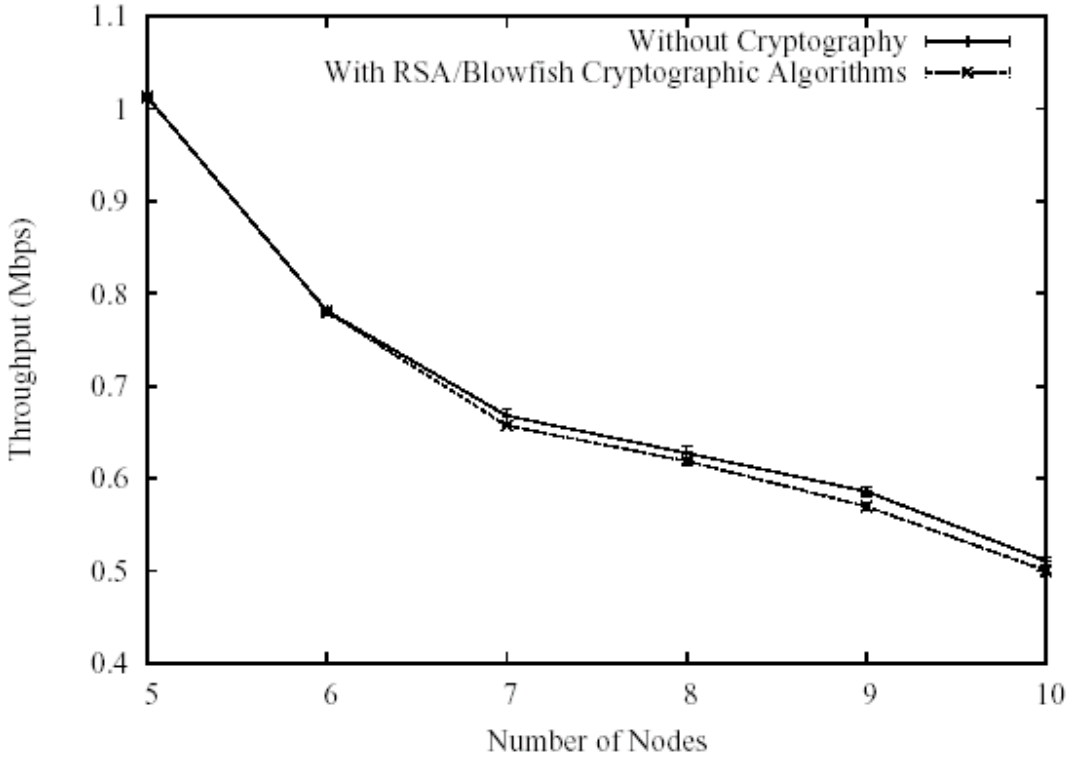


Figura 60 – Desempenho no emprego conjunto dos algoritmos RSA e Blowfish

6.3.2. Resultados de simulação para o segundo cenário

Foi realizada uma simulação para comparar o desempenho dos quatro algoritmos criptográficos desenvolvidos neste trabalho, em um cenário com vários pares de nós (seis a trinta nós), com a finalidade de avaliar o impacto na comunicação de apenas dois nós. O principal objetivo dessa simulação foi medir a capacidade de entrega de pacotes pelos protocolos de comunicação entre apenas dois nós.

As avaliações foram baseadas em simulações de até 30 terminais móveis que formavam uma rede *Ad Hoc*, disposta sobre uma área retangular de 1500m x 1500m, durante um tempo de 900 segundos. Os quatro algoritmos criptográficos foram testados em condições idênticas ao ambiente descrito na simulação do cenário anterior.

As figuras 61 a 65 destacam o desempenho relativo aos algoritmos criptográficos. Todos os algoritmos criptográficos entregam uma grande porcentagem de pacotes, independentemente do número de nós existentes na rede simulada.

Contudo, é visível que, quando existe um número maior de nós, os algoritmos simétricos desenvolvidos (Blowfish e Godzuk) têm um desempenho muito melhor que os algoritmos assimétricos (RSA e ECC).

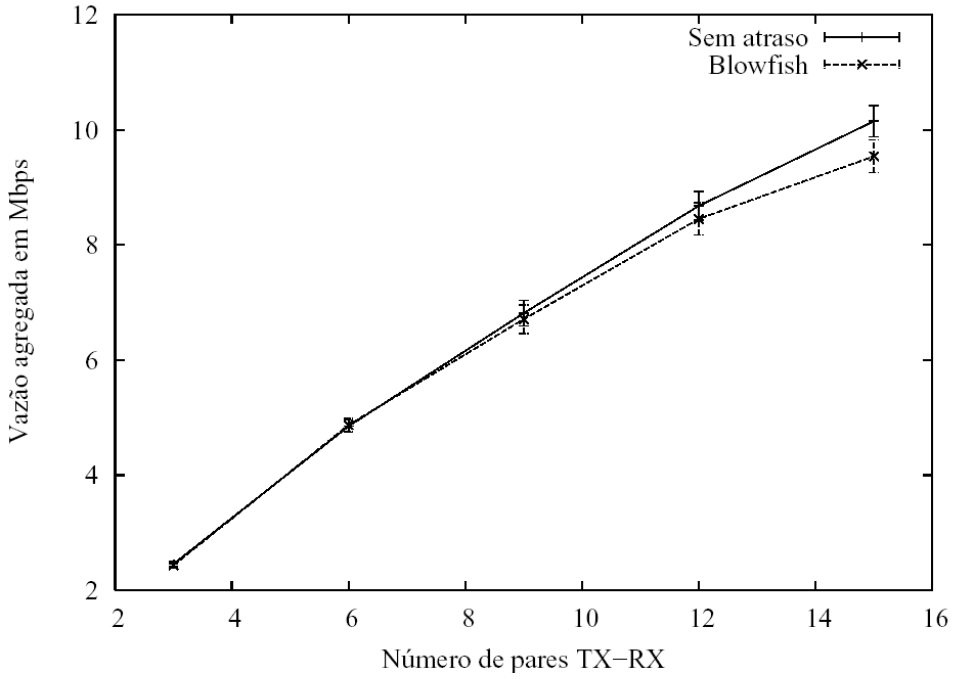


Figura 61 – Vazão da rede com o algoritmo Blowfish

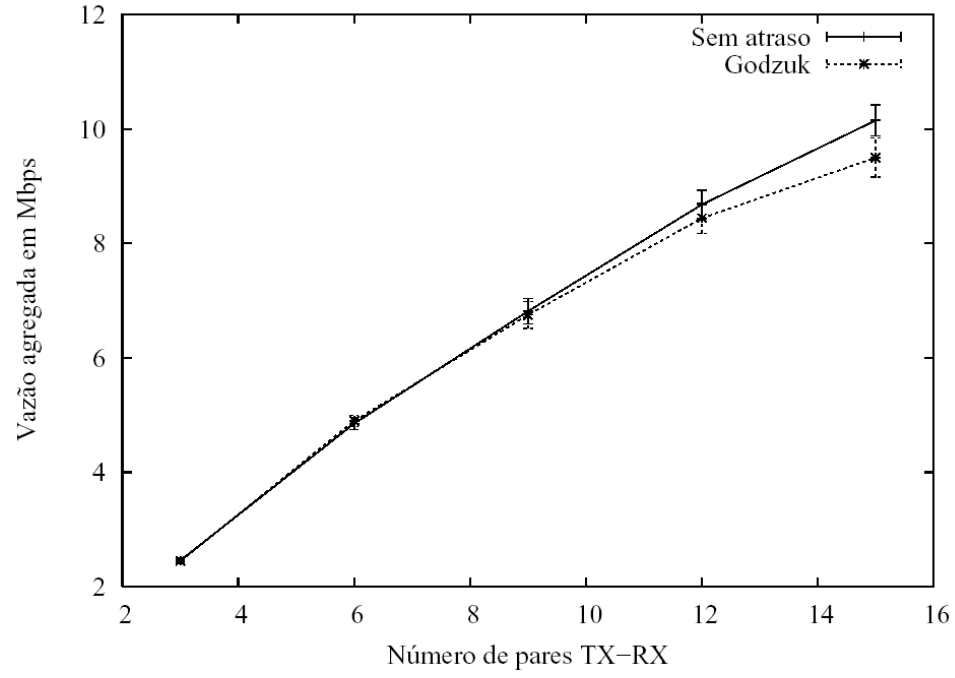


Figura 62 – Vazão da rede com o algoritmo Godzuk

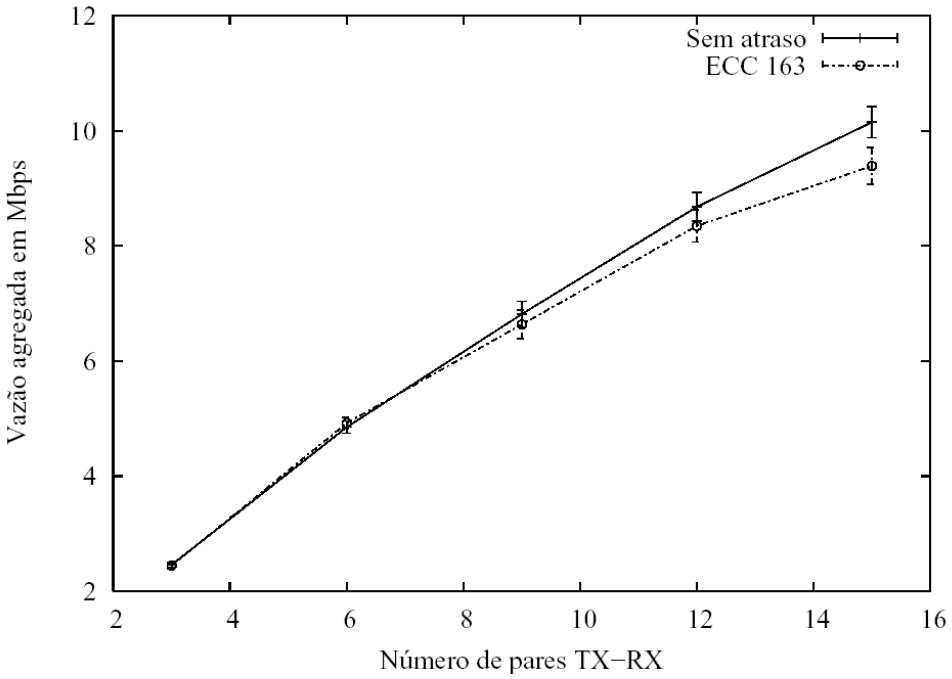


Figura 63 – Vazão da rede com o algoritmo ECC

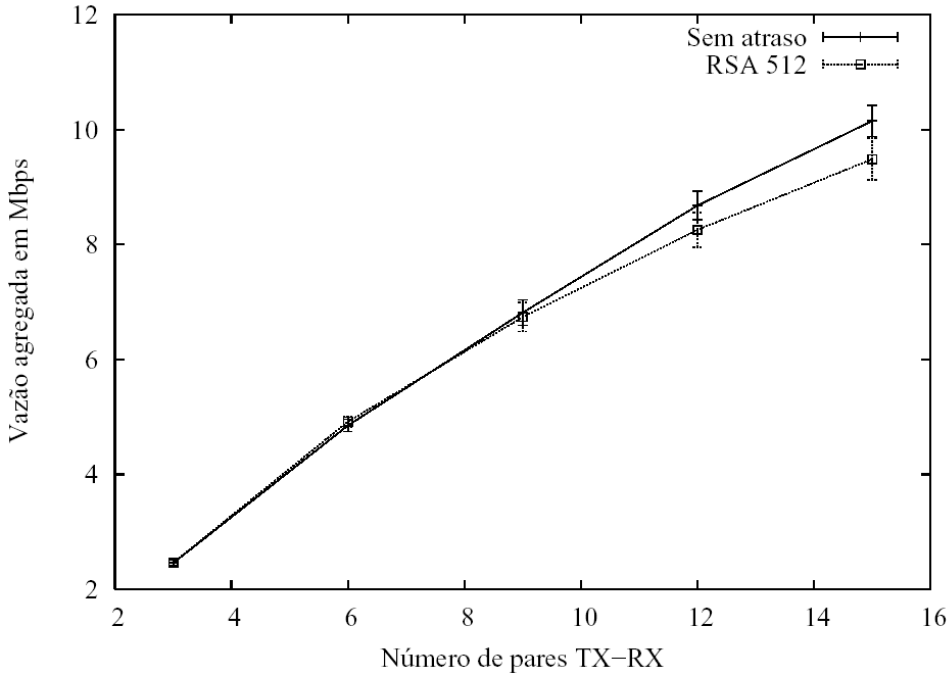


Figura 64 – Vazão da rede com o algoritmo RSA 512

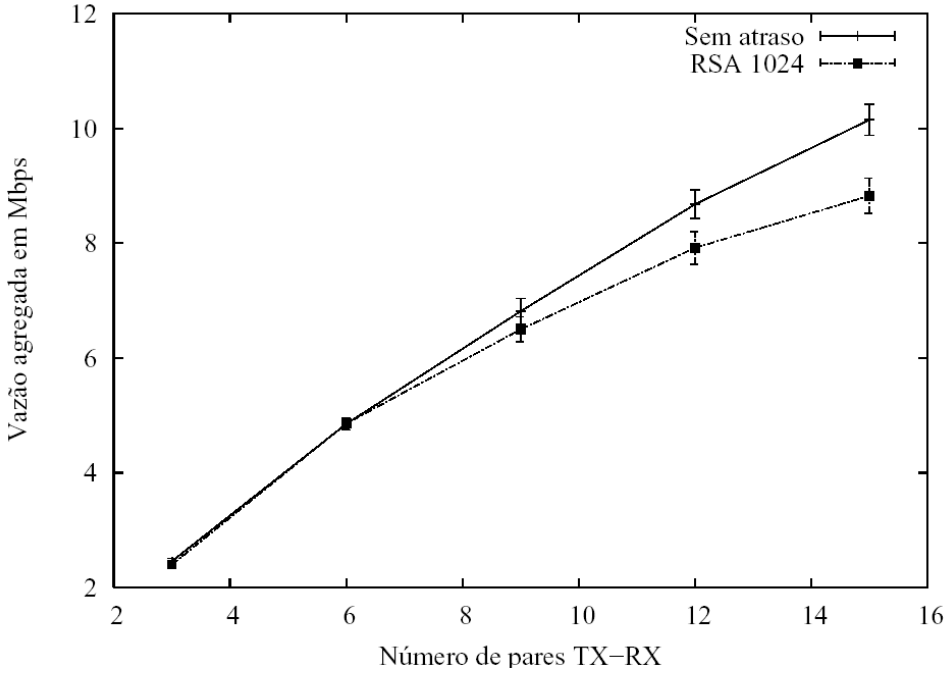


Figura 65 – Vazão da rede com o algoritmo RSA 1024

6.4. Considerações

Este Capítulo apresentou, de forma comparativa, o impacto da implementação dos algoritmos criptográficos RSA, ECC, Blowfish e Godzuk numa rede sem fio. Esses resultados ilustram o pequeno impacto da inserção desses algoritmos quando implementados em *hardware* e comparados com as implementações em *software*.

Pode-se comprovar que os algoritmos criptográficos simétricos (Blowfish e Godzuk) têm melhor desempenho e menor atraso de pacotes, causando um menor impacto na execução da comunicação entre os nós da rede.

No caso dos algoritmos criptográficos assimétricos (RSA e ECC), pode-se verificar que seu desempenho compromete a comunicação da rede, sobretudo quando o número de nós cresce. Particularmente o algoritmo criptográfico RSA com chave de 1024 *bits* teve o pior desempenho, sendo desaconselhável seu uso nos cenários apresentados. O uso dos algoritmos assimétricos nessas redes é aconselhável apenas quando empregado em conjunto com um algoritmo criptográfico simétrico, por combinar o uso das potencialidades que os dois algoritmos possuem e obter as

vantagens que cada sistema oferece, minimizando as suas desvantagens, sobretudo no cenário apresentado.

Capítulo 7

Conclusão

Neste trabalho foi proposta e avaliada a implementação em *hardware* de algoritmos criptográficos robustos, simétricos e assimétricos. Com base nas investigações efetuadas, pôde-se verificar a aplicação desses circuitos em sistemas de segurança, para redes sem fio.

Foi possível verificar conforme já afirmado na literatura, que os sistemas criptográficos mais eficientes utilizam simultaneamente os algoritmos simétrico e assimétrico para estabelecer uma conexão segura. A criptografia de chave pública permite aos usuários publicar suas chaves a qualquer nó, sem a necessidade de um canal seguro, como exigido pelos algoritmos de chave privada. Só o proprietário da chave pode decifrar a mensagem codificada com a sua chave privada, que é matematicamente unida à chave publica. Já os algoritmos simétricos apresentam melhor desempenho quando comparados aos algoritmos assimétricos.

Assim, este trabalho utilizou algoritmos criptográficos simétricos e assimétricos em *hardware* para utilização integrada, os chamados sistemas criptográficos híbridos, nos quais se utiliza a criptografia assimétrica nas chaves e a criptografia simétrica nos textos, aproveitando as vantagens e minimizando as desvantagens dos dois sistemas criptográficos.

Além disso, as chaves públicas permitem assinaturas digitais de um usuário,

garantindo a sua autenticidade. A desvantagem é que esses algoritmos têm desempenho inferior aos de chave privada. Os sistemas que usam uma combinação dos dois algoritmos podem criar um enlace robusto e mais seguro com melhor desempenho, principalmente nas modernas redes sem fio.

Inicialmente, foram implementados em *hardware* quatro algoritmos criptográficos, sendo dois assimétricos (RSA e ECC) e dois simétricos (Blowfish e Godzuk). Posteriormente, aplicou-se uma técnica para avaliação do consumo de potência de forma comparativa entre esses quatro circuitos implementados e necessários em equipamentos móveis. Por fim, desenvolveu-se uma técnica para avaliar o impacto da inserção de *hardware* nos protolocos de comunicação de redes móveis, usando a ferramenta de simulação NS, com base nos atrasos das implementações obtidas por simulações no SYNOPSYS, obtendo-se as métricas para avaliar vazão e atrasos fim-a-fim, percentual de pacotes perdidos com base nesses atrasos, e desempenho de cada circuito desenvolvido.

No projeto das arquiteturas dos algoritmos criptográficos implementados foram aplicados conceitos de paralelismo espacial e paralelismo temporal, com o objetivo de se obter um melhor compromisso entre desempenho, área e consumo.

Comparando os resultados obtidos neste trabalho, para a implementação do algoritmo criptográfico RSA, com outras implementações em *hardware* e *software*, pôde-se verificar que o desempenho do circuito final deste trabalho foi superior e/ou análogo à maioria das implementações atuais encontradas.

A freqüência de operação do integrado RSA final foi de aproximadamente 15 MHz e 6 MHz, para chaves de 512 e 1024 *bits*, respectivamente, o que é muito aceitável para um circuito que calcula o módulo da exponencial de números longos. Outra característica importante para a posterior implementação desse circuito em silício é a área, que necessitou de poucos elementos básicos de lógica (multiplexadores, somadores completos, registradores, memórias) na sua implementação.

Para as duas implementações do *chip* RSA descritas na tese, foram obtidas as taxas de transmissão para cada uma das implementações, na cifragem e na decifragem, mostradas na Tabela 12.

Implementações	Cifrar / Verificar assinatura	Decifrar / Gerar assinatura
Implementação 512 *bits*	260 *kbits*/s	102 *kbits*/s
Implementação 1024 *bits*	38 *kbits*/s	7 *kbits*/s

Tabela 12 – Taxa de transmissão das implementações do algoritmo RSA

As taxas de transmissão obtidas foram computadas levando-se em consideração os valores típicos usados para cifrar e decifrar, que na expressão matemática é o expoente da exponencial (E) da chave. Esses valores têm a largura de 12 *bits* e 30 *bits* para cifrar, e de 20 *bits* e 50 *bits* para decifrar. São valores considerados típicos para chaves de 512 *bits* e 1024 *bits* de tamanho.

O valor obtido para o consumo de energia no processamento de mensagens de 1024 *bytes* do circuito RSA (384,5 mJ) pode ser considerado elevado em relação a outros circuitos que implementam outros algoritmos, tendo em vista o tempo de processamento da mensagem e a potência média consumida pelo circuito, de 1788 mW. Contudo, para aplicações de curto processamento, como cifrar/decifrar chaves de algoritmos criptograficos simétricos, é satisfatório.

Os somadores formam o elo mais fraco na cadeia de processamento desse circuito, sendo os responsáveis finais pela freqüência de operação global do circuito. Nesse contexto, as melhorias de desempenho e redução de área propostas para esse circuito foram baseadas na própria melhoria desses componentes. O projeto de uma implementação assíncrona é outra possibilidade para melhorar ainda mais o desempenho do circuito, uma vez que se trocaria o tempo mais longo de execução (somador) pelo tempo médio de execução, o que acarretaria desvantagem de incremento na área final do circuito e possivelmente um aumento de consumo de energia.

Outro circuito realizado neste trabalho foi o desenvolvimento de um acelerador criptográfico que implementa o algoritmo simétrico Blowfish. Esse algoritmo tem se mostrado muito promissor na atualidade, por ser não patenteado e pela sua robustez. Para a arquitetura resultante foram obtidos os resultados ilustrados na Tabela 13.

Resultados	ASIC	FPGA
Máximo retardo	20 ns	100 ns
Tempo total para uma cifragem	240 ns	1600 ns
Freqüência de operação	50 MHz	10 MHz
Taxa de cifragem	200 *Mbits*/s	40 *Mbits*/s

Tabela 13 – Desempenho do algoritmo Blowfish

Com base nos resultados de desempenho do algoritmo Blowfish, verificou-se que sua implementação compara-se à implementação de [15] e é compatível com a aplicação proposta em redes *sem fio*.

Por ter uma arquitetura em *pipeline*, em que vários estágios do circuito funcionam simultaneamente, o consumo médio de potência do circuito do algoritmo Blowfish para o processamento de mensagens de 1024 *bytes* foi de aproximadamente 1493 mW. Contudo, como o tempo para processar essa mensagem é pequeno, o consumo de energia total foi o menor dentre todas as outras implementações, implicando que, para esse algoritmo criptografico simétrico, o consumo de energia é satisfatório.

Quanto ao algoritmo Godzuk, seu desenvolvimento foi fundamentado na técnica de escalonamento por reversões múltiplas de arestas (SMER). Esta técnica possui alto potencial para a criptografia, devido ao fato de proporcionar um aumento significativo na difusão da informação.

O algoritmo simétrico Godzuk foi desenvolvido para atender às normas de segurança exigidas pelo 3GPP na definição dos requisitos da terceira geração de celulares, que limitava a área máxima do algoritmo implementado em 10.000 portas lógicas equivalentes e pedia um desempenho superior a 2 *Mbits*/s.

Os resultados da implementação desse algoritmo são apresentados na Tabela 14, e mostram que o Godzuk atende aos requisitos de segurança, área e desempenho definidos pelo 3GPP para a terceira geração de celulares.

Resultados	ASIC
Freqüência de operação	78 MHz
Taxa de cifragem	255 *Mbits*/s
Tempo total para cifragem de mensagem de 1024 *bytes*	39 μs

Tabela 14 – Resultados da implementação de Godzuk

Comparando os resultados obtidos nos testes e avaliações, pode-se confirmar que os circuitos desenvolvidos para este trabalho executam as funções dos algoritmos criptográficos RSA, ECC, Blowfish e Godzuk sem uma perda significativa de desempenho das redes *Ad Hoc* e com baixo consumo de energia.

Este trabalho permitiu o desenvolvimento de uma metodologia para integração de *hardware* (de segurança) em protocolos desenvolvidos em *software*. A utilização dessa técnica proporcionou a possibilidade de inserção de sistemas de segurança em redes já existentes, permitindo a avaliação do impacto desses circuitos no seu desempenho. Esses resultados ilustram o pequeno impacto da inserção dos algoritmos RSA, ECC, Blowfish e Godzuk quando implementados em *hardware* e comparados com as respectivas implementações em *software*.

Pôde-se comprovar que os algoritmos criptográficos simétricos (Blowfish e Godzuk) têm melhor desempenho e menor atraso de pacotes, causando um menor impacto na execução da comunicação entre os nós da rede. No caso dos algoritmos criptográficos assimétricos (RSA e ECC), pôde-se verificar que seu desempenho compromete a comunicação da rede, sobretudo quando o número de nós cresce. Particularmente o algoritmo criptográfico RSA com chave de 1024 *bits* teve o pior desempenho, sendo desaconselhável seu uso nos cenários apresentados. O uso dos algoritmos assimétricos nessas redes é aconselhável apenas quando empregado em conjunto com um algoritmo criptográfico simétrico.

Com o intuito de aperfeiçoar ainda mais esta metodologia, trabalhos futuros poderão conter novos estudos de outros algoritmos criptográficos em *hardware* e sua implementação nas redes *sem fio*. Outros algoritmos criptográficos podem ser uma boa possibilidade de comparação, para avaliar melhor a robustez e o desempenho dos circuitos apresentados, corroborando seus melhores resultados.

Outro trabalho que poderá ser realizado é a integração, num mesmo circuito, de um algoritmo criptográfico simétrico e outro assimétrico. Esse novo circuito teria um aumento de desempenho, com a desvantagem de incremento na área final do circuito, quando comparado com os circuitos criptográficos funcionando independentemente.

Capítulo 8

Anexo A

Matemática de apoio para criptografia

Para apresentar as aplicações em curvas elípticas, faz-se necessário um estudo da aritmética modular e da álgebra de curvas elípticas, bem como das operações de curvas elípticas sobre corpos finitos.

8.1. Redução modular

Aplicações de criptografia necessitam de uma aritmética rápida e precisa, impossível de se obter ao se trabalhar com o plano dos reais, que gera problemas com arredondamentos, truncagem de valores e limites. Para resolver isto, a criptografia sempre trabalha com um conjunto de valores inteiros. Assim, em se tratando de criptografia, é imprescindível conhecer a operação de redução modular, definida pela Equação 31 [36].

$$x \bmod n = x - (\lfloor x / n \rfloor . n) \qquad \text{Equação 31}$$

Esta operação resulta em um resíduo ou resto da divisão inteira de dois números inteiros, x por n, sendo $n > 0$. O próprio resíduo também é um inteiro, sempre maior ou igual a 0 e menor ou igual a $(n - 1)$, ou seja, sempre estará no intervalo $0 \leq (x \bmod n) \leq n-1$. Em outras palavras, aplicando-se a operação de redução modular para qualquer x sobre um número n, é possível mapear todos os elementos do conjunto de números inteiros $\{0, 1, ..., n-1\}$, confinando os resultados a este conjunto. Assim, dados quaisquer números inteiros positivos x e n, pode-se dividi-los, obtendo-se um quociente q e um resto r, obedecendo à relação básica dada pela Equação 32 [49].

$$x = qn + r, \text{ sendo } (0 \leq r < n) \text{ e } q = \lfloor x / n \rfloor \qquad \text{Equação 32}$$

O símbolo $\lfloor x / n \rfloor$ é o maior inteiro menor ou igual ao resultado da divisão de x

por n, isto é, x / n.

A operação de redução modular é muito freqüentemente chamada de módulo, apesar deste ser um nome mais adequado para o número n usado na redução [36].

8.1.1. Exemplos de operações de redução modular

Encontrar o resto inteiro da divisão x / n para $x > n$ é bastante simples, como no Exemplo 1 [36]. Para os casos em que $x < n$, o resto da divisão vem a ser o próprio x, como indica o Exemplo 2. Ambos os casos ilustram o resultado para números positivos ($x > 0$).

$$10 \bmod 7 = 10 - (\lfloor\, 10 / 7 \,\rfloor \times 7) = 10 - 1 \times 7 = 3 \qquad \text{Exemplo 1}$$

$$5 \bmod 7 = 5 - (\lfloor\, 5 / 7 \,\rfloor \times 7) = 5 - 0 \times 7 = 5 \qquad \text{Exemplo 2}$$

Para números negativos ($\boldsymbol{x} < 0$), há que se ter mais cuidado na operação, indicada no Exemplo 3, onde o maior inteiro menor ou igual ao quociente da divisão de –10 / 7 vem a ser –2.

$$-10 \bmod 7 = -10 - (\lfloor\, -10 / 7 \,\rfloor \times 7) = 10 - (-2 \times 7) = 4 \qquad \text{Exemplo 3}$$

8.1.2. Propriedades da redução modular

As principais propriedades da operação de redução modular são [36]:

Soma: $(x + y) \bmod n = [(x \bmod n) + (y \bmod n)] \bmod n$

Subtração: $(x - y) \bmod n = [(x \bmod n) - (y \bmod n)] \bmod n$

Produto: $(x \,.\, y) \bmod n = [(x \bmod n)\,(y \bmod n)] \bmod n$

8.1.3. Inversa multiplicativa

Na álgebra de curvas elípticas pode ocorrer a necessidade de se obter o resultado de uma divisão módulo *n*. Neste caso, é utilizada a inversa multiplicativa.

Sejam os inteiros positivos *y* e *n*. O inteiro ξ tal que (*y* **.** ξ) mod *n* = 1 é chamado de inversa multiplicativa de *y* módulo *n*. Esta inversa não existe para qualquer par [*y*, *n*]. Para que ξ exista, *y* e *n* devem ser primos entre si (um não divide o outro, ou um não é fator do outro), isto é, tem-se que ter MDC(*y*, *n*) = 1. Pode-se definir a divisão modular como [36]:

$$(x / y) \bmod n = (x \,.\, \xi) \bmod n \qquad \text{Equação 33}$$

Na Equação 33, ξ é a inversa multiplicativa de *y, isto é,* (*y* **.** ξ) mod *n* = 1. Como *n* é sempre um número primo nas aplicações em criptografia, e como o MDC de um número primo com qualquer número menor que ele mesmo é igual a 1, é certo que *y* e *n* são primos entre si e que sempre haverá uma inversa multiplicativa de *y*. Segue um exemplo:

(1 / 4) mod 23 = 6, pois (1 **.** ξ) mod 23 = (1 **.** 6) mod 23 = 6 mod 23 = 6

Sendo ξ inteiro tal que (4 **.** ξ) mod 23 = 1, tem-se que o MDC(4, 23) = 1 e, pela relação básica da Equação 32, tem-se:

$$4 \,.\, \xi = q \,.\, 23 + 1$$

Testando para valores inteiros de $q > 0$, tem-se ξ = 6 para $q = 1$.

Um algoritmo de inversão binária que fornece a inversa multiplicativa (denotada por a^{-1}) de um número $a \in Z[1, (p-1)]$, onde *p* é um número primo, é apresentado na Figura 66 a seguir [81].

```
Algoritmo:    Inversão Binária
Entrada:  p primo e a ∈ Z[1, (p–1)]
Saída:    a^–1 mod p

u ← a, v ← p, A ← 1, c ← 0
while u ≠ 0 do
    while u is even do
        u ← u / 2
        if A is even then
            A ← A / 2
        else
            A ← (A + p) / 2
        endif
    endwhile
    while v is even do
        v ← v / 2
        if c is even then
            c ← c / 2
        else
            c ← (c + p) / 2
        endif
    endwhile
    if u ≥ v then
        u ← u – v, A ← A – c
    else
        v ← v – u, c ← c – A
    endif
endwhile
return (c mod p)
```

Figura 66 – Inversão binária para inversa multiplicativa

8.2. Álgebra de curvas elípticas

A criptografia de curvas elípticas trabalha com pontos em uma curva, e as curvas elípticas possuem propriedades interessantes. Dado um conjunto de pontos pertencentes a uma curva – um corpo, é possível definir operações específicas e um elemento identidade. Adição e multiplicação de pontos de uma curva são as principais operações feitas nessas curvas [58].

8.2.1. *Adição de pontos*

A operação *soma* de dois pontos pertencentes a uma curva elíptica é definida como sendo um outro ponto na curva [54], como indicado na Equação 34. Não se trata de uma soma como na álgebra convencional. As regras para a sua aplicabilidade são um tanto mais complexas [49].

$$R = P + Q \qquad \text{Equação 34}$$

Graficamente, a regra para adição dos pontos P e Q é a seguinte: primeiramente, desenha-se uma linha reta entre os pontos P e Q, prolongada até um ponto de interseção –R; a seguir, desenha-se uma linha vertical a partir do ponto –R até um outro ponto de interseção R. O ponto R é o resultado da soma dos pontos P e Q, representado na Figura 67 [45].

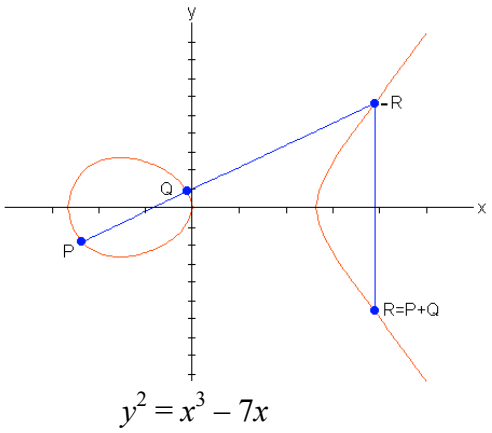


Figura 67 – Soma elíptica (R = P + Q)

8.2.2. *Elemento identidade*

Um elemento identidade nessa álgebra é um ponto *O* (chamado de ponto no infinito), tal que a soma de qualquer outro ponto P da curva a esse ponto resulta no próprio ponto P (equivalente à soma de um inteiro > 0 com 0, na álgebra tradicional). Dessa forma, dada uma curva elíptica E e os pontos P, $O \in E$:

$$R = P + Q \qquad \text{Equação 35}$$

Em termos gráficos, o ponto *O* está localizado em um lugar infinitamente distante, sobre o eixo vertical.

8.2.3. *Soma de um ponto com ele mesmo*

Somar um ponto com ele mesmo equivale a *dobrar* um ponto, como indica a Equação 36.

$$R = 2P = P + P \qquad \text{Equação 36}$$

Esta operação, mostrada na Figura 68, é obtida através de uma reta tangente ao ponto da curva que se deseja *dobrar* (P). A tangente atravessa o ponto –R dessa curva, o qual, rebatido no eixo horizontal, define R como resultado da soma [48].

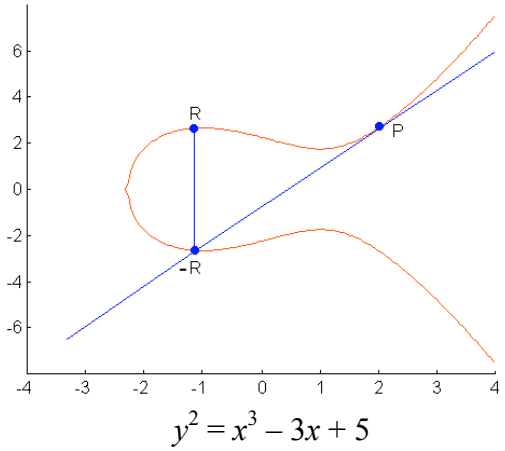
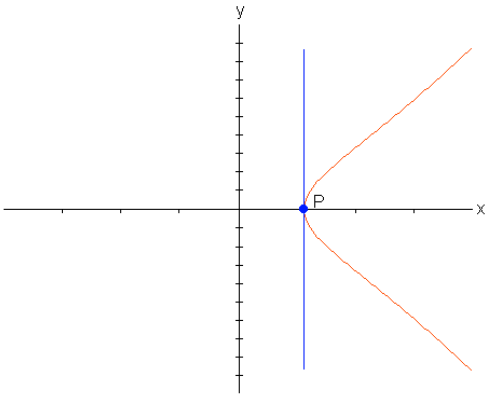


Figura 68 – Soma de um mesmo ponto (R = P + P = 2P)

8.2.4. *"Dobra" de um ponto de ordenada 0*

A *dobra* de um ponto P(x_p, y_p) de ordenada $y_p = 0$ leva ao ponto *O* ou ponto no infinito [58], como representado na Figura 69 [48].

$$P = (x_p, y_p) \text{ onde } y_p = 0 \Rightarrow 2P = O \qquad \text{Equação 37}$$



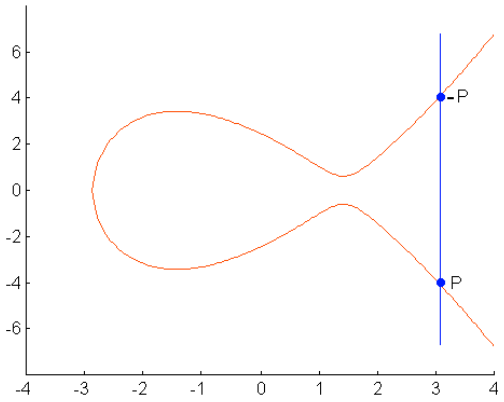
$$y^2 = x^3 + 5x - 7$$

Figura 69 – *Dobro* de um ponto (x_p, y_p), onde y_p = 0

8.2.5. Soma de um ponto com seu oposto

O resultado da soma de um ponto P com seu *negativo* –P é o ponto *O*. A operação é mostrada na Figura 70 [48].

$$P + (-P) = O \text{ onde } P = (x_p, y_p) \text{ e } (-P) = (x_p, -y_p) \qquad \text{Equação 38}$$



$$y^2 = x^3 - 6x + 6$$

Figura 70 – Soma de um ponto com seu oposto (P + (– P) = O)

8.2.6. Multiplicação de pontos

Se existe um ponto P em uma curva elíptica, todos os múltiplos deste ponto também estarão nesta curva [44].

A multiplicação de um ponto em uma curva elíptica por um escalar é feita somando o ponto a ele mesmo tantas vezes quantas forem indicadas pelo fator de multiplicação [54], indicado por *k* na Equação 39, onde P e Q são pontos sobre uma curva elíptica e *k* é um inteiro maior que 2.

$$P + (-P) = O \text{ onde } P = (x_p, y_p) \text{ e } (-P) = (x_p, -y_p) \qquad \text{Equação 39}$$

É possível melhorar a eficiência neste cálculo, conhecendo a ordem do ponto P. O menor inteiro positivo *k* tal que $k \cdot P = O$ é chamado ordem de P [56].

A título de exemplo, a partir da Figura 68, para calcular Q = 15P, basta reaplicar, continuamente, as operações de soma e de soma de mesmo ponto, de forma a obter o valor desejado (também um ponto na curva). Assim é possível diminuir de 15 para apenas seis operações [54].

$$Q = 15P = P + 2(P + 2(P + 2P)) \qquad \text{Equação 40}$$

Há outro método para melhorar a eficiência do cálculo da multiplicação, que é a expansão balanceada proposta por Koblitz [12].

8.3. Curvas elípticas sobre corpos finitos

Curvas elípticas podem ser definidas sobre um conjunto finito de valores inteiros denominado campo ou corpo finito, denotado por F_q ou GF(q). As curvas elípticas sobre corpos finitos primos F_p (curvas em campos de característica p) e as curvas elípticas sobre corpos finitos de característica dois F_{2^m} (curvas em campos de característica 2) constituem duas das opções de F_q muito utilizadas [58].

8.3.1. Curvas elípticas sobre corpos finitos primos

O conjunto F_p é composto por valores de 0 a p–1. Para que os resultados alcançados nas operações de soma e multiplicação estejam dentro de um corpo finito primo F_p, todas as operações devem ser finalizadas calculando-se o resto da divisão por p [58].

Assim, uma curva elíptica E: $y^2 = x^3 + ax + b$ sobre o campo F_p , onde a, b pertencem a F_p, é definida por:

$$(y^2) \bmod p = (x^3 + ax + b) \bmod p \qquad \text{Equação 41}$$

Todos os pontos (x, y) que satisfazem a Equação 41 estão incluídos na curva elíptica, sendo que x, y pertencem a F_p. A curva também inclui o ponto O.

As equações a seguir definem as operações de soma de pontos numa curva elíptica sobre F_p :

$$P_3(x_3, y_3) = P_1(x_1, y_1) + P_2(x_2, y_2), \text{ onde } P_3 \neq O \qquad \text{Equação 42}$$

$$\lambda = [(y_2 - y_1) / (x_2 - x_1)] \bmod p, \text{ se } x_1 \neq x_2 \qquad \text{Equação 43}$$

$$\lambda = [(3x^2 + a) / 2y_1] \bmod p, \text{ se } x_1 = x_2 \text{ , } y_2 \neq 0 \qquad \text{Equação 44}$$

$$x_3 = (\lambda^2 - x_1 - x_2) \bmod p \qquad \text{Equação 45}$$

$$y_3 = (\lambda\,(x_1 - x_3) - y_1) \bmod p \qquad \text{Equação 46}$$

Juntamente com as regras de adição definidas pelas equações acima, o conjunto de pontos da curva elíptica E sobre F_p forma um grupo, com o ponto O servindo como sua identidade. Este é o grupo utilizado na construção de sistemas de criptografia baseados em curvas elípticas [54].

8.3.2. Exemplo de curvas elípticas sobre corpos finitos primos

A Criptografia em Curvas Elípticas (ECC) deve empregar uma forma de curva elíptica que está definida sobre um corpo finito. O caso mais empregado ocorre ao se aplicar um grupo elíptico módulo p, sendo p um número primo [49].

Como exemplo, sejam $p = 23$ e a curva elíptica E: $y^2 = x^3 + x + 1$, com $a = 1$ e

$b = 1$, definida sobre F_{23}. Como definido pela Equação 8, os dois números a e b devem ser escolhidos entre inteiros não negativos menores que p, de forma que satisfaçam a desigualdade agora representada pela Equação 47.

$$(4a^3 + 27b^2) \bmod p \neq 0 \qquad \text{Equação 47}$$

Com esta restrição, garante-se que a curva realmente forma um grupo elíptico mod p (não possui fatores repetidos), cujos elementos (pontos pertencentes a esta curva) são pares de inteiros não negativos e menores que p, que satisfazem a equação de definição (Equação 41), junto com o ponto de infinitude O.

Para o grupo elíptico, interessam os valores não negativos inteiros, ou seja, pontos entre o intervalo bidimensional (0, 0) até (p, p). Quanto maior o número primo p, maiores serão as chances de se constituir um grupo.

Os pontos que constituem um grupo elíptico podem ser obtidos da seguinte forma [49]:

a) Calcula-se $(x^3 + ax + b) \bmod p$ para cada x, sendo $0 \leq x < p$.

b) Para cada resultado obtido, determina-se a existência de uma raiz quadrada módulo p. Se não houver, não há pontos na curva com esse valor de x. Se houver, poderão ser dois os valores possíveis em y, salvo o caso em que $y = 0$.

É possível obter os pontos da curva aplicando-se o algoritmo acima, limitando-se aos pontos (0, 0) e (23, 23). Todos os pontos encontrados compõem o chamado corpo finito da curva elíptica para $p = 23$.

Pode-se verificar, como exemplo, que o ponto P(5, 4) satisfaz a equação de definição da curva, pois:

$$(4^2) \bmod 23 = (5^3 + 5 + 1) \bmod 23 \qquad \text{Exemplo 4}$$

$$(16) \bmod 23 = (125 + 5 + 1) \bmod 23$$

$$16 = 16$$

Assim, o ponto P(5, 4) pertence à curva dada.

8.3.3. Curvas elípticas sobre corpos finitos de característica dois (F_2^m)

O campo finito F_{2m} é um espaço vetorial de dimensão m sobre o campo F_2 o qual consiste dos dois elementos 0 e 1. F_{2m} é freqüentemente referenciado como campo finito de característica 2 ou campo finito binário. Como ele é um espaço vetorial, todo elemento a de F_{2m} pode ser representado como uma cadeia de *bits* (a_0 a_1 ... a_{m-1}) [51]:

$$a = a_0 \cdot \beta_0 + a_1 \cdot \beta_1 + ... + a_{m-1} \cdot \beta_{m-1}\ ,\ \text{onde } a_i \qquad \text{Equação 48}$$

O conjunto $\{\beta_0, \beta_1, ..., \beta_{m-1}\}$ é chamado base de F_{2m} sobre F_2. Existem muitas bases diferentes, e algumas delas possuem implementações mais eficientes do que outras.

Um polinômio irredutível de grau m sobre F_2 pode ser escrito na forma da Equação 49, onde $f_i \in \{0, 1\}$.

$$f(x) = x^m + f_{m-1}x^{m-1} + ... + f_2x^2 + f_1x + f_0, \text{ onde } f_i \in \{0, 1\} \qquad \text{Equação 49}$$

Irredutível significa que ele não pode ser fatorado como um produto de dois polinômios sobre F_2, cada um de grau menor do que m. Estes chamados polinômios de redução $f(x)$ definem a representação de base polinomial de F_{2m}, ou seja:

$$F_{2m} \cong \{a_{m-1}x^{m-1} + ... + a_1x + a_0 \mid a_i \in \{0, 1\}\ \} \qquad \text{Equação 50}$$

$$\cong \{\ (a_{m-1} ... a_1\ a_0) \mid a_i \in \{0, 1\}\ \}$$

Assim, os elementos de F_{2m} podem ser representados pelo conjunto de todas as cadeias binárias de comprimento m. O elemento identidade multiplicativo é representado pela cadeia de *bits* (0 0 ... 0 1) e o elemento identidade aditivo é representado por todos os *bits* da cadeia iguais a 0.

A adição é executada como XOR *bit* a *bit* dos coeficientes do vetor [82], i.e., dados dois elementos de F_{2m}, $a = (a_{m-1} ... a_1\ a_0)$ e $b = (b_{m-1} ... b_1\ b_0)$, então $a + b = c = (c_{m-1} ... c_1\ c_0)$, onde $c_i = (a_i + b_i) \bmod 2$.

Se $a = (a_{m-1} ... a_1\ a_0)$ e $b = (b_{m-1} ... b_1\ b_0)$ são elementos de F_{2m}, a multiplicação é executada da seguinte forma: $a \cdot b = r = (r_{m-1} ... r_1\ r_0)$, onde o polinômio $r(x) = r_{m-1}x^{m-1} + ... + r_1x + r_0$ é o resto da divisão do polinômio $(a_{m-1}x^{m-1} + ... + a_1x + a_0) \cdot (b_{m-1}x^{m-1} + ... + b_1x + b_0)$ pelo polinômio reduzido $f(x)$.

O conjunto F_{2m} possui dois tipos principais de representação: representação polinomial e representação de base ótima [58]. Segundo [50], para implementações de *hardware*, a representação de base ótima é a melhor opção, ao passo que, em *software*, o uso da representação polinomial é mais indicado. Uma terceira representação, chamada representação de subcampos, também pode ser conferida em [12].

A conveniência do uso de um conjunto de cadeias de *m bits* está no fato de nos aproximarmos mais da representação *natural* de palavras de dados de *m bits* em memória [48]. Como resultado do campo F_{2m} ser de característica 2, a equação que representa uma curva elíptica em F_{2m} é ligeiramente diferente das equações $y^2 = x^3 + ax + b$ e $(y^2) \bmod p = (x^3 + ax + b) \bmod p$, apresentadas anteriormente. A Equação 51, que representa uma curva elíptica em F_{2m}, é aplicável tanto nas representações polinomiais de F_{2m} quanto nas de base ótima [50][83], possuindo como única restrição: $a, b \in F_{2m}$ e $b \neq 0$ [48][83].

$$y^2 + xy = x^3 + ax^2 + b \text{ (em } F_{2m}) \qquad \text{Equação 51}$$

Segue o exemplo da representação do conjunto F_{24} (com $2^4 = 16$ elementos).

$F_{24} = \{(0001), (0010), (0100), (1000), (0011), (0110), (1100), (1011),$

$= \{(1), (x), (x^2), (x^3), (x + 1), (x^2 + x), (x^3 + x^2), (x^3 + x + 1), (x^2 + 1), (x^3 + x), ...\}$

Elemento gerador:

Define-se um elemento gerador g como um dos elementos de F_{2m}, a partir do qual pode-se gerar todo o conjunto, bastando calcular as potências desse elemento [83]. De outra forma, pode-se dizer que, dado um número primo p, é possível chegar a um número gerador g com uma propriedade muito interessante. Cada número entre 1 e p–1 pode ser escrito como uma potência de g quando calculado mod p. Por exemplo, dado $p = 5$, o gerador é $g = 2$, pois [84]:

$g^0 = 2^0 = 1 \bmod 5 = 1$

$g^1 = 2^1 = 2 \bmod 5 = 2$

$g^3 = 2^3 = 8 \bmod 5 = 3$

$g^2 = 2^2 = 4 \bmod 5 = 4$

Dessa forma, para o exemplo F_{24} utilizou-se como gerador $g = (0010)$ (em forma

de polinômio: $\{x\}$) e a função irredutível $f(x) = x^4 + x + 1$. Os demais elementos deste conjunto são resultados da potenciação desse valor, quando, então, se obtém:

$F_{24} = \{g^0, g^1, g^2, g^3, g^4, g^5, g^6, g^7, g^8, g^9, g^{10}, g^{11}, g^{12}, g^{13}, g^{14}, g^{15}\}$, onde:

$g^0 = \{1\} = \{0x^3 + 0x^2 + 0x + 1\} = (0001)$

$g^1 = \{x\} = \{0x^3 + 0x^2 + 1x + 0\} = (0010)$

$g^2 = \{x^2\} = \{0x^3 + 1x^2 + 0x + 0\} = (0100)$

$g^3 = \{x^3\} = \{1x^3 + 0x^2 + 0x + 0\} = (1000)$

Obs.: Para $g^i \bmod f(x)$, $i \geq 4$, o resultado em F_{24} é obtido com a operação abaixo:

$$\begin{array}{rl} x^4 & \left| \underline{x^4 + x + 1} \right. \\ \underline{-x^4 - x - 1} & 1 \\ -x - 1 & \end{array}$$

$g^4 = \{x^4\} \bmod f(x) = \{-x - 1\} = \{0x^3 + 0x^2 - 1x - 1\} = (0011)$

$g^5 = \{x^5\} \bmod f(x) = \{-x^2 - x\} = \{0x^3 - 1x^2 - 1x + 0\} = (0110)$

$g^6 = \{x^6\} \bmod f(x) = \{-x^3 - x^2\} = \{-1x^3 - 1x^2 + 0x + 0\} = (1100)$

$g^7 = \{x^7\} \bmod f(x) = \{-x^3 + x + 1\} = \{-1x^3 + 0x^2 + 1x + 1\} = (1011)$

$g^8 = \{x^8\} \bmod f(x) = \{x^2 + 2x + 1\} = \{0x^3 + 1x^2 + 2x + 1\} = (0101)$

$g^9 = \{x^9\} \bmod f(x) = \{x^3 + 2x^2 + x\} = \{1x^3 + 2x^2 + 1x + 0\} = (1010)$

$g^{10} = \{x^{10}\} \bmod f(x) = \{2x^3 + x^2 - x - 1\} = \{2x^3 + 1x^2 - 1x - 1\} = (0111)$

$g^{11} = \{x^{11}\} \bmod f(x) = \{x^3 - x^2 - 3x - 2\} = \{1x^3 - 1x^2 - 3x - 2\} = (1110)$

$g^{12} = \{x^{12}\} \bmod f(x) = \{-x^3 - 3x^2 - 3x - 1\} = \{-1x^3 - 3x^2 - 3x - 1\} = (1111)$

$g^{13} = \{x^{13}\} \bmod f(x) = \{-3x^3 - 3x^2 + 1\} = \{-3x^3 - 3x^2 + 1\} = (1101)$

$g^{14} = \{x^{14}\} \bmod f(x) = \{-3x^3 + 4x + 3\} = \{-3x^3 + 0x^2 + 4x + 3\} = (1001)$

$g^{15} = \{x^{15}\} \bmod f(x) = \{4x^2 + 6x + 3\} = \{0x^3 + 4x^2 + 6x + 3\} = (0001)$

Em aplicações reais de curvas elípticas sobre F_{2m}, o valor de m deve ser tal que permita a geração de uma tabela (similar à apresentada acima) grande o suficiente, de forma a tornar o sistema imune a quebras [83]. Hoje em dia o uso de m igual a 160 tem

se mostrado uma boa opção.

As equações a seguir definem as operações de soma de pontos numa curva elíptica sobre F_{2m}.

$P_3(x_3, y_3) = P_1(x_1, y_1) + P_2(x_2, y_2)$, onde $P_3 \neq O$ Equação 52

$\lambda = [(y_2 + y_1) / (x_2 + x_1)]$, se $x_1 \neq x_2$ (em F_{2m}) Equação 53

$\lambda = [(x_1^2 + y_1) / x_1]$, se $x_1 = x_2 \neq 0$ (em F_{2m}) Equação 54

$x_3 = \lambda^2 + \lambda + a + x_1 + x_2$ (em F_{2m}) Equação 55

$y_3 = \lambda (x_1 + x_3) + x_3 + y_1$ (em F_{2m}) Equação 56

8.3.4. Exemplo de curvas elípticas sobre corpos finitos de característica dois (F_2^n)

Considerando uma curva elíptica $E(F_24)$ com a equação $y^2 + xy = x^3 + g^4x^2 + 1$ ($a = g^4$ e $b = g^0 = 1$), pode-se observar que o ponto (g^5, g^3) satisfaz a equação, desde que:

$$y^2 + xy = x^3 + g^4x^2 + 1$$

$$(g^3)^2 + (g^5)(g^3) = (g^5)^3 + g^4(g^5)^2 + 1$$

$$g^6 + g^8 = g^{15} + g^{14} + 1$$

$$(1100) + (0101) = (0001) + (1001) + (0001)$$

$$(1001) = (1001) \Rightarrow (g^5 + g^3) \in E(F_2^4)$$

$$(1, g^{13})\ (g^3, g^{13})\ (g^5, g^{11})\ (g^6, g^{14})\ (g^9, g^{13})\ (g^{10}, g^8)\ (g^{12}, g^{12})$$

$$(1, g^6)\ (g^3, g^8)\ \mathbf{(g^5, g^3)}\ (g^6, g^8)\ (g^9, g^{10})\ (g^{10}, g)\ (g^{12}, 0)\ (0, 1)$$

Esses pontos, junto com o ponto *O*, compõem a curva elíptica dada.

Capítulo 9

Referências

[1] SINGH, Simon, *The Code Book The Evolution of Secrecy From Mary Queen of Scots to Quantum Cryptography*. New York: Doubleday, 1999. ISBN: 0-385-49531-5

[2] BAMFORD, James, *Body of Secrets: Anatomy of the Ultra-Secret National Security Agency from the Cold War through the Dawn of a New Century*. New York: Doubleday, 2001

[3] FORD, W., BAUM, M. S., *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption*. 2nd ed. Englewood Cliffs, New Jersey: Prentice Hall, 2001.

[4] VANSTONE, Scott A. *Next generation security for wireless: elliptic curve cryptography*. Disponível em <http://www.compseconline.com/hottopics/ hottopic20_8/Next.pdf >. Acesso em: 21 jul. 2003.

[5] TOH, C. K., *Ad Hoc Mobile Wireless Network – Protocols and Systems*. In 1st Edition, Prentice-Hall 2002.

[6] YOSHIDA, Elias, *Informação, Comunicação e a Sociedade do Conhecimento*. <http://www.ime.usp.br/~is/ddt/mac339/projetos/2001/demais/elias>. Acesso em jun. 2003.

[7] BATINA, L., BERNA, S., PRENEEL, B., VANDEWALLE, J., *Hardware architectures for public key cryptography*. Integration, the VLSI journal 34, 1–64, Elsevier, 2003.

[8] KÄRPIJOKI, V., *Security in ad hoc networks*. *In* Seminar on Network Security, Sjokulla, Finland 2001.

[9] MCLOONE, M., MCCANNY, J. V. *Efficient Single-Chip Implementation of SHA-384 and SHA-512*. *In* IEEE International Conference on Field-Programmable Technology (FPT), Hong Kong, July 2002.

[10] SCHINDLER, V., *High Speed RSA Hardware Based on Low-Power Pipelined Logic*. Institut für Angewandte Informationsverarbeitung und Kommunikationstechnologie Technische Universität Graz, Áustria, Dez 1996.

[11] WANG, P., WEI-CHANG, T., SHUNG, C., *New VLSI Architectures of RSA Public-Key Cryptosystem*. Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, ROC China, 1998.

[12] KOBLITZ, Neal, MENEZES, Alfred, VANSTONE, Scott. *The State of Elliptic Curve Cryptography*. Disponível em <http://modular.fas.harvard.edu/edu/Fall2001/124/misc/koblitz_ecc.pdf >. Acesso em set 2005.

[13] MOHAMMED, Elsayed, EMARAH, A. E., EL-SHENNAWY, Kh. *Elliptic Curve Cryptosystems on Smart Cards*. 2001 IEEE 35th International Carnahan Conference on, out 2001, p 213-222.

[14] FLEURY, André, *Criptografia: uma questão de segurança nacional*. Disponível em <http://www.securenet.com.br/artigo.php?artigo=101>. Acesso em: 19 abr. 2005.

[15] LIN, M. C.-J., YOUN-L. LIN *A VLSI implementation of the blowfish encryption/decryption algorithm*. Proceedings of the 2000 conference on Asia South Pacific 2000.

[16] OMURA, J. K., *Novel Applications of Criptography in Digital Communications*. IEEE Communications Magazine, 1990, pp 21-29.

[17] SCHNEIER, Bruce, *Applied Cryptography*. Second Edition, John Wiley & Sons Inc., 1996.

[18] MENEZES, Alfred J., P. C. V. O., VANSTONE, Scott A., *Handbook of Applied Cryptography*. CRC Press LLC, 1997.

[19] WOODBURY, A. D., *Efficient Algorithms for Elliptic Curve Cryptosystems on Embedded Systems*, Worcester Polytechnic Institute, Master Thesis, Setembro 2002.

[20] BETH, Th., FRISCH, M., SIMMONS, G. J., *Public-Key Cryptography: State of the Art and Future Direction*. Lecture Notes in Computer Science, vol. 578, Springer-Verlag, july 1991.

[21] XINJUN Du, Ying Wang, Jianhua Ge, Yumin Wang, *A Group Key Establishment Scheme for Ad Hoc Networks*. 17th International Conference on Advanced Information Networking and Applications (AINA'03) p. 518, 2003.

[22] PHILLIPS, B., J., BURGESS, N., *An Overview Of Public Key Cryptography*. Digital Arithmetic Group, The University Of Adelaide, Australia, 1995.

[23] National Bureau of Standards, Data Encryption Standard, U.S. Departament of Commerce, Federal Information Processing Standards Publication FIPS 46-2, December 1993.

[24] XUEJIA, Lai, MASSEY J. L., *A Proposal for a New Block Encryption Standard*. Advances in Cryptology – EUROCRYPT'90, Proceedings, LNCS 473, pp. 389-404, Springer-Verlag, Berlin, 1991.

[25] SHIMIZU, A., MIYAGUCHI, S., *Fast Data Encipherment Algorithm – FEAL*, EUROCRYPT 87, Amsterdam april 1987.

[26] SCHNEIER, Bruce, *Description of a New Variable-Length Key, 64 bit Block Cipher (Blowfish)*. Cambridge Security Workshop Proceedings, Spring-Verlag, pages 191-204, Dec. 1993.

[27] SALOMÃO, S. L. C., *Architectures in Hardware to Cryptographics Accelerators*, Doctor Thesis, COPPE/UFRJ, Rio de Janeiro, Brazil, 2000.

[28] RIVEST, R. L., SHAMIR, A., ADLEMAN, L. M., *A method for obtaing Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, Vol. 21, No. 2, Feb. 1978, pp.120 – 126.

[29] MENEZES, A., *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.

[30] ANTON, E. R., DUARTE, O. C. M. B., *Group Key Establishment in Wireless Ad Hoc Networks*. Workshop em Qualidade de Serviço e Mobilidade – WQoSM 2002, Angra dos Reis, RJ, Brazil, November 2002.

[31] HUBAUX, J., BUTTYAN, L., CAPKUN, S., *The quest for security in mobile ad hoc networks*. In: *ACM Symposium on Mobile Ad Hoc Networking and Computing - MobiHOC*, 2001.

[32] VIEIRA, A. C. C., PEDROZA, A. C. P., MESQUITA FILHO, A. C , *Security Application for Ad Hoc Network*, published in 3º Workshop de segurança em redes de computadores – Wseg -SBRC2003, Natal, Brazil, 2003.

[33] DIFFIE, W., HELLMAN, M. E., *New Directions in Cryptography*, IEEE Transactions on. Information. Theory, vol IT-22, 1976, pp 644-654.

[34] BAUER, F. L., *Decrypted Secrets: Methods and Maxims of Cryptology*. Berlin Heidelberg: Springer-Verlag, 1997.

[35] SALOMÃO, S. L. C., et al., *Hipcrypto: A high-performance VLSI cryptographic chip*. Published in the proceedings of the 11th Annual IEEE International Asic Conference (ASIC98), September 1998.

[36] CARVALHO, Daniel Balparda. *Segurança de dados com criptografia – Métodos e Algoritmos*. 2. ed. Rio de Janeiro: Book Express, 2001.

[37] GRANT, Gail L., *Understanding Digital Signatures: Establishing Trust over the Internet and Other Networks*. New York: Computing McGraw-Hill, 1997. 304p.

[38] ANTON, E. R., *Análise de Desempenho de Protocolos para Estabelecimento de Chave de Grupo em Redes Ad Hoc*. COPPE/UFRJ Tese de Mestrado, Rio de Janeiro, Brasil, 2003.

[39] COUTINHO, S. C., *Números Inteiros e Criptografia RSA*. 1 ed. Instituto de Matemática Pura e Aplicada, IMPA e Sociedade Brasileira de Matemática, SBM, Rio de Janeiro, 1997.

[40] RSA Data Security, Inc., *Frequently Asked Questions about Today's Cryptography*. 2003. <http://www.rsa.com>.

[41] VIEIRA, A. C. C., *Algoritmo Criptográfico RSA Adaptado para implementação em Hardware*. Publicado na Revista da Associação Brasileira de Engenharia Militar – ABEM, pp 52 – 62, Número 96, Ano LXVII, dezembro de 2004, Brasil.

[42] KORNERUP, P. *A systolic, linear-array multiplier for a class of right-shift algorithms*, IEEE Trans. Comput., vol. 43, pp. 892–898, Aug. 1994.

[43] CHEN, P. S., S. A. HWANG, C. W. WU, *A systolic RSA public key cryptosystem*, *in* Proc. IEEE Int. Symp. Circuits and Systems (ISCAS), May 1996, pp. 408–411.

[44] ANGELFRIET, Arnoud. *The Diffie-Hellman system*. 01 jan. 2004. Disponível em <http://www.iusmentis.com/technology/encryption/elliptic-curves>. Acesso em: 27/10/2004.

[45] *Elliptic Curve Cryptography*. Disponível em <http://world.std.com/~dpj/elliptic.html>. Acesso em: 25 nov. 2003.

[46] STALLINGS, William. *Cryptography and Network Security – Principles and Practice*. 3rd Edition, Upper Saddle River, NJ: Prentice Hall, 2003

[47] BOTES, J. J., PENZHORN, W. T., *Public-Key Cryptosystems Based on Elliptic Curves. Communications and Signal Processing*. 1993. Proceedings of the 1993 IEEE South African Symposium on 6 Aug 1993

[48] BELINGUERES, Gabriel. *Introducción A Los Criptosistemas de Curva Elíptica*. Disponível em <http://www.toptutoriales.com/matematicas/criptografia/criptografia7.htm>. Acesso em: 21 jul. 2004.

[49] RIBEIRO, Vinicius Gadis. *Um estudo sobre a Matemática empregada em Criptografia*. 84 p. – (TI 983). Porto Alegre: PPGC da UFRGS, 2001.

[50] BARWOOD, George. *Elliptic Curve Cryptography FAQ v1.12*. 22 dez. 1997. Disponível em <http://www.cryptoman.com/elliptic.htm>. Acesso em: 19 mai. 2004.

[51] RIEDEL, Ingo. *Security in Ad-hoc Networks: Protocols and Elliptic Curve Cryptograflhy on an Embedded Platform*. 2003.

[52] *Remarks on the Security of the Elliptic Curve Cryptosystem*. Certicom *Whitepaper* – set. 1997. Disponível em <http://www.certicom.com>.

[53] CHOUINARD, Jean-Yves. *Notes on Elliptic Curve Cryptography – Design of Secure Computer Systems*. Disponível em <http://www.site.uottawa.ca>. 24 set. 2002.

[54] WOLSKI, Eduardo. *Sistemas Criptográficos Baseados em Curvas Elípticas*. Departamento de Engenharia Elétrica, Universidade de Brasília – UnB. Disponível em <http://www.redes.unb.br/security/criptografia/curvaselipticas/ curvas.html>. Acesso em: dez. 2003.

[55] *Current Public-Key Cryptographic Systems*. *A Certicom Whitepaper* – abr. 1997. Disponível em <http://www.certicom.com>.

[56] BARRETO, Paulo S. L. M. *Curvas Elípticas e Criptografia: Conceitos e Algoritmos*. 01 jun. 1999. Universidade de São Paulo – USP, 1999. Disponível em <http://planeta.terra.com.br/informatica/ paulobarreto>. Acesso em: 19 mai. 2004.

[57] BENITS JÚNIOR, Waldyr Dias. *Sistemas Criptográficos baseados em identidades pessoais*. 2003, 213 f. Tese (Mestrado em Ciência da Computação) – Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 2003. Disponível em < http://www.ime.usp.br/~benits/ dissertacao.zip>. Acesso em: 22 mar. 2004.

[58] BARBOSA, Julio Cesar. *Criptografia de chave pública baseada em curvas elípticas*. Rio de Janeiro: 2003. 28f. Monografia de final de curso – COS 762 (Mestrado em Redes) – COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2003.

[59] *Elliptic Curve Cryptosytems*. Disponível em <http://www.rsasecurity.com/ rsalabs/technotes/elliptic_curve.html>. Acesso em: 22 mar. 2004.

[60] VIEIRA, A. C. C., PINHO, A. C., SALOMÃO, S. L. C., *Godzuk Cryptographic Algorithm for 3rd Generation Mobile Systems*. Published in the proceedings of the IEEE International Telecommunications Symposium -ITS2002, Sep 2002.

[61] COPPERSMITH, D., FRANKLIN, M., PATARIN, J., *Low exponent RSA with related messages*. Proc. Advances in Cryptology, vol 1070, New York , july 1996, pp 216-231.

[62] ÖZTÜRK, E., SUNAR, B., SAVAS, S., *Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic*. IEEE, March, 2004.

[63] VAUDENAY, Serge, *On the Weak Keys of Blowfish. In Fast Software Encryption 3*, Lecture Notes in Computer Science, Vol 1039, Springer-Verlag, pp 27-32, 1996.

[64] VIEIRA, A. C. C., ALCÂNTARA, J. M. S., GÁLVEZ-DURAND, F. ALVES, V. C., *SCOB, A Soft-core for the Blowfish cryptographic algorithm*. Published in the proceedings of the XII Brazilian Symposium on Integrated Circuits, Oct 1999.

[65] IEEE standard 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. August 1999 <http://pdos.csail.mit.edu/decouto/papers/802.11a.pdf> Acesso em: mar 2004.

[66] ANASTASI, Giuseppe, BORGIA, Eleonora, CONTI*, Marco et al*, *IEEE 802.11 Ad Hoc Networks: Performance Measurements*. 23rd International Conference on Distributed Computing Systems Workshops Providence, Rhode Island, USA, 2003.

[67] MATSUI, M, *New Block Encryption Algorithm MISTY*. Fast Software Encryption – 4th International Workshop (FSE'97), LNCS 1267, Springer Verlag, 1997, pp.54-68.

[68] FRANÇA, F. M. G., *Scheduling weightless systems with self-timed Boolean networks*, published in the Workshop on Weightless Neural Network, April 1996.

[69] HANKERSON, D., J. L. H. A. A. M., *Software implementation of elliptic curve cryptography over binary field*. *In* Lecture notes in computer science, pp. 1-24, 2001.

[70] SCHNEIER, Bruce, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 2nd Edition, New York: John Wiley & Sons, 1995. 784p.

[71] MUDGE, Trevor, *Power: A First Class Design Constraint for Future Architectures*. High Performance Computer Conference, Bangalore, India, Dec 2000.

[72] GOUVEIA, L. C. P., Uma estratégia para redução de consumo de potência em processadores para terminais móveis, dissertação mestrado, COPPE/UFRJ, Rio de Janeiro, Brazil, 2001.

[73] CORSON, S., MACKER, J., *Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations*, 1999, <ftp://ftp.funet.fi/pub/standards/RFC/rfc2501.txt>.

[74] GOUVEIA, L., ALCÂNTARA, J. M. S., GÁLVEZ-DURAND, F., *A strategy for designing the processor of a mobile terminal with WAP suppor*. In: *Proceedings of SAWCAS*, Rio de Janeiro, Novembro 2001.

[75] VIEIRA, A. C. C., ALCÂNTARA, J. M. S., GÁLVEZ-DURAND, F, *A methodology for dynamic power consumption estimation using VHDL descrition*, published in the proceedings of the 15th Symposium on Integrated Circuits and System – SBCCI, September 2002, Brazil.

[76] MONARCH, *Mobile Networking Architectures*. Disponível em: <http://www.monarch.cs.cmu.edu>, acesso em abr 2003.

[77] JOHANSSON, P., et al., *Scenario-based performance analysis of routing protocols for mobile ad-hoc networks*. *In* International Conference on Mobile Computing and Networking (MobiCom'99), 1999, pp. 195-206.

[78] *Specification of the Bluetooth System, version 1.1*. February 2001, Official Bluetooth Website <http://www.bluetooth.com/>.

[79] ZIMMERMANN, PHILIP R., *The Official PGP User's Guide*, MIT Press May 1995 ISBN 0-262-74017-6 216 pp.

[80] VIEIRA, A. C. C., A. PIRES, PEDROZA, A. C. P., MESQUITA FILHO, A. C, *Performance Evaluation of Hardware Cryptographic Algorithms for Ad Hoc Network*, published in 11th International Conference on Telecommunications – ICT2004,. August 2004, Brasil.

[81] BROWN, M. *et al*. *SW Implementation of the Nist Elliptic Curves over Prime Field, Topics in Cryptology*. CT RSA, 2001.

[82] BLAKE, I., SEROUSSI, G., SMART, N. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.

[83] *ECC On Line Tutorial*. Disponível em <http://www.certicom.com>. Acesso em: 27/10/2004.

[84] ANGELFRIET, Arnoud. *Elliptic curve cryptography*. 01 jan. 2004. Disponível em <http://www.iusmentis.com/technology/encryption/diffie-hellman>. Acesso em: 27/10/2004.

[85] 3rd Generation Partnership Project, Technical Specification Group Services and Systems Aspects, 3G Security, *Cryptographic Algorithm Requirements*, 3G TS 33.105, version 3.2.0-1999.

[86] AGRAWAL, D. P., *Future directions in mobile computing and networking systems*, Mobile Computing and Communications Review, Vol. Vol. 3, pp. 13-18, Oct 1999.

[87] BOYD, C., *Modern data encryption*, Electronics and Communication Enginnering Journal, vol. 5, no 5, october 1993, pp 271-278.

[88] BRICKELL, E. F., ODLYZKO, A. M., *Cryptanalysis: A survey of recent results*, proc. IEEE, vol 76, no 5, may 1988, pp 578-593.

[89] BROCH, J., et al., *Performance of multi-hop wireless ad hoc network routing protocols*, Proceedings of the fourth annual ACM/IEEE international conference on Mobile computing and networking, p 85-97, October 25-30, 1998, Dalas, Texas, USA.

[90] BURD, T. D., *Energy-Efficient Processor System Design*. Ph.D thesis, University of California, Berkeley, Spring 2001.

[91] CORSON, S., MACKER, J., CIRINCIONE, G., *Internet-based Mobile Ad Hoc Networking*, published in the IEEE Internet Computing, Jul/Aug 1999.

[92] ELGAMAL, T., *A Public key cryptosystems and signature scheme based on discrete logarithms*. *IEE Transactions on Information Theory*, pp. 469-472, 1985.

[93] FEGHHI, Jalal, WILLIAMS, Peter; FEGHHI, Jalil, *Digital Certificates: Applied Internet Security*. New York: Addison-Wesley, 1998. 453p.

[94] FOKINE, Klas, *Key Management in Ad Hoc Networks*. Institutionen för Systemteknik 581 83 Lk, Swedish, http://www.ep.liu.se/exjobb/isy/2002/3322, 2002.

[95] FRANÇA, F., ALVES, V., GRANJA, E., *Edge Reversal Based Asynchronos Timing Scheme*. International Symposyum on Circuits and Systems, Monterrey, USA, 1998.

[96] *General Report on the Design, Specification and Evolution of 3GPP Standard Confidentiality and Integrity Algorithm*. <http://www.3gpp.org/ftp/Specs/html-info/33908.htm> Acesso em: 10 out 2002.

[97] GOODMAN, J. R., *Energy Scalable Reconfigurable Cryptographic Hardware for Portable Applications*, Ph.D thesis, Massachusetts Institute of Technology, 2000.

[98] HAAS, Z. J., ZHOU, L., *Securing ad hoc networks*, IEEE Network Magazine, vol. Vol 13, nov/dec 1999.

[99] KIYOMICHI, Araki, SATOH, Takakazu, MIURA, Shinji, *Overview of Elliptic Curve Cryptography*. In: Proceeding of Public Key Cryptography, LNCS, pp. 29-49, Springer-Verlag, 1998.

[100] KONG, J., ZERFOS, P., LUO, H., LU, S., ZHANG, L., *Providing robust and ubiquitous security support for mobile ad-hoc networks*, tech. rep., Computer Science Department, University of California at Los Angeles, 2001.

[101] KURUP, P., ABBASI, T., *Logic Synthesis Using Synopsys* Second Edition Kluwer Academic Publishers, 1997.

[102] LALANA KAGAL, A. J., UNDERCOFFER, Jeffrey, FININ, T., *Vigil: Enforcing security in ubiquitous environments*, tech. rep., University of Maryland, Baltimore County, 2001.

[103] MARNANE, W., and DALY, A., *Efficient Architectures for implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic*. *In International Syposium on Field Programable Gate Arrays*, pages 40–49. ACM Sigda, Feb 2002.

[104] MCDONALD, A., ZNATI, T., *A Mobility-Based Framework for Adaptive Clustering in Wireless Ad Hoc Networks*, in Wireless Ad Hoc Networks. IEEE JSAC, August 1999.

[105] MHOR W., S. Onoe, *The 3GPP Proposal for IMT-2000*, published in IEEE Communication Magazine, December, 1999.

[106] *Mobile Ad hoc Networks (MANET)*. Disponível em: <http://www.ietf.org/html.charters/manet-charter.html>, acesso em set 2002.

[107] MONTGOMERY, P. L., *Modular Multiplication without trial division*. *Math. Computation*, 44:519–521, 1985.

[108] NICHOLS, Randall K., *ICSA Guide to Cryptography*. New York: McGraw Hill, 1998. 840p.

[109] NYBERG, K., KNUDSEN, L., *Provable Security against Differential Cryptanalysis*, published in the Journal of Cryptology, vol. 8, No.1, 1995.

[110] NYBERG, K., *Linear Approximation of Block Ciphers*, published in the Proceedings of Eurocrypt´94. Springer-Verlag 1995.

[111] OBRACZKA, K., TSUDIK, G., *Multicast routing issues in ad hoc networks*, IEEE ICUPC, October 1998.

[112] PFLEEGER, Charles P., *Security in Computing*. New Jersey: Prentice Hall, 1996. 574p.

[113] *Report on the Evaluation of 3GPP Standard Confidentiality and Integrity Algorithm*, ETSI/SAGE 3GPP specification, number AS WG3 S3-000104-1999.

[114] SALOMÃO, S. L. C., et al., *Hardware Implementation of KASUMI Cryptographic Algorithm for Third Generation Mobile Systems*. Published in the 1st IEEE South-American Workshop on Circuits and Systems – SAWCAS´2000, nov, 2000, Brazil.

[115] SALOMÃO, S. L. C., et al., *Improved IDEA*, published in the proceedings of the 13th Symposium on Integrated Circuits and System Design -–SBCCI 2000, September 2000, Brazil.

[116] SCHMIDT, M., *Subscriptionless mobile networking: Anonymity and privacy aspects within personal area networks*, tech. rep., Institute for Data Communications Systems, University of Siegen, Germany, 2001.

[117] SCHNEIER, Bruce, *E-Mail Security: How to Keep Your Electronic Messages Private*. New York: John Wiley & Sons, 1995. 384p.

[118] SHAND M., VUILLEMIN J., *Fast implementation of RSA cryptography*, *in* Proc. 11th IEEE Symp. Computer Arithmetic, 1993, pp. 252–259.

[119] SMITH, Richard E., *Internet Cryptography*. New York: Addison-Wesley, 1997. 356p.

[120] SOARES, Luiz F. G.; LEMOS, Guido; COLCHER, Sérgio, *Redes de Computadores: Das LANs, MANs e WANs às Redes ATM*. 2ª Edição, Rio de Janeiro: Ed. Campus, 1995. 740p. Cap. 17: Segurança em Redes de Computadores, p.447-488.

[121] STEINER, M., TSUDIK, G., WAIDNER, M., *CLIQUES: A New Approach to Group Key Agreement*, published in 18th International Conference on Distributed Computing Systems, may 1998.

[122] STINSON, Douglas R., *Cryptography: Theory and Practice*. New York: CRC Press, 1995. 448p.

[123] STONEBURNER G., *Underlying technical models for information technology security*. Recommendations of the National Institute of Standards and Technology, NIST Special Publication 800-33, December 2001.

[124] TANENBAUM, Andrew S., *Computers Networks*. 3rd Edition, New Jersey: Prentice Hall, 1996. 813p. Cap. 7: The Application Layer, p.577-766.

[125] TRÄSKBÄCK, M., *Security of Bluetooth: An overview of Bluetooth Security*, Department of Electrical and Communications Engineering Helsinki University of Technology 2000, <http://www.cs.hut.fi/Opinnot/Tik-86.174/Bluetooth_ Security.pdf> Acesso em: 10 set 2004

[126] TRINTA, Fernando Antonio Mota, MACÊDO, Rodrigo Cavalcanti de. *Um Estudo sobre Criptografia e Assinatura Digital*. Departamento de Informática. Universidade Federal de Pernambuco, Setembro 1998. Disponível em <http://www.di.ufpe.br/~flash/ais98/cripto/criptografia.htm>. Acesso em: 13 jul. 2004.

[127] VANHALA, A., *Security in ad hoc networks*, in Research Seminar on Security in Distributed Systems, 2000.

[128] VENKATRAMAN, L., AGRAWAL, D., *A novel authentication scheme for Ad Hoc networks*, 2nd IEEE Wireless Communications and Networking Conference, Chicago Il, September 23-28, 2000.

[129] VIEIRA, A. C. C., ALVES, V. C., et al., *Implementation of Cryptographic Applications on the Reconfigurable FPGA Coprocessor microEnable*, published in the proceedings of the 13th Symposium on Integrated Circuits and System Design -–SBCCI 2000, September 2000, Brazil.

[130] VIEIRA, A. C. C., *The RSA cryptographic algorithm study aiming at its development in hardware*, Master Thesis, COPPE/UFRJ, Rio de Janeiro, Brazil, 1999.