Segurança em sistemas embarcados críticos – utilização de criptografia para comunicação segura

Daniel Fernando Pigatto

Segurança em sistemas embarcados críticos – utilização de criptografia para comunicação segura

Daniel Fernando Pigatto

Orientadora: **Profa. Dra. Kalinka Regina Lucas Jaquie Castelo Branco**

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

USP – São Carlos
Agosto de 2012

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Pigatto, Daniel Fernando
P628s Segurança em sistemas embarcados críticos - utilização de criptografia para comunicação segura / Daniel Fernando Pigatto; orientadora Kalinka Regina Lucas Jaquie Castelo Branco. -- São Carlos, 2012.
88 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2012.

1. Sistemas embarcados críticos. 2. Segurança. 3. Criptografia. 4. Redes de computadores. 5. Sistemas distribuídos. I. Branco, Kalinka Regina Lucas Jaquie Castelo, orient. II. Título.

Agradecimentos

Aos meus pais, Agenor e Dinora que nunca mediram esforços para que hoje eu pudesse estar finalizando mais este projeto. Ficam meus agradecimentos pelo apoio moral e financeiro e por acreditarem na minha capacidade de concluir um mestrado. À minha irmã Fernanda e aos demais familiares por terem depositado em mim suas esperanças e por todo apoio moral nestes dois anos.

À amizade e orientação da Profª Drª Kalinka Regina Lucas Jaquie Castelo Branco, que me acolheu desde o primeiro dia em São Carlos e me orientou em questões técnicas e até mesmo em momentos de dificuldades pessoais. Agradeço todo o empenho e competência para que este projeto pudesse se tornar realidade e todas as horas dedicadas a me ajudar a construir uma consciência acadêmica ampla, sólida e ética.

Aos colegas de LaSDPC e LSEC: Adriana Molina Centurion, Dionisio Machado Leite Filho, Douglas Rodrigues, Edvard Oliveira, Edwin Luis Choquehuanca Mamani, Elvis de Castro Lima, Fausto Guzzo da Costa, Julio Cezar Estrella, Lourenço Alves Pereira Junior, Luis Nakamura, Maycon Leone Peixoto, Natássya da Silva, Paulo Gurgel, Pedro Felipe do Prado, Rayner de Melo Pires, Renê de Souza Pinto, Roni Guillermo Apaza Aceituno, Thiago Caproni e muitos outros que certamente faltarão nesta lista.

Aos colegas de república, que dividiram diversos momentos de estudo e de descontração: Alessandro Nakamuta, Bruno Guazzelli Batista, Bruno Tardiole Kuehne, Gabriel Massote Prado, João Paulo Orlando e Paulo Sérgio Franco Eustáquio.

A quatro colegas que tiveram contribuições mais específicas: João Paulo Orlando pela parceria no início de uma nova fase em cidade e universidade novas; Natássya da Silva pelas contribuições diretas na parte de implementação do meu projeto durante o desenvolvimento de seus projetos de IC e TCC; Renê de Souza Pinto pelo apoio na parte de *hardware* e de preparação do ambiente de experimentos; e Rayner de Melo Pires que dedicou horas de trabalho para me ajudar a concluir este projeto com sucesso.

Aos docentes do ICMC, em especial do Grupo de Sistemas Distribuídos, que sempre contribuíram em seminários e em disciplinas cursadas.

Aos alunos de graduação da Universidade de São Paulo que desenvolveram iniciações científicas e trabalhos de conclusão de curso dos quais eu pude participar ativamente: Natássya da Silva, Felipe Sikansi, Gabriel Salla, Alex Sartin, Rafael Romeiro e Julio Rodrigues.

Aos meus professores de graduação da URI - Erechim, os primeiros incentivadores para que este sonho pudesse se tornar realidade, em especial ao Prof. MSc. Alexandro Magno dos Santos Adário, à Profª MSc. Carla Lisiane de Oliveira Castanho, ao Prof. MSc. Fabio Asturian Zanin, ao Prof. MSc. Neilor Avelino Tonin e ao Prof. MSc. Paulo Ricardo Rodegheri.

Aos amigos, tanto os de Erechim quanto os que fiz em São Carlos. É impossível citar nomes e não esquecer alguém, por isso deixo meu agradecimento a todos que de uma forma ou de outra tiveram alguma participação.

E por fim, mas não menos importante, ao CNPq pelo apoio financeiro, ao Instituto de Ciências Matemáticas e de Computação e à Universidade de São Paulo pela estrutura disponibilizada para realização deste projeto.

Resumo

Este trabalho contempla o estudo de algoritmos criptográficos para prover comunicação segura entre sistemas embarcados críticos tendo em vista o grande crescimento na utilização e disseminação desse tipo de sistema, bem como a alta necessidade em se assegurar as informações que são enviadas e recebidas. Um dos desafios a serem contemplados é o estudo e a avaliação do impacto no desempenho desses sistemas, levando em consideração limitações de recursos inerentes a esta plataforma e a criticidade da comunicação em sistemas de tempo real. Os experimentos realizados são de cunho prático por meio de um protótipo implementado em *kits* Gumstix Overo EVM. Os resultados avaliam os principais algoritmos de criptografia, provendo informações que podem auxiliar na escolha de uma solução criptográfica própria para ambientes embarcados.

Abstract

This research includes the study of cryptographic algorithms to ensure communication among critical embedded systems, considering the large growth of application and dissemination of this type of system, as well as the high necessity to ensure the security of information that is exchanged. One of the challenges to be addressed is the study and evaluation of the performance impact on these systems, considering resource constraints inherent to the platform and the criticality of the communication in real-time systems. The experiments are of practical lead through a prototype implemented in Gumstix Overo EVM kits. The results evaluate the main encryption algorithms, providing information that may help in choosing a cryptographic solution suitable for embedded environments.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

3DES - *Triple DES*

AES - *Advanced Encryption Standard*

ARM - *Advanced RISC Machine*

CISC - *Complex Instruction Set Computer*

CPU - *Central Processment Unit*

DES - *Data Encryption Standard*

DoS - *Denial of Service*

ECC - *Elliptic Curve Cryptography*

EVM - *Earned Value Management*

FIPS - *Federal Information Processing Standard*

GPS - *Global Positioning System*

INCT-SEC - Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos

LCD - *Liquid Crystal Display*

LTS - *Long Term Support*

Miracl - *Multiprecision Integer and Rational Arithmetic C/C++ Library*

NIST - *National Institute of Standards and Technology*

PDA - *Personal Digital Assistant*

RAM - *Random-Access Memory*

RC2 - *Ron's Code ou Rivest Cipher 2*

RC4 - *Ron's Code ou Rivest Cipher 4*

RC6 - *Ron's Code ou Rivest Cipher 6*

Relic - *Efficient LIbrary for Cryptography*

RISC - *Reduced Instruction Set Computing*

ROM - *Read-Only Memory*

RSA - *Rivest, Shamir, Adleman*

SOA - Arquitetura Orientada a Serviço

SSL - *Secure Sockets Layer*

TLS - *Transport Layer Security*

USB - *Universal Serial Bus*

VANET - *Vehicular Ad-hoc Network*

VANT - Veículo Aéreo Não Tripulados

VTNT - Veículo Terrestre Não Tripulado

Capítulo

1

Introdução

1.1 Contextualização

A computação é uma área que vem experimentando um desenvolvimento sem comparação ao longo das últimas décadas. Dentre as mudanças mais significativas destaca-se a alta conectividade dos recursos computacionais, o que permitiu a solução de vários problemas de modo mais eficiente, a um custo relativamente mais baixo.

A motivação para a conexão dos diferentes recursos dos sistemas computacionais pode variar desde o simples compartilhamento de discos e impressoras, passando por motivações ligadas a entretenimento doméstico, até a união de uma grande quantidade de processadores, viabilizando um aumento na potência computacional disponível, provendo melhor tolerância a falhas e maior mobilidade dos usuários.

Mobilidade é um dos principais propulsores do surgimento de sistemas embarcados. Eles aplicam-se a cenários onde, normalmente, a acessibilidade é dificultada e se dedicam a tarefas específicas, como por exemplo, sensores de queimadas em meio a uma floresta que possuem a função única de identificar focos de incêndio ou ainda veículos aéreos não tripulados (VANTs) que podem executar missões de sobrevoo de florestas para identificar possíveis crimes ambientais.

De acordo com Gill (2005), sistema embarcado é um tipo de sistema que sofreu modificações significativas ao longo das últimas décadas e deu origem a uma área de pesquisas um tanto quanto independente. O escopo de pesquisas em torno destes sistemas expandiu de arquiteturas limitadas para uma nova geração de dispositivos voltados a aplicações de

tempo real, complexas e distribuídas. Tendo em vista a disseminação destes dispositivos, a preocupação com a segurança da informação transmitida e recebida por eles passa a ser um crescente tópico de pesquisa. Normalmente segurança é implementada por meio de criptografia, tema que será amplamente abordado no Capítulo 3.

Este projeto de mestrado visa o estudo de algoritmos criptográficos para sistemas embarcados críticos tendo em vista suprir a crescente demanda por segurança nestes ambientes, bem como a alta necessidade de se garantir integridade, confidencialidade e autenticidade das informações que trafegam nesse tipo de sistema. Um dos desafios a serem contemplados é o estudo e a avaliação do impacto da aplicação de criptografia no desempenho de tais sistemas.

1.2 Problemática, Motivação e Objetivos

Há algum tempo não se concebe um sistema computacional completamente isolado dos demais, ocorrendo a necessidade constante de comunicação com outros sistemas, sejam eles computacionais ou não. Os principais problemas de segurança surgem justamente na comunicação, de modo que usuários não autorizados sejam capazes de se passar por usuários conectados e se conectar, furtando informações, interrompendo o funcionamento de serviços e até mesmo danificando física ou logicamente os dispositivos. Isso permite que um invasor, caso consiga se passar por um usuário legítimo, tenha o acesso facilitado a todas as informações que desejar manipular.

Quando se faz uso de sistemas embarcados críticos torna-se inevitável a execução de tarefas em tempo real e que fazem uso de computadores geralmente dedicados. Eles são geralmente submetidos a situações extremas, como altas ou baixas temperaturas, altas velocidades, operações perigosas, etc. Desse modo o mau funcionamento e a maior suscetibilidade que os mesmos apresentam a falhas podem levar a prejuízos que normalmente resultam em perdas de grandes quantias monetárias e até mesmo em perdas humanas (Wolf, 2008).

Os desafios para implementar sistemas embarcados exigem novas abordagens para a segurança de todos os aspectos do projeto destes sistemas. Cabe ressaltar que técnicas de segurança desenvolvidas para a computação empresarial e doméstica não satisfazem, na maioria das vezes, requisitos de aplicações embarcadas.

O trabalho apresentado por Tiri et al. (2005), elenca as vulnerabilidades dos sistemas embarcados, como por exemplo o fato de um circuito integrado permitir a obtenção de informações relacionadas à chave secreta por meio do monitoramento do consumo de energia. Isso deixa clara a necessidade de se prover e utilizar mecanismos eficientes e que permitam que esses sistemas se tornem menos vulneráveis e mais seguros contra ataques dos mais variados tipos.

Outros trabalhos como Potlapally et al. (2003) e Koopman (2004), apresentam os problemas relacionados ao consumo de energia quando se faz uso de algoritmos criptográficos que possuem cálculos intensos. Existe a necessidade de se avaliar até onde o consumo de energia gerado por uma solução criptográfica pode ser vantajoso de modo a encontrar um equilíbrio entre a desvantagem do aumento no consumo de energia e a vantagem da segurança proporcionada por essa criptografia.

Muitos dos trabalhos encontrados na literatura estão preocupados em apresentar características e requisitos para sistemas embarcados não se prendendo a detalhes de projeto, como por exemplo, desconsiderando segurança como uma nova dimensão no desenvolvimento de sistemas embarcados, tão importante quanto outras métricas como custo, performance e energia (Ravi et al., 2004; Kocher et al., 2004). Embora boa parte dos estudos deixa clara a imaturidade da área de segurança aplicada a sistemas embarcados críticos, normalmente a comunicação entre tais sistemas não é implementada considerando aspectos de segurança (Henzinger e Sifakis, 2006) (Brändle e Naedele, 2008).

Quando se pensa em comunicação sem fio nesses sistemas há um considerável aumento da necessidade de garantir a confidencialidade, a integridade e a autenticidade das informações transmitidas (Wollinger et al., 2003), de modo que esforços devem ser concentrados para que a troca de informações seja segura. Dois problemas de segurança focos deste trabalho são a dificuldade da adição de criptografia em canais de comunicação e o consumo que isso pode acarretar.

Desta forma, o objetivo central deste trabalho é estudar e avaliar a utilização de criptografia para prover uma forma segura de comunicação em sistemas embarcados críticos, levando-se em consideração o desempenho avaliado em termos de utilização dos recursos computacionais.

Um segundo objetivo é o de se obter *expertise* na área de segurança em sistemas embarcados críticos para prover uma abordagem adequada e uma distinção entre os algoritmos, tipos e tamanhos de chave mais apropriados para cada tipo de equipamento considerado. Dessa forma será possível reunir e apresentar os principais conceitos e aspectos referentes a este tipo de sistema com experimentações práticas que permitem a aplicação de técnicas estatísticas para avaliação de desempenho em sistemas computacionais (Jain, 1991).

Neste trabalho espera-se obter resultados relevantes tanto para a área de comunicação quanto para a de sistemas embarcados, principalmente considerando-se que sistemas embarcados críticos têm se tornado algo amplamente utilizado, não podendo ter seu funcionamento prejudicado ou paralisado, e que a comunicação sem fio segura destes sistemas é de vital importância para o bom funcionamento dos mesmos.

1.3 Estrutura do Texto

Este trabalho está dividido em seis capítulos. O Capítulo 2 apresenta uma revisão bibliográfica sobre sistemas embarcados críticos evidenciando as principais diferenças destes com sistemas tradicionais. O Capítulo 3 inicia uma revisão bibliográfica de conceitos de segurança e aborda, ao fim, segurança no campo dos sistemas embarcados críticos. O Capítulo 4 caracteriza a preparação do ambiente de experimentações e discorre, brevemente, sobre alguns conceitos de avaliação de desempenho de sistemas computacionais, requisitos para o entendimento do capítulo seguinte. O Capítulo 5 apresenta cinco estudos de caso que foram realizados para avaliar situações específicas de aplicação deste trabalho, os quais apresentam os resultados obtidos e discussões a respeito dos mesmos. E, por fim, o Capítulo 6 apresenta as conclusões deste trabalho.

Capítulo

2

Sistemas Embarcados Críticos

2.1 Considerações Iniciais

Este capítulo tem por objetivo introduzir conceitos relacionados a sistemas embarcados, evidenciando suas características e os principais problemas enfrentados por projetistas deste tipo de sistemas. Uma revisão bibliográfica geral será apresentada com ligeiras referências ao ambiente de experimentações selecionado para este trabalho, o qual será melhor detalhado no Capítulo 4.

2.2 Conceituação

Observando-se os aparelhos eletrônicos que nos cercam atualmente, não é difícil encontrar exemplos de sistemas embarcados. O forno micro-ondas, por exemplo, é dotado de uma capacidade computacional que o permite desempenhar funções específicas. Ao se pressionar a tecla PIPOCA, um sistema interno deve ajustar a potência correta, selecionar e medir o tempo em que o forno ficará acionado e emitir um sinal quando a tarefa for concluída. Para realizar esta simples operação, o "cérebro" do forno deve receber sinais de sensores (como o da porta, para saber se a mesma foi realmente fechada), acionar o equipamento de potência, verificar o tempo da operação, acionar o motor que fará a rotação do prato, permitir que o usuário interrompa a operação a qualquer momento, atualizar o visor e medir quanto tempo se passou desde o início da operação.

Dispositivos responsáveis por funções dedicadas, como os presentes em um micro-ondas, são chamados de sistemas embarcados. Estes sistemas estão inseridos em um sistema mais amplo, como é o caso de um refrigerador, onde existe um sistema de refrigeração, composto de um motor e uma serpentina, mas existe um sistema embarcado que recebe comandos de ajustes de temperatura e monitora o funcionamento do motor de refrigeração, ligando-o e desligando-o ao atingir temperaturas pré-programadas. O mesmo existe em televisores, rádios, *players*, roteadores, semáforos, celulares, automóveis, aviões etc., sendo cada um deles especificamente desenvolvido para desempenhar funções inerentes aos sistemas maiores.

Dentre os sistemas embarcados, alguns deles podem ser considerados críticos. Os domínios de aviônicos e automotivos são exemplos clássicos onde custos, ciclos de produção curtos e requisitos que visam dependabilidade, robustez, segurança, controle de emissão de poluentes e demandas específicas de cada cenário influenciam na obtenção de um produto final (Januzaj et al., 2010). Devido ao alto investimento em pesquisas nesta área, a perda de um produto destes acarreta em perdas financeiras consideravelmente altas.

Diversas são as definições formais de sistemas embarcados encontradas na literatura. Para Barr (1999), um sistema embarcado (do inglês, *embedded system*, ou ainda sistema embutido) é uma combinação de *software* e *hardware* projetada para desempenhar uma tarefa específica, ou seja, um sistema baseado em microprocessador que opera embutido como subsistema de um determinado sistema maior, seja ele computacional ou não. Na visão de Berger (2002), mais focada em *hardware*, sistemas embarcados são compostos, normalmente, por uma unidade de processamento, a qual consiste em um circuito integrado fixado a um circuito impresso.

Com uma visão mais voltada para o mercado, Bloomfield et al. (2005) definiu sistemas embarcados como ubíquos, pois podem ser encontrados em uma vasta quantidade de produtos, caros e possuidores de um vasto mercado.

Já de acordo com Barr Groups (2012), sistemas embarcados são uma combinação de *hardware* e *software*, e talvez mais algumas peças mecânicas, destinadas a executar uma função específica. Em alguns casos, são parte de um sistema ou produto. Exemplos destes sistemas são: fornos micro-ondas, telefones celulares, calculadoras, relógios digitais, mísseis de cruzeiro, receptores GPS (*Global Positioning System*), monitores cardíacos, impressoras a *laser*, radares, controladores de motores, câmeras digitais, semáforos, controles remotos, máquinas de fax, *pagers*, caixas registadoras, escadas rolantes, bombas de gás, leitores de cartão, termostatos, analisadores de grãos, entre outros.

Quando comparados a computadores *desktop* tradicionais, os sistemas embarcados diferem em uma série de características, tais como (Berger, 2002): sistemas embarcados são dedicados a tarefas específicas, enquanto *desktops* são considerados plataformas de computação genéricas; são comumente sensíveis a custo; sofrem restrições de tempo real; quando uma falha ocorre em um sistema embarcado, as implicações são, normalmente,

mais graves do que em *desktops*, justamente porque são, na maioria das vezes, utilizados em situações de risco; trabalham com limitações de energia; frequentemente operam sob condições climáticas extremas; possuem menos recursos computacionais que computadores *desktop*, uma realidade que vem mudando em relação à capacidade de processamento, mas ainda não com relação à quantidade de memória; armazenam todo seu código diretamente em memória ROM; e requerem ferramentas e metodologias específicas para serem projetados com eficiência.

2.3 Características de Sistemas Embarcados

A computação embarcada difere da computação tradicional em vários sentidos. A busca por funcionalidade é comum entre ambas, porém ao trabalhar com sistemas embarcados existem outros fatores que devem ser levados em consideração tais como limitações de recursos de processamento, memória e energia. Estas medidas não são consideradas neste trabalho, pois são amplamente abordadas por outros trabalhos na área. Este trabalho se prende à avaliação de tempo de criptografia e *throughput* médios obtidos por cada algoritmo criptográfico nas condições que serão apresentadas na seção experimental.

Os sistemas computacionais embarcados devem prover, muitas vezes, funcionalidades sofisticadas, tais como (Wolf, 2008):

- **Algoritmos complexos**: as operações realizadas pelo microprocessador podem ser muito sofisticadas. Exemplo disso é o microprocessador que controla o motor de um automóvel, o qual deve executar complicadas funções de filtragem para otimizar o desempenho do carro, minimizando a poluição e a utilização de combustível;
- **Interface de usuário**: microprocessadores são frequentemente utilizados para controle de complexas interfaces de usuário que podem incluir vários menus e muitas opções, o que pode ser visto nas interfaces modernas adotadas por sistemas atualmente. A navegação por mapas em GPS é um bom exemplo de interface sofisticada.

Para aumentar a complexidade, ao se trabalhar com sistemas embarcados as operações realizadas devem voltar-se frequentemente para cumprir certos requisitos (Wolf, 2008):

- **Tempo Real**: muitos sistemas embarcados têm que operar em tempo real, ou seja, se a resposta não for entregue em determinado intervalo de tempo, geralmente muito pequeno, o sistema pode ter sérios problemas. Em alguns casos, a existência de falhas no cumprimento de tarefas pode até ameaçar vidas. Em outros, atrasos podem não representar problemas de segurança, mas causar o descontentamento de clientes;

- ***Multirate***: não apenas as operações devem ser concluídas dentro de prazos estabelecidos, mas muitos sistemas embarcados possuem várias tarefas acontecendo ao mesmo tempo e com requisitos temporais de processamento diferentes. Eles podem, simultaneamente, controlar algumas operações que são executadas em baixas taxas de execução e outras que funcionam em altas taxas. Aplicações multimídia são exemplos de comportamento *multirate*. O áudio e o vídeo de um fluxo multimídia executam a ritmos muito diferentes, mas devem permanecer bem sincronizados. O não cumprimento desta premissa por qualquer uma das partes (áudio ou vídeo) pode comprometer uma apresentação.

Além destes fatores citados que devem ser lembrados durante o projeto de sistemas embarcados, existem preocupações quanto aos custos inerentes a estes sistemas. Wolf (2008) cita os seguintes:

- **Custo de fabricação**: o custo total da construção do sistema é bastante importante em muitos casos. O custo de fabricação é determinado por alguns fatores, incluindo o tipo de microprocessador utilizado, a quantidade de memória necessária e os tipos de dispositivos de entrada/saída;

- **Potência e energia**: o consumo de energia afeta diretamente o custo do *hardware*, já que um sistema pode exigir um maior suprimento de energia. O consumo de energia afeta a vida da bateria, que é essencial em muitas aplicações, bem como o consumo por dissipação, que pode representar um problema até mesmo em aplicações *desktop*.

Além dos fatores citados por Wolf (2008) tem-se também o fator citado por Bloomfield et al. (2005), que acrescenta:

- **Custo de desenvolvimento (*software*)**: o *software* associado a um sistema embarcado normalmente custa de U$15 a U$30 por linha de código, podendo chegar a valores maiores que U$100. Quando trata-se de aplicações críticas, o custo por linha de código chega a aproximadamente U$1.000, o que pode se tornar muito caro até mesmo para uma aplicação pequena que possua, por exemplo, 5.000 linhas.

Programadores de propósito geral não têm, na maioria dos casos, preocupações com o desempenho de suas aplicações, uma vez que podem utilizar recursos disponíveis sem limitações e precisam que seus programas rodem “rápido o suficiente”, e não “o mais rápido possível”.

Em sistemas embarcados, por outro lado, desempenho é um objetivo claro de todo desenvolvedor, ou seja, os programas devem atender a prazos bem definidos. No cerne da computação embarcada está a computação de tempo real, a qual é a ciência e a arte

de programar respeitando prazos específicos de tempo (Wolf, 2008). O programa recebe dados de entrada e possui um prazo para realização das operações computacionais necessárias. Se o programa não produzir o resultado necessário dentro do prazo, então o programa não funciona, mesmo que a eventual saída produzida seja correta.

Para desenvolver mecanismos de segurança em *software* para sistemas embarcados críticos faz-se necessário conhecer a arquitetura geral destes sistemas de modo a levar em consideração os elementos de *hardware* sobre os quais algoritmos criptográficos estarão executando. A próxima seção apresenta, de modo geral, a arquitetura de sistemas embarcados e faz referências ao ambiente de experimentações adotado neste trabalho, que será detalhado no Capítulo 4.

2.4 Arquitetura Geral de Sistemas Embarcados

Todo sistema embarcado, assim como outros sistemas computacionais, possui elementos que o compõem fisicamente e que interagem entre si para que o funcionamento do sistema como um todo seja possível. Esta seção apresenta particularidades em relação à memória, à comunicação e ao microprocessador empregados em sistemas embarcados.

2.4.1 Memória

Um componente importante em um sistema embarcado é a memória. Muitas vezes, há a necessidade do uso de mais de um tipo de memória, de modo a armazenar dados do *software* embarcado e informações temporárias provenientes de processamentos ou coletas realizadas pelo sistema. Existem ainda memórias dedicadas a armazenar instruções básicas do processador.

As memórias em geral apresentam as seguintes características (Oliveira e Andrade, 2006):

- **Tempo de acesso**: tempo necessário para realizar acesso à memória e efetuar uma operação de leitura ou escrita;
- **Capacidade**: quantidade efetiva de dados que podem ser armazenados no interior da memória;
- **Não volatilidade**: capacidade da memória não perder os dados armazenados mesmo quando não houver energia elétrica;
- **Tempo de latência**: intervalo mínimo entre cada operação de leitura ou escrita na memória. O mau uso desta informação pode acarretar na utilização de espaços indevidos de memória, podendo causar a perda ou manipulação incorreta de dados.

As diferentes combinações das características supracitadas dão origem a diferentes tipos de memórias. Existem casos em que uma memória volátil, por exemplo, pode ser empregada em um sistema de modo a reduzir seu preço final, desde que a perda de informações armazenadas naquela memória não caracterize um problema. Casos como este podem ser vistos em situações onde uma nova coleta de informações pode ser realizada caso haja uma interrupção do fornecimento de energia para aquela memória ou sistema, sendo restabelecida na sequência.

2.4.2 Comunicação

O termo comunicação, no escopo deste trabalho, refere-se à comunicação entre dois ou mais sistemas embarcados, ou ainda entre um sistema embarcado e uma estação de controle.

Existem formas cada vez mais complexas de se conectar esses sistemas embarcados. Isso se dá uma vez que muitos sistemas embarcados complexos estão associados a um sistema de informação de alto nível. Nesse sentido, Arquitetura Orientada a Serviço (SOA), um paradigma para implementação de aplicações voltado para a *Web*, pode prover a integração de serviços de sistemas embarcados de baixo nível e de serviços de sistemas de informação de alto nível (Kakanakov, 2005) (Kakanakov e Spasov, 2005) (Thramboulidis et al., 2008) (Moritz et al., 2008). Na prática, a utilização de SOA em sistemas embarcados pode fornecer uma série de benefícios, tais como: configuração separada do ambiente, melhoria da reusabilidade e manutenção, nível maior de abstração e de interoperabilidade, interface mais interativa entre dispositivos e sistemas de informação e facilidade de uso de serviços fornecidos por servidores cada vez mais poderosos.

Independentemente da complexidade em se conectar um sistema embarcado a um computador, a conexão não passa de um caminho que envia e recebe *bits*. Esta comunicação se dá por meio de portas específicas para comunicação. Neste trabalho de pesquisa, o *kit* Gumstix Overo EVM (*Earned Value Management*) foi utilizado, o qual apresenta algumas formas de conexão por meio de uma placa de expansão, tais como: *Bluetooth*, *Wireless* 802.11(b/g), *Ethernet* e USB. Este *kit* será melhor abordado na seção 4.2.

2.4.3 Microprocessador

O processador é uma das principais unidades da placa de um sistema computacional, uma vez que é diretamente responsável pelo processamento de instruções e dados. Um dispositivo eletrônico deve possuir pelo menos um processador mestre, o qual atuará como dispositivo de controle central. Pode ainda contar com processadores intermediários para auxiliar no processamento, todos controlados pelo processador mestre.

Existem diversos tipos de processadores desenvolvidos para embarcados disponíveis no mercado. Apesar do grande número de processadores para sistemas embarcados, eles podem ser classificados em "grupos", ou, em outras palavras, arquiteturas: CISC (*Complex Instruction Set Computer*) e RISC (*Reduced Instruction Set Computing*). Em resumo, o que diferencia uma arquitetura de outra, segundo Noergaard (2005), é a quantidade de instruções de código de máquina que ela consegue executar. Processadores são considerados parte de um mesmo grupo quando são capazes de executar o mesmo número de instruções de código de máquina.

A escolha por microprocessadores para aplicação em sistemas embarcados é justificada por duas razões (Wolf, 2008):

- São uma maneira muito eficiente de implementar sistemas digitais;
- Facilitam a concepção de famílias de produtos desenvolvidos para fornecer conjuntos de características diferentes com preços diferentes.

Microprocessadores executam programas de forma muito eficiente. Processadores RISC modernos podem executar, na maioria das vezes, uma instrução por ciclo de *clock*, enquanto que processadores de alta performance conseguem executar várias instruções por ciclo. Embora possa existir sobrecarga para interpretar instruções, muitas vezes é possível escondê-la por meio do uso correto de paralelismo dentro da CPU.

Dentre as várias arquiteturas existentes no mercado, destaca-se a arquitetura ARM. A *Advanced RISC Machine* é uma família da arquitetura RISC que vem sendo desenvolvida há anos, especificamente desde 1985 com o protótipo ARM1 (Wolf, 2008) (Sloss et al., 2004). A arquitetura é licenciada para companhias que manufaturam CPU ou integram o processador ARM em um sistema maior. De fato, ARM é uma família de processadores que compartilham princípios de projeto e um mesmo conjunto de instruções (Sloss et al., 2004).

Segundo Sloss et al. (2004), um programador pode visualizar o núcleo ARM como unidades funcionais interligadas por barramentos de dados. Estes dados entram no núcleo por meio de barramentos específicos e podem ser simplesmente itens de dados ou instruções para execução. Um decodificador traduz as instruções antes de executá-las e cada uma delas pertence a um conjunto específico de instruções.

O processador ARM, assim como todos os processadores RISC, usa uma arquitetura do tipo *load-store*, ou seja, possui dois tipos de instruções para transferência de dados dentro e fora do processador: instrução de cópia de dados dos registradores para a memória e instrução de cópia de dados da memória para os registradores (Sloss et al., 2004).

As principais características do processador ARM incluem: 16 registradores de uso geral, conjunto de instruções extensível com o uso de coprocessadores, baixo consumo de

energia e tamanho reduzido (Berger, 2002). É o processador que domina o mercado de sistemas embarcados (Hennessy e Patterson, 2011).

Neste projeto, o processador utilizado será o ARM Cortex-A8, que oferece uma gama de soluções para dispositivos como *smartphones*, telefones móveis, televisão digital, impressoras e servidores (ARM, 2012). Este processador é o que integra o *kit* Gumstix Overo EVM, arquitetura adotada como ambiente de experimentações deste trabalho.

2.5 Considerações Finais

Neste capítulo foram apresentados os conceitos básicos de sistemas embarcados críticos. Características como capacidade de armazenamento, processamento e consumo de energia são tópicos relevantes para essa plataforma. O *hardware* utilizado possui características específicas que devem ser levadas em consideração quando do desenvolvimento para a arquitetura.

Um problema inerente a essa plataforma é a questão de segurança da informação, que devido às limitações existentes em sistemas embarcados críticos, deve ser muito bem projetada além de passar por experimentações práticas que ajudem a verificar o comportamento dos principais algoritmos de criptografia quando executados nestes ambientes. Nesse sentido o capítulo que segue aborda conceitos de segurança em sistemas computacionais de modo geral e em sistemas embarcados críticos de modo específico.

Capítulo

3

Segurança da Comunicação em Sistemas Embarcados Críticos

3.1 Considerações Iniciais

Segurança em redes de computadores é uma área de intensas pesquisas. Isso se deve ao crescente uso de computadores nas últimas décadas e à interconexão deles em redes dos mais variados tipos e tamanhos, na maioria das vezes conectadas à rede mundial de computadores, a Internet. Com toda esta possibilidade de troca de informações e acesso a dados armazenados em bases espalhadas ao redor do mundo, organizações e até mesmo usuários domésticos passaram a se preocupar com a necessidade de implantar mecanismos para assegurar que seus recursos e informações estejam devidamente protegidos.

No que diz respeito a sistemas computacionais embarcados de ordem crítica, a segurança torna-se um requisito ainda mais desejado, uma vez que a troca de informações sigilosas entre estes dispositivos e, muitas vezes, com suas bases de controle, é constante.

Este capítulo apresenta conceitos básicos de segurança em redes de computadores, incluindo criptografia de chave simétrica e assimétrica e assinatura digital, e, ao fim, foca especificamente em segurança aplicada a sistemas embarcados críticos. Apresenta também a proposta principal deste trabalho, mostrando a necessidade real de realização de experimentações práticas que comprovem quais os algoritmos e tamanhos de chave mais apropriados para os cenários abordados.

3.2 Conceituação

De acordo com Stapko (2007), segurança de computadores consiste em proteger informações pessoais ou confidenciais e/ou recursos computacionais de indivíduos ou organizações que poderiam deliberadamente destruir ou utilizar tais informações para fins maliciosos. Algumas propriedades devem ser garantidas para uma completa e eficaz implementação de segurança em sistemas computacionais (Kurose et al., 2003; Bishop, 2005; Stallings, 2008):

- **Confidencialidade**: trata-se da ocultação de informações ou de recursos, protegendo-os contra acesso não autorizado. Um exemplo prático pode ser observado em instituições de ensino, onde o acesso à informação é restringido de acordo com classes de usuários. Existe ainda a preocupação em manter secretas as informações pessoais de cada aluno. Confidencialidade, portanto, é a garantia de que somente o remetente e o destinatário pretendido terão o poder de entender o conteúdo da mensagem. Se algum intruso conseguir interceptar a mensagem, não deverá conseguir extrair informações do texto cifrado (disfarçado, ou ilegível).

- **Autenticidade**: é a garantia de que a entidade participante da comunicação é realmente quem ela afirma ser. Remetente e destinatário precisam confirmar a identidade mútua. Quando a comunicação se dá pessoalmente entre seres humanos, esse problema é facilmente solucionado por reconhecimento visual. O problema existe quando a comunicação não permite que as partes sejam vistas (caso dos sistemas computacionais).

- **Integridade**: integridade se refere à confiabilidade dos dados ou recursos, ou seja, trata-se da garantia de que não houve mudanças durante a comunicação (ou seja, não contêm modificação, inserção, exclusão ou repetição). Extensões das técnicas de soma de verificação encontradas em protocolos de transporte e de enlace confiáveis podem ser utilizadas para proporcionar integridade à mensagem.

- **Não repúdio de mensagem**: o receptor pode, ainda, comprovar que a mensagem veio de um remetente específico. Trata-se de uma proteção de negação, por parte de uma das entidades envolvidas na comunicação, de ter participado de parte ou de toda a comunicação, propriedade conhecida como não-repúdio (ou irretratabilidade).

- **Disponibilidade**: disponibilidade refere-se à capacidade de acesso a informações e serviços sempre que necessário, ou seja, um sistema estará disponível se oferecer os serviços, de acordo com o projeto do sistema, sempre que os usuários os solicitarem.

Segundo Stallings (2008), ameaça pode ser definida como um potencial para violação da segurança quando há uma circunstância, capacidade, ação ou evento que pode quebrar

a segurança e causar danos. Ou seja, uma ameaça é um possível perigo que pode explorar uma vulnerabilidade.

O intruso de um sistema pode, muitas vezes, não só escutar o que se passa no canal de comunicação (intruso passivo), como também pode gravar mensagens e reproduzí-las mais tarde, injetar suas próprias mensagens ou modificar mensagens legítimas antes que elas cheguem ao receptor (intruso ativo) (Tanenbaum, 2003).

Os ataques passivos normalmente monitoram transmissões com o objetivo de obter informações que estejam sendo transmitidas. São muito difíceis de serem detectados, uma vez que não envolvem alteração de dados. Como exemplo de ataques passivos tem-se (Bishop, 2005) (Stallings, 2008):

- **Liberação do conteúdo da mensagem**: quando um intruso consegue ter acesso a mensagens transmitidas entre duas entidades.
- **Análise de tráfego**: utilizado para observar e detectar comportamentos específicos, como por exemplo, frequência com que duas entidades trocam mensagens e tamanho dessas mensagens. A utilidade destas informações poderia ser a descoberta da natureza da comunicação que estava ocorrendo.

Os ataques ativos envolvem algum tipo de modificação do fluxo de dados ou a criação de um fluxo falso. Como exemplo de ataques ativos tem-se (Bishop, 2005) (Stallings, 2008):

- **Disfarce**: ocorre quando uma determinada entidade tenta se passar por outra;
- **Repetição (ou repasse)**: trata-se da captura passiva de uma mensagem e sua posterior retransmissão para produzir um efeito não autorizado;
- **Modificação de mensagens**: consiste na alteração de parte de uma mensagem legítima ou no adiamento ou reordenação de mensagens para produzir um efeito não autorizado;
- **Negação de serviço (*Denial of Service* – DoS)**: impede ou inibe o uso ou gerenciamento normal das instalações de comunicação. Em outras palavras, é o impedimento do funcionamento normal de um servidor. Esta negação pode ocorrer na fonte, no local de destino ou ao longo do caminho intermediário.

É normal a existência de ataques a sistemas computacionais e evitar esses ataques não é tarefa trivial, uma vez que encontrar vulnerabilidades e corrigí-las pode ser bastante complexo. Quanto maior a complexidade de um sistema, mais vulnerabilidades de segurança podem ser encontradas e maior poderá ser o prejuízo financeiro caso esse sistema seja comprometido.

A arte de criar mensagens cifradas (criptografia) e a arte de solucioná-las (criptoanálise) são chamadas coletivamente de criptologia (Stallings, 2008).

3.3 Criptografia

Uma das mais frequentes formas de se implementar segurança em um sistema computacional é conhecida como criptografia. Em resumo, a criptografia pode ser entendida como um conjunto de métodos e técnicas para criptografar (cifrar ou codificar) informações legíveis por meio de um algoritmo de criptografia parametrizado por uma chave, convertendo um texto original, denominado texto aberto (texto claro ou texto simples), em um texto ilegível, denominado texto cifrado (cifra ou texto código). Posteriormente, é possível para o receptor decriptar este texto cifrado, ou seja, efetuar o processo reverso e recuperar as informações originais (Tanenbaum, 2003) (Moreno et al., 2005).

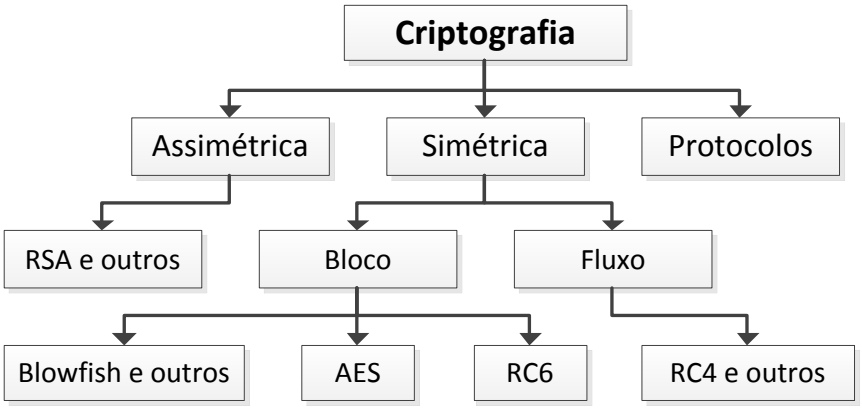


Figura 3.1: Classificações da criptografia (adaptado de Elminaam et al. (2010))

Normalmente, novos algoritmos são divulgados à comunidade à medida em que vão sendo desenvolvidos e o sigilo das informações é assegurado pela chave, a qual deve ser mantida em segredo e oferecida apenas às entidades pertinentes. Desta forma, o tamanho da chave é muito importante, já que quanto maior seu comprimento, mais segura torna-se a criptografia (Tanenbaum, 2003). Além disso, com base no tipo de chave, pode-se classificar a criptografia em chave simétrica ou assimétrica. A Figura 3.1 mostra as possíveis classificações da criptografia.

As criptografias simétrica e assimétrica são largamente utilizadas na construção de sistemas seguros. Apesar de algoritmos assimétricos exigirem de 100 a 1000 vezes mais processamento do que algoritmos simétricos, existem situações e aplicações em que a criptografia assimétrica possui vantagens e é conveniente utilizá-la (Coulouris et al., 2005). Estes dois mecanismos serão melhor explicados nas seções que seguem.

3.3.1 Criptografia Simétrica

A criptografia de chave simétrica (ou criptografia de chave privada) possui este nome porque os processos de criptografia e decriptografia são realizados com uma única chave, ou seja, tanto o emissor quanto o receptor detêm a mesma chave e esta deve ser mantida em segredo para que se possa garantir a confidencialidade das mensagens ou da comunicação. Exemplos de algoritmos classificados como simétricos são o DES (*Data Encryption Standard*) (NIST, 1993) e o AES (*Advanced Encryption Standard*) (NIST, 2001).

Como ilustrado na Figura 3.2, o texto legível é criptografado em texto cifrado pelo emissor utilizando uma chave secreta compartilhada. Após ser transmitida, a mensagem cifrada é então decriptografada pelo receptor utilizando a mesma chave secreta.

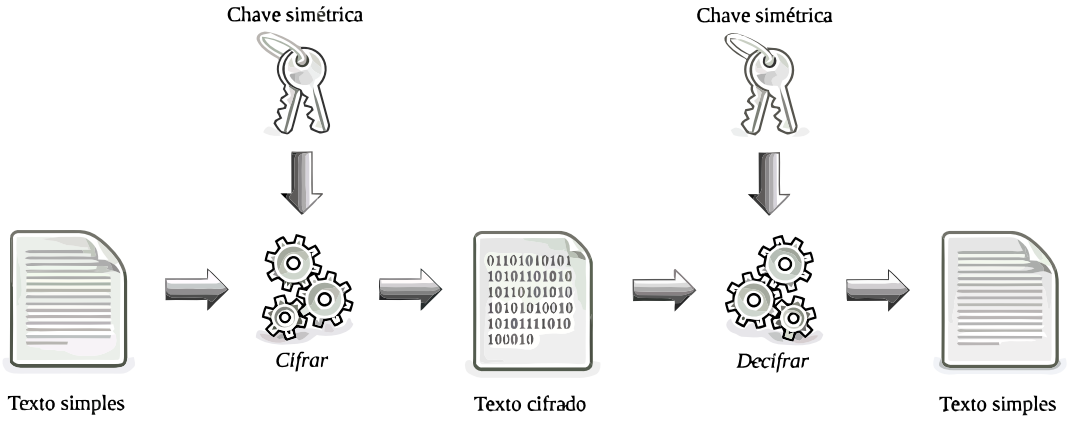


Figura 3.2: Processo da criptografia simétrica (Adaptado de Pires (2010))

A principal vantagem da criptografia de chave simétrica é que os algoritmos deste tipo são rápidos e podem operar em tamanhos arbitrários de mensagens. Em contrapartida, a desvantagem está na dificuldade de gerenciamento da chave compartilhada, a qual deve ser enviada de modo seguro a todos os usuários autorizados antes que as mensagens possam ser trocadas e ainda deve ser mantida em segredo (Moreno et al., 2005).

Os algoritmos simétricos podem ser classificados em duas categorias: de fluxo e de bloco. Os algoritmos de fluxo (*stream cyphers*) operam em fluxos de dados criptografando símbolo por símbolo, ou seja, um *bit* (ou *byte*) de cada vez. Por outro lado, os algoritmos de bloco (*block cyphers*) operam sobre blocos de tamanho fixo e pré-definido. Nos algoritmos de fluxo, uma chave de criptografia é utilizada para um símbolo apenas, enquanto nos de bloco, uma única chave é utilizada para todos os símbolos de um bloco (Just, 2012).

Exemplos amplamente utilizados de algoritmos de chave simétrica são o DES e o AES, como já mencionado anteriormente. Eles serão melhor explicados a seguir.

O DES foi selecionado pelo NIST (*National Institute of Standards and Technology*) em 1977 como FIPS PUB 46 (*Federal Information Processing Standard*) e reafirmado em 1983,

1988 e 1993 (NIST, 1993). Seu funcionamento envolve a codificação de blocos fixos de 64 *bits* usando uma chave de 56 *bits*. Após 16 iterações, o resultado é um novo bloco de 64 *bits* (Bishop, 2005; Stallings, 2008).

O AES é um algoritmo simétrico que foi selecionado em 2001 pelo NIST como o padrão FIPS-197 (NIST, 2001). Este algoritmo trabalha em blocos de dados de tamanho fixo de 128 *bits*, chamados de *State*, os quais são organizados como uma matriz de quatro linhas e quatro colunas de *bytes*. Os tamanhos das chaves podem ser três: 128 *bits*, 192 *bits* ou 256 *bits* (Stallings, 2008).

Como a maioria dos algoritmos simétricos, o AES aplica a mesma função criptográfica a cada bloco de uma entrada específica. As iterações aplicadas sobre cada bloco alteram a matriz *State* por meio de transformações não-lineares, lineares e baseadas em chave, sendo variável entre 10, 12 ou 14 o número de iterações de acordo com os parâmetros do algoritmo (tamanho da chave e do bloco de mensagem sendo, respectivamente, 16, 24 ou 32 *bytes*). Cada iteração altera o bloco de 128 *bits* para outro bloco de mesmo tamanho e cada *byte* da matriz *State* é afetado pelas seguintes transformações (Feldhofer et al., 2004; Hodjat et al., 2005):

1. *SubBytes* substitui cada *byte* da matriz *State* independentemente, de acordo com uma tabela de substituição. Esta operação é não-linear e muitas vezes chamada de operação *S-Box*.

2. *ShiftRows* altera a ordem de cada linha com uma série de deslocamentos. O número de *bytes* deslocados é diferente para cada linha.

3. *MixColumns* transforma as colunas da matriz *State* por meio da multiplicação por uma constante polinomial.

4. *AddRoundKey* combina a matriz *State* de 128 *bits* com uma chave de 128 *bits* por meio de operações do tipo XOR.

3.3.2 Criptografia Assimétrica

A criptografia assimétrica, mais conhecida como criptografia de chave pública, utiliza um par de chaves denominadas chave privada e chave pública. Qualquer uma das chaves pode ser utilizada para criptografar os dados, porém a mesma não pode ser utilizada para decriptografá-los, isto é, se a criptografia for realizada com a chave pública, somente a respectiva chave privada poderá realizar a decriptografia, ou vice-versa. Para que este tipo de criptografia obtenha sucesso é fundamental que a chave privada seja mantida em segredo, enquanto a chave pública pode, e deve, ser divulgada a outros usuários que desejam se comunicar (Stallings, 2008).

Na Figura 3.3 é possível visualizar como se dá o funcionamento da criptografia assimétrica. O emissor da mensagem utiliza a chave pública do receptor para criptografar o texto aberto em texto cifrado e, após o recebimento do texto cifrado resultante, o receptor utiliza sua chave privada para decriptografar o texto cifrado, obtendo novamente um texto legível.

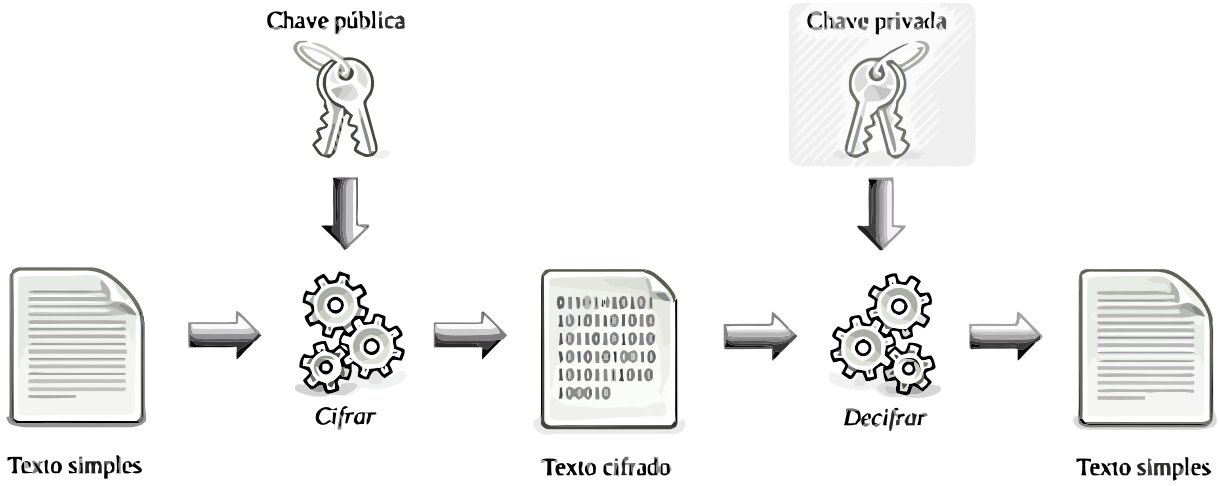


Figura 3.3: Processo da criptografia assimétrica (Adaptado de Pires (2010)

Devido ao fato de uma chave ser pública e a outra ser mantida em segredo, um criptossistema de chave pública deve atender às seguintes condições (Bishop, 2005):

1. Deve ser possível criptografar ou decriptografar uma mensagem dada a chave apropriada;
2. Deve ser computacionalmente inviável derivar a chave privada a partir da chave pública;
3. Deve ser computacionalmente inviável derivar a chave privada por meio de ataques do tipo Texto Puro Escolhido [1].

O algoritmo RSA (Rivest, Shamir, Adleman) tornou-se praticamente um sinônimo de criptografia de chave pública, embora existam muitos outros algoritmos potencialmente utilizáveis (Kurose et al., 2003). O RSA é considerado um algoritmo forte, porém sua principal desvantagem é a lentidão com que opera, uma vez que costuma-se aplicar chaves grandes para garantir uma segurança adequada. De acordo com Stapko (2007), uma das propriedades mais úteis do RSA é que ele pode ser usado tanto para a operação básica de chave pública (troca de mensagem entre duas entidades) quanto para autenticação (onde uma entidade criptografa uma mensagem com sua chave privada e a envia para alguém

[1]Texto Puro Escolhido: neste tipo de ataque, o criptoanalista tem a possibilidade de escolher o texto normal e conseguir seu texto cifrado correspondente.

que detenha sua chave pública, garantindo que a origem é realmente do dono da chave privada). Uma aplicação interessante para o RSA está no protocolo SSL (*Secure Sockets Layer*), que utiliza uma operação de chave pública e outra de autenticação.

A operação fundamental do algoritmo RSA é a exponenciação modular e sua segurança é garantida pela dificuldade na fatoração de números inteiros grandes. As chaves são geradas matematicamente utilizando dois números primos grandes. Mesmo conhecendo o produto deles, a segurança do algoritmo é avalizada pela complexidade em se fatorar esse produto e adquirir os valores secretos (Gura et al., 2004).

Outro exemplo de algoritmo assimétrico são os baseados em curvas elípticas. O ECC (*Elliptic Curve Cryptography*) pode garantir o mesmo nível de segurança que o RSA, porém utilizando chaves de tamanho menor. A chave do ECC pode ser mais compacta em relação ao RSA, o que pode trazer benefícios quando aplicado a sistemas embarcados, devido a uma menor exigência de memória e reduzido consumo de energia, entre outras vantagens (Zhang et al., 2010). O ECC será foco na parte experimental deste trabalho e, portanto, será melhor explicado na seção 4.2.3.

Segundo Coulouris et al. (2005), o uso de algoritmos de chave pública, apesar de menos eficiente devido ao elevado número de cálculos necessários, oferece a vantagem da conveniência para o tráfego de pequenas quantidades de informação, não sendo necessário efetuar uma troca prévia de chaves secretas.

3.4 Assinatura Digital

Para assinar papéis e documentos no mundo real, uma forma muito conhecida é o uso de uma assinatura, a qual afirma ou garante que um documento foi lido e consentido por uma pessoa física. O processo equivalente no mundo virtual é a assinatura digital, uma técnica criptográfica muito utilizada em documentos digitais.

Exatamente como acontece com as assinaturas por escrito, a assinatura digital deve ser verificável, não falsificável e incontestável. Isto é, deve ser possível provar que um documento assinado por um indivíduo foi na verdade assinado por ele (verificável) e que somente aquele indivíduo poderia ter assinado o documento (a assinatura não pode ser falsificada e o signatário não pode mais tarde repudiar o documento, nem negar que o assinou) (Kurose et al., 2003). Normalmente, algoritmos de chave pública são empregados para alcançar soluções de assinatura digital.

Sempre que o uso de criptografia faz parte de um processo de troca de informações, torna-se necessário considerar a criticidade da transmissão, isto é, se houver atrasos na troca de mensagens, por exemplo, haverá um comprometimento do sistema? Como existem vários casos em que isto poderia tornar o sistema inútil ou atrasá-lo consideravel-

mente, adota-se o uso de resumos de mensagem. Isto implica que ao invés de criptografar a mensagem por inteiro, aplica-se assinatura digital apenas no cabeçalho da mensagem.

Um resumo de mensagem é bastante similar a uma soma de verificação. Algoritmos de resumo de mensagens pegam uma mensagem *m*, de comprimento arbitrário, e calculam uma "impressão digital" dos dados com comprimento fixo, conhecida como resumo de mensagem *H(m)*. O resumo de mensagem protege os dados, uma vez que, se *m* for modificado para *m'*, então a mensagem *H(m)* processada para os dados originais (e transmitida com os dados) não combinará com *H(m')* processada sobre os dados modificados, *m'*. (Kurose et al., 2003). Desta forma, apenas parte da mensagem é utilizada para aplicar a assinatura, tornando-se necessário criptografar uma quantidade menor de informação.

Funções comuns em técnicas de resumo de mensagem, verificação de redundância cíclica e somas de verificação são as funções de *hash*. Elas são baseadas na execução de funções de compressão do tipo *one-way* sobre um bloco de dados de qualquer tamanho, resultando em uma saída de tamanho fixo (Çayirci e Rong, 2009).

Este procedimento ocorre em diversas iterações e o tamanho do bloco de saída determina o nível de segurança provida por funções de *hash*. Uma função de *hash* deve ser facilmente computável e publicamente conhecida, permitindo praticidade nas implementações de *hardware* e *software* (Çayirci e Rong, 2009).

3.5 Segurança aplicada a Sistemas Embarcados Críticos

Segurança na comunicação, como já discutido, é uma exigência de um número cada vez maior de aplicações que executam em sistemas embarcados, indo desde sistemas de pequeno porte como PDAs, fones de ouvido sem fio e sensores até sistemas de maior porte como roteadores, *gateways*, *firewalls*, servidores de armazenamento, servidores *Web* e veículos não tripulados. Da mesma forma que este tipo de dispositivos experimentou avanços tecnológicos de grande impacto, houve, consequentemente, uma sofisticação dos ataques à segurança. Pode-se ainda observar que o custo da falta de segurança em sistemas computacionais pode ser muito alto (Kocher et al., 2004).

Com vista aos principais problemas e necessidades da área de segurança da informação no contexto de sistemas embarcados críticos, uma busca na literatura por trabalhos relacionados a este foi conduzida para melhor visualizar o estado da arte. Eles serão melhor detalhados a seguir.

3.5.1 Trabalhos Correlatos

Existem muitos trabalhos de investigação na área de segurança da informação e de segurança específica para sistemas embarcados críticos. O assunto é foco de pesquisas há

anos e alguns resultados interessantes podem ser vistos em publicações de importantes eventos. Em Ertaul e Lu (2005) existe a proposta do emprego de criptografia de curva elíptica para transferência de dados e compartilhamento seguro de chaves em redes móveis do tipo *ad-hoc*. São utilizados sete diferentes mecanismos do algoritmo, entre eles o *El-Gamal* e o *Diffie-Hellman*. O trabalho compara os resultados obtidos pelo ECC com o RSA e conclui que, para um mesmo nível de resistência contra os mais conhecidos tipos de ataques, um sistema baseado em curvas elípticas pode trabalhar com chaves muito menores. Os estudos foram realizados com dispositivos móveis conectados em redes *ad-hoc*, porém sem a existência de movimento durante os testes, ou seja, apesar de serem dispositivos móveis, as condições durante os testes eram fixas.

Em Ramachandran et al. (2007), os autores utilizam *Pocket* PCs e sensores como ambientes de testes de variações de algoritmos de curva elíptica para desenvolver protocolos de comunicação eficientes. Os testes de desempenho realizados buscam estudar a habilidade computacional destes dispositivos em processar funções criptográficas. Dentre os resultados obtidos, um importante ponto ressaltado pelos autores é a necessidade de otimização de algumas funções matemáticas, especialmente relacionadas a multiplicações. Outra consideração importante é a dificuldade de se realizar operações computacionais muito intensas em sensores, devido às fortes limitações de recursos apresentadas por estes dispositivos.

O trabalho de Potlapally et al. (2003) faz uma avaliação do impacto de protocolos de comunicação no consumo de energia em dispositivos com recursos limitados. O ambiente de testes consiste de um iPAQ PDA conectado a uma rede local via *Wi-Fi* e rodando o sistema operacional Linux. Pelas conclusões do trabalho, os algoritmos assimétricos são os que geram um maior gasto de energia, seguidos dos algoritmos simétricos e, em terceiro, pelos algoritmos *hash*. O consumo de energia dos assimétricos é muito dependente do tamanho da chave, enquanto que nos simétricos o tamanho da chave não é fator determinante.

Em AL-Rousan et al. (2009) é apresentado um algoritmo de segurança para troca de informações entre sensores sem fio com baixo consumo de energia. Trata-se de um esquema de segurança que utiliza um algoritmo *hash one-way* para troca de informações associado a um algoritmo de criptografia simétrica. De acordo com os testes realizados, há baixo consumo de energia e garantia dos principais requisitos de segurança (integridade, confidencialidade e autenticidade).

Outro trabalho que aborda o consumo de energia é Minaam et al. (2010), que efetua testes especificamente com algoritmos simétricos processando diferentes tipos de dados. Os algoritmos utilizados para os testes são: AES, DES, 3DES (*Triple DES*), RC2 (*Ron's Code ou Rivest Cipher 2*), *Blowfish* e RC6 (*Ron's Code ou Rivest Cipher 6*). Entre algumas conclusões obtidas com o trabalho, pode-se perceber que o *Blowfish* é o que apresenta

melhor desempenho, seguido do RC6; o 3DES ainda tem um desempenho inferior ao DES; e o RC2 é o que mais consome energia.

Apesar desses trabalhos apresentarem conceitos envolvendo criptografia em sistemas embarcados, eles estão focados basicamente no consumo de energia. Diferentemente, o foco deste trabalho está em assegurar a informação transmitida entre os dispositivos envolvidos nela de modo eficiente e obter dados que ajudem a verificar o impacto do tamanho da chave nos resultados finais em um protótipo de sistema embarcado que será apresentado no Capítulo 4. Uma revisão sobre segurança para sistemas embarcados apresentada por Wollinger et al. (2003) mostra a importância cada vez maior destes dispositivos e a necessidade de investigação e implementação de segurança, principalmente de criptografia otimizada para tais dispositivos.

O artigo de revisão Short (2008) elenca fatores que devem ser considerados durante o projeto e desenvolvimento de sistemas embarcados sensíveis a tempo real, classificando-os em requisitos de ambiente, *hardware*, *software* e de comunicação. Porém, este artigo não aborda especificamente requisitos de segurança para tais sistemas, o que é citado por outros trabalhos como um fator importante de se considerar desde o projeto de um sistema embarcado até sua concepção. É o caso do trabalho apresentado em Armoush et al. (2009) que, tendo em vista requisitos de segurança, busca estabelecer *design patterns* para fornecer soluções abstratas a problemas comumente encontrados enquanto projetando um sistema embarcado. Para desenvolver sistemas críticos, faz-se necessário integrar os já consolidados métodos de projeto utilizados em *hardware* e *software*, sem esquecer de considerar requisitos de segurança e processos que garantam a qualidade dos módulos desenvolvidos.

Existe um histórico de fatores que vêm exigindo que segurança seja um requisito de projeto e que seja implementada de maneira eficiente, dentre eles Kocher et al. (2004): (a) o aumento crescente de ataques de vários tipos, o que exige que os sistemas sejam seguros mesmo com a possibilidade de serem física ou logicamente acessados; (b) recursos de processamento são facilmente extrapolados devido à demanda de algoritmos criptográficos; (c) sistemas movidos por baterias têm muitas limitações de recursos (processamento, memória, armazenamento); (d) devem possuir arquiteturas flexíveis o suficiente para que seja possível efetuar alterações para novos mecanismos e padrões criptográficos; (e) novos objetivos de segurança exigem maior interação entre desenvolvedores de sistemas embarcados e estudiosos de segurança.

Paralelamente a estas preocupações inerentes ao desenvolvimento de sistemas embarcados críticos seguros, existe a necessidade de se avaliar cada sistema isoladamente, contemplando características mais específicas. O trabalho apresentado por Ravi et al. (2004) discute a falta de técnicas de segurança específicas para sistemas embarcados, discutindo requisitos de segurança, limitações apresentadas por estes dispositivos, questões específicas de rede, algoritmos mais apropriados, principais ataques à segurança e melhorias na autonomia de bateria.

Estudos direcionados têm sido mais frequentes em trabalhos mais recentes. Com o surgimento das redes veiculares (ou VANETs – *Vehicular Ad-hoc Networks*), pesquisas em segurança que contemplam as particularidades deste cenário têm sido conduzidas. As VANETs oferecem uma variedade de aplicações que levam em conta, por exemplo, a assistência ao condutor, a propagação de informações turísticas, localização de postos de gasolina e cobrança de pedágio automatizado. As VANETs também podem ser aplicadas na entrega de entretenimento, por exemplo, na implementação de um sistema de compartilhamento de vídeo entre os veículos e aplicações para a segurança no trânsito, prevenção de acidentes e congestionamentos. Além disso, há a possibilidade de acompanhamento, em tempo real, dos veículos, tornando-se uma solução para situações de sequestro (Yousefi et al., 2006).

Um dos principais desafios técnicos enfrentados por VANETs é a inexistência de uma entidade central para gerenciar a comunicação. Sendo assim, um canal único de comunicação precisa ser escolhido, permitindo que veículos encontrados nos mais diversos pontos de uma rodovia possam enviar informações importantes para os demais veículos que trafegam nela. Estas características geram preocupações com privacidade e autenticidade das informações, uma vez que entidades maliciosas podem, por exemplo, tentar desviar outros veículos para rodovias congestionadas em seu próprio benefício. Na pesquisa publicada em Hartenstein e Laberteaux (2008), apresenta-se a Criptografia de Curva Elíptica como uma opção de autenticação que pode ser utilizada em VANETs devido ao seu baixo consumo de recursos computacionais em relação a outros algoritmos de criptografia assimétrica.

Muitos trabalhos na área propõem uma abordagem associativa, integrando algoritmos de chaves pública e privada, fornecendo assim confidencialidade, autenticidade, integridade e performance para suas aplicações. No entanto, o desempenho de algoritmos de curvas elípticas tem se mostrado viável para aplicação em não apenas tarefas básicas de criptografia de chave pública, mas também na criptografia de dados propriamente ditos, uma ação normalmente realizada por algoritmos simétricos. Jena e Jena (2011) propõem um sistema de encriptação eficiente para criptografar mensagens longas. A diferença está nas operações matemáticas executadas pelo algoritmo, que reduzem significativamente a complexidade das operações, permitindo a sua utilização em sistemas com limitações, como sistemas embarcados.

O trabalho de Peng e Fang (2010) compara as implementações de algoritmos de chave pública em cartões inteligentes. Os algoritmos selecionados são RSA e ECC. Apenas o último foi implementado e executado em um processador Intel 8051. O tempo necessário para a geração da chave foi de 5,2 segundos, para a criptografia dos dados foi de 21,3 segundos e para a decriptografia de 17,1 segundos. Estes valores, de acordo com os autores, são baixos e aceitáveis para a arquitetura tratada.

Algumas modificações nos sistemas de criptografia que tradicionalmente utilizam algoritmos assimétricos pode configurar uma abordagem vantajosa. No caso de sistemas

embarcados, as limitações de armazenamento de memória e memória *cache* são fatores que devem ser considerados na execução de algoritmos criptográficos. O ECC pode ser aplicado a cenários com estes tipos de limitações, porque, como o tamanho da chave utilizada é consideravelmente menor do que a do RSA, o consumo de recursos será correspondentemente reduzido. No trabalho conduzido por Habib et al. (2009) o RSA foi substituído pelo ECC para reduzir o consumo de recursos de um esquema de segurança específico para WiMAX e melhorar seu desempenho, o que mostra uma maior eficiência do algoritmo ECC.

De modo a verificar o impacto do uso de algoritmos criptográficos e da variação no tamanho de chaves em sistemas limitados como os sistemas embarcados críticos, a proposta deste trabalho é a implementação de um protótipo que contemple características similares às de um sistema real, que sirva como ambiente de experimentações para executar avaliações de desempenho de algoritmos de criptografia mais comuns na literatura. Deste modo, após uma avaliação dos principais trabalhos existentes, uma arquitetura alvo foi selecionada. Trata-se do *kit* Gumstix Overo EVM, que será melhor abordado no Capítulo 4. De modo geral, o *kit* já tem aplicações práticas publicadas na literatura, inclusive no contexto de VANTs, sendo viável sua aplicação devido ao peso reduzido e ao processamento que atende às necessidades básicas de comunicação neste cenário (Phang et al., 2010; Martinez et al., 2010).

Compõe a proposta também a preparação do ambiente com um sistema operacional que seja portável para a arquitetura em questão, de modo a obter uma relação de equilíbrio entre desempenho e segurança durante as avaliações que serão apresentadas em forma de estudos de caso no Capítulo 5. Os resultados pretendidos neste trabalho vêm ao encontro do que os autores preconizam em Januzaj et al. (2010), abordando como foco central a segurança da comunicação em dispositivos aéreos e terrestres visando reduzir custos e ciclos de produção e aumentar a robustez e a segurança nestes dispositivos.

3.6 Considerações Finais

Este capítulo apresentou os principais conceitos de segurança em redes de computadores, caracterizando criptografia simétrica e assimétrica e assinatura digital. Apresentou também alguns exemplos de algoritmos mais aplicados e algumas de suas características.

Ficou evidenciado que o processo criptográfico apesar de bem conhecido e existente há muito tempo, ainda precisa de incrementos. Novas técnicas e a utilização de criptografia em novos sistemas e até mesmo a melhoria desse uso em ambientes e plataformas já conhecidas constituem ainda elementos de pesquisa e investigação da sociedade científica e industrial.

Tendo em mente esses conceitos de segurança e as reais necessidades do estudo de criptografia e algoritmos criptográficos em diferentes ambientes, mais especificamente em sistemas embarcados críticos, que desenvolveu-se o ambiente de experimentos apresentado e detalhado no próximo capítulo, tendo em vista a realização de experimentos que serão apresentados no Capítulo 5.

Capítulo

4

Plataforma de Experimentações e Requisitos de Avaliação de Desempenho

4.1 Considerações Iniciais

Este trabalho está fortemente inserido no contexto de sistemas embarcados críticos, como já mencionado. Devido a isso, a avaliação de algoritmos de criptografia precisa adotar um ambiente de características similares a um sistema deste tipo, podendo sofrer as reais influências da limitação de recursos, como processamento, memória e energia.

Este capítulo apresenta o ambiente escolhido para realização de experimentos, os algoritmos e bibliotecas utilizados nos testes e explica as razões pelas quais estes elementos foram escolhidos. Além disso, uma contextualização sobre estatística e avaliação de desempenho de sistemas computacionais é apresentada de modo a introduzir conceitos que serão aplicados no Capítulo 5.

4.2 Plataforma de Testes e Experimentações

Para uma melhor avaliação de resultados, um ambiente de testes e experimentações foi definido de modo a obter um protótipo semelhante ao de um sistema embarcado real. Sendo assim, esta seção apresenta o *hardware* e o sistema operacional escolhidos e os algoritmos de criptografia usados nas avaliações.

4.2.1 Hardware

O ambiente de experimentações adotado foi um *kit* desenvolvido pela empresa Gumstix chamado Overo EVM (ver Figura 4.1). A escolha pelo *kit* se deve ao fato de já existirem exemplos de aplicações práticas, inclusive no contexto de VANTs, sendo viável sua aplicação devido ao peso reduzido e ao processamento que atende às necessidades básicas de comunicação (Phang et al., 2010; Martinez et al., 2010). É importante ressaltar que todo caso deve ser avaliado particularmente e que este trabalho adota este *kit* como ambiente de experimentações para poder obter resultados reais em lugar de resultados simulados que poderiam não considerar todos os fatores passíveis de exercerem influências nos resultados.

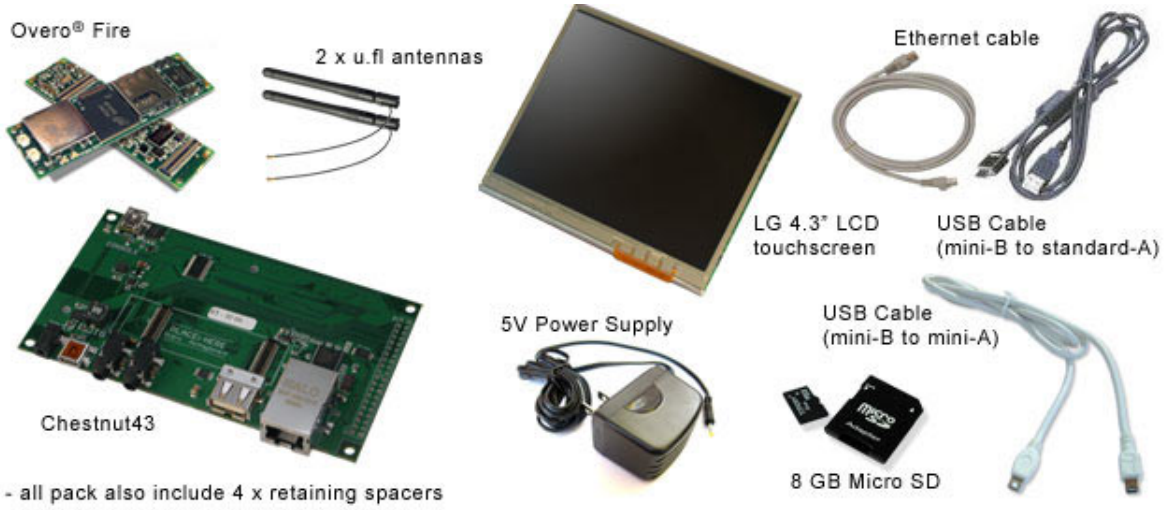


Figura 4.1: *Kit* Gumstix Overo EVM (Gumstix, 2012)

O *kit* é composto por uma placa Overo Fire COM (*computer on-module*) com processador ARM Cortex-A8 OMAP3530 (720 MHz), memória RAM de 256MB e suporte a cartão de memória *MicroSD* e conexões *Bluetooth* e *Wireless*. Acompanha uma placa de expansão *Chestnut43* com conexões *Ethernet*, USB e console serial via mini-USB. Um monitor LCD *touchscreen* de 4.3" também integra o *kit*.

4.2.2 Sistema Operacional

O *kit* Gumstix Overo EVM, de acordo com a documentação disponível na página do fabricante (Gumstix, 2011b), foi desenvolvido para operar, dentre outros, com os sistemas operacionais Windows Mobile, Linux ARM e Android. Devido à adoção do *toolkit* OpenSSL (que será abordado na seção 4.2.3), o sistema operacional escolhido foi o Linux ARM, mais especificamente a distribuição Ubuntu 10.04 LTS (*Long Term Support*).

A documentação oficial da Gumstix inclui informações de como proceder para gerar uma imagem otimizada do sistema operacional (exclusiva para Ubuntu) por meio da fer-

ramenta RootStock, como particionar o cartão e como conectar-se ao *hardware* via console serial (Gumstix, 2011a,b,c,d; Ubuntu, 2011). Porém, esta documentação é falha em alguns aspectos, por isso, uma adaptação da mesma foi necessária durante a fase de preparação do ambiente de experimentações. Os principais aspectos para que seja possível reproduzir o ambiente de experimentos deste trabalho são apresentados no Apêndice A.

4.2.3 Algoritmos Abordados

O uso de comunicação sem fio em sistemas embarcados críticos aumenta a necessidade de garantir a confidencialidade, a integridade e a autenticidade das informações transmitidas (Wollinger et al., 2003), o que exige que esforços sejam concentrados para garantir que o envio da informação seja seguro. Considerando-se esta necessidade, a escolha de algoritmos apropriados passa a consistir um desafio, uma vez que para assegurar requisitos de segurança no contexto de um sistema embarcado haverá um consequente aumento no uso de recursos computacionais (Fan et al., 2008), o que consiste um fator crítico devido à limitação de recursos deste tipo de sistema, como já mencionado.

Frente a isso, um levantamento dos principais algoritmos utilizados na literatura foi realizado. Destacam-se os algoritmos simétricos DES, 3DES, AES, *Blowfish* e RC2 e o assimétrico RSA (Garfinkel et al., 2003; Nadeem e Javed, 2005; Umaparvathi e Varughese, 2010). Todos estes algoritmos fazem parte da Crypto (OpenSSL, 2012a), uma biblioteca criptográfica do OpenSSL (OpenSSL, 2012b) que inclui também alguns outros algoritmos.

O Projeto OpenSSL é um esforço colaborativo para desenvolver um *toolkit* robusto, de nível comercial, recursado e *open source* que implementa os protocolos *Secure Sockets Layer* (SSL v2/v3) e *Transport Layer Security* (TLS v1), assim como uma biblioteca criptográfica completa de uso genérico (OpenSSL, 2012b). A adoção do *toolkit* OpenSSL se dá pela ampla aplicação em trabalhos encontrados na literatura, como em Potlapally et al. (2003) e em Ravi et al. (2004), que efetuam comparações de desempenho entre algoritmos implementados pela biblioteca Crypto (OpenSSL, 2012a). Outro elemento determinante é o fato de os algoritmos já estarem validados, uma vez que são públicos e já foram submetidos a testes em diversas situações. Normalmente, novos algoritmos são divulgados à comunidade à medida em que vão sendo desenvolvidos, sendo esta a forma mais clássica de se validar um algoritmo de criptografia (Tanenbaum, 2003).

Dois algoritmos baseados em curvas elípticas foram implementados no trabalho de conclusão de curso de Silva (2011). Eles possuem parâmetros que definem a curva elíptica e o ponto utilizado em comum pelos usuários definidos como fixos. Ambos também possuem uma estrutura para a definição das chaves públicas e privadas, assim como funções para a criação das chaves, para a criptografia da mensagem e para a decriptografia.

As implementações foram baseadas no funcionamento do algoritmo de criptografia de curvas elípticas e na documentação das bibliotecas. Para cada algoritmo foram produzi-

dos dois códigos em C para representar duas possibilidades de curvas elípticas: uma com chave de 160 *bits* com os parâmetros especificados pelo SECG (Certicom Corp., 2000), e outra com chave de 256 *bits* especificada pelo NIST (NIST, 2009). Cada algoritmo possui peculiaridades diferentes de acordo com o uso de cada biblioteca e estão detalhados tecnicamente no Apêndice B.

Para melhor planejar e interpretar os experimentos realizados neste trabalho, técnicas de avaliação de desempenho serão adotadas durante o desenvolvimento dos estudos de caso apresentados no Capítulo 5. Estas técnicas serão melhor explicadas a seguir.

4.3 Técnicas de Avaliação de Desempenho em Sistemas Computacionais

Para planejar experimentos e avaliar resultados, técnicas de avaliação de desempenho em sistemas computacionais propostas por Jain (1991) foram adotadas. Particularmente, considerando-se sistemas críticos, em que questões de confiabilidade são fundamentais, o problema de desempenho é ainda mais importante face ao grande desafio existente em manter um equilíbrio aceitável entre confiabilidade e desempenho. Desempenho e confiabilidade são, de modo geral, considerações antagônicas e é exatamente esse ponto que faz com que o projeto de sistemas computacionais críticos apresente vários desafios.

Usuários de computadores, administradores e projetistas demonstram interesse em avaliação de desempenho, uma vez que o objetivo da técnica é obter resultados que possibilitem atingir ou prover a maior performance possível com o menor custo. A avaliação de desempenho pode ser útil para determinar a melhor configuração de um determinado cenário de *software* e/ou *hardware* associados, permitindo aferir qual deles é mais eficiente na solução de um problema e o quão melhor ele é em relação às demais opções disponíveis.

Segundo Jain (1991) os passos necessários para se realizar uma avaliação de desempenho sistemática são:

1. **Definir os objetivos e o sistema**: o primeiro passo em qualquer avaliação de desempenho é definir os objetivos do estudo e o que constitui o sistema, estabelecendo suas fronteiras.

2. **Listar serviços e saídas**: quando um usuário solicita qualquer um desses serviços, existe uma série de saídas possíveis. Algumas dessas saídas são desejáveis e outras não, portanto é necessário saber quais variáveis de resposta serão monitoradas.

3. **Selecionar métricas**: o próximo passo é selecionar os critérios para comparar a performance. Esses critérios são chamados métricas. Em geral, as métricas são relacionadas com velocidade, precisão e disponibilidade dos serviços.

4. **Listar parâmetros**: trata-se dos parâmetros que afetam a performance.

5. **Selecionar fatores para estudo**: a lista de parâmetros pode ser dividida em duas partes: aqueles que serão variados durante a avaliação e aqueles que permanecerão constantes. Os parâmetros que serão variados são denominados fatores e seus valores são denominados níveis.

6. **Selecionar a técnica de avaliação**: as três abordagens para avaliação de desempenho são: modelagem analítica, simulação e aferições em sistemas reais. A seleção da técnica correta depende do tempo e recursos disponíveis para resolver o problema e o nível desejado de precisão.

7. **Selecionar a carga de trabalho**: a carga de trabalho consiste em uma lista de requisições de serviços ao sistema, que busca ser semelhante à carga de trabalho normalmente executada por um sistema do tipo avaliado.

8. **Projeto de experimentos**: define-se uma sequência de experimentos que forneça o máximo de informação com o mínimo de execuções.

9. **Analisar e interpretar dados**: é importante avaliar os resultados obtidos e conhecer bem o sistema avaliado, podendo identificar discrepâncias nos testes executados que poderão interferir nas conclusões da avaliação.

10. **Apresentar resultados**: o passo final para todos os projetos de desempenho é apresentar os resultados de modo que seja fácil o entendimento. Isso geralmente necessita que os resultados sejam apresentados em forma de gráficos.

Uma carga de trabalho pode ser real ou sintética. Real é aquela observada em um sistema sendo usado para operações normais. Ela não pode ser repetida, e portanto, não é recomendada para o uso como carga de trabalho de testes. Em vez disso, uma carga de trabalho sintética, cujas características são semelhantes às da carga de trabalho real e pode ser aplicada várias vezes de forma controlada, é desenvolvida e utilizada para estudos (Jain, 1991).

O objetivo de um projeto de experimentos adequado é obter o máximo de informação com o mínimo de experimentos. Isso economiza um trabalho considerável que teria sido gasto com a coleta dos dados. A análise adequada de experimentos também ajuda a separar os efeitos de vários fatores que podem afetar o desempenho. Além disso, permite determinar se um fator tem um efeito significativo ou se a diferença observada é simplesmente devido a variações aleatórias causadas por erros de medição e parâmetros que não foram controlados (Jain, 1991).

Existem alguns termos utilizados durante a etapa de projeto e análise de experimentos, dentre eles (Jain, 1991):

- **Variável de resposta**: é a saída de um experimento, normalmente a métrica utilizada para avaliar o desempenho do sistema;
- **Fatores**: são as variáveis que afetam a variável de resposta do sistema;
- **Níveis**: são os valores que um determinado fator pode assumir;
- **Interação**: indica a dependência entre os fatores avaliados.

Os três modelos de projeto de experimentos mais utilizados são: projeto simples, fatorial completo e fatorial parcial. Suas vantagens e desvantagens serão explicadas a seguir (Jain, 1991):

- **Fatorial simples**: em um projeto fatorial simples, começa-se com uma configuração típica e varia-se um fator de cada vez para ver como esse fator afeta o desempenho. Esse tipo de projeto é o mais fácil de se utilizar, porém não é estatisticamente eficiente;
- **Fatorial completo**: um planejamento fatorial completo utiliza todas as combinações possíveis em todos os níveis de todos os fatores. A vantagem de um planejamento fatorial completo é que todas as combinações possíveis e cargas de trabalho são examinadas;
- **Fatorial parcial**: algumas vezes, o número de experimentos necessários para um planejamento fatorial completo é muito grande. Nesses casos, pode-se usar apenas uma fração do planejamento fatorial completo.

Descrever o comportamento de um sistema utilizando apenas a média, não é suficiente em determinadas situações. A média $\bar{x}$ não apresenta nenhuma informação sobre o espalhamento dos dados, isto é, o quão distante as amostras estão do valor médio. A média de dois conjuntos $A = \{5, 10, 15\}$ e $B = \{0, 10, 20\}$ é exatamente a mesma, (10), porém, o espalhamento dos dados em cada conjunto é diferente.

O desvio padrão e a variância são parâmetros que medem a dispersão de um conjunto de dados em relação à média. A variância, representada por σ^2, é definida como o desvio quadrático médio da média e é calculada sobre uma amostra de dados partindo da Equação 4.1, onde *n* corresponde ao número de elementos da amostra coletada e $(x_i - \bar{x})^2$ corresponde ao quadrado da distância entre uma amostra x_i e a média da amostra $\bar{x}$. O quadrado da diferença é utilizado para computar o valor absoluto da distância.

$$\sigma^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1} \tag{4.1}$$

O desvio padrão, representado por σ, refere-se à raiz quadrada da variância (Equação 4.2). Tem a mesma função da variância, porém apresenta a vantagem de permitir uma interpretação direta da variação da amostra de dados, pois o desvio padrão é expresso na mesma unidade de medida dos dados.

$$\sigma = \sqrt{variancia} = \sqrt{\sigma^2} \tag{4.2}$$

4.3.1 Intervalo de Confiança

Para encontrar uma estimativa perfeita para a média seria necessário utilizar um número infinito de amostras. Como isso não é possível, adota-se a obtenção de limites probabilísticos (intervalo de confiança), c_1 e c_2, de modo que a média exata $\bar{x}$ pertença ao intervalo $[c_1, c_2]$, com uma certa probabilidade $(1-\alpha)$ de acerto. A Equação 4.3 representa a probabilidade de um intervalo de confiança estar correto, onde α corresponde ao nível de significância e $1-\alpha$ corresponde ao coeficiente de confiança.

$$P(c_1 \leq \bar{x} \leq c_2) = 1 - \alpha \tag{4.3}$$

De modo geral, o intervalo de confiança é explicitado em um percentual próximo a 100%, por exemplo, 90% ou 95%. Já o nível de significância α é explicitado como fração e é usualmente próximo de zero, por exemplo, 0,05 ou 0,1 (Kamienski et al., 2002).

4.3.2 2^K Fatorial Completo

O fatorial completo descreve a influência de cada um dos fatores elencados com relação a um parâmetro de resposta, de acordo com os níveis escolhidos. É calculado com base em um modelo de regressão linear da forma como mostra a Equação 4.4 (exemplo com 2 fatores, ou $K = 2$).

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B \tag{4.4}$$

Onde y é a variável de resposta e x_w (w é o fator, nesse caso A e B) pode assumir os valores **-1** e **1** que representa cada um dos 2 níveis escolhidos. Combinando os níveis de cada fator, obtêm-se $2^{k=2}$ cenários:

$$\begin{aligned} y_1 &= q_0 - q_A - q_B + q_{AB} \\ y_2 &= q_0 + q_A - q_B - q_{AB} \\ y_3 &= q_0 - q_A + q_B - q_{AB} \\ y_4 &= q_0 + q_A + q_B + q_{AB} \end{aligned} \tag{4.5}$$

Deste modo, pode-se obter os valores de q_w em relação à variável de resposta de cada um dos cenários:

$$\begin{aligned} q_0 &= \tfrac{1}{4}(y_1 + y_2 + y_3 + y_4) \\ q_A &= \tfrac{1}{4}(-y_1 + y_2 - y_3 + y_4) \\ q_B &= \tfrac{1}{4}(-y_1 - y_2 + y_3 + y_4) \\ q_{AB} &= \tfrac{1}{4}(y_1 - y_2 - y_3 + y_4) \end{aligned} \tag{4.6}$$

A importância de um fator é medida pela proporção na variação total da variável de resposta que é influenciada por esse fator. Se um fator é responsável por 95% e outro fator por 5% da variação total, então o segundo fator pode ser considerado não importante em muitas situações práticas. O dividendo da Equação 4.1 é chamado de variação total de y ou soma dos quadrados total (SST [1]), como mostra a Equação 4.7, onde K é a quantidade de fatores, y_i é a variável de resposta para o experimento i e $\bar{y}$ é a média da variável de resposta de todos os experimentos.

$$\text{Variação total de } y = \text{SST} = \sum_{i=1}^{2^K} (y_i - \bar{y})^2 \tag{4.7}$$

Para $k = 2$, a variação pode ser dividida em três partes, segundo a Equação 4.8.

$$SST = 2^2 q_A^2 + 2^2 q_B^2 + 2^2 q_{AB}^2 \tag{4.8}$$

As três partes do lado direito da equação representam a porção da variação total explicada pelos fatores A, B e pela interação AB, respectivamente. Então, $2^2 q_A^2$ é a porção de SST explicada pelo fator A e é denotado por SSA. Da mesma forma, SSB é $2^2 q_B^2$ e SSAB é $2^2 q_{AB}^2$. Assim temos a Equação 4.9.

$$SST = SSA + SSB + SSAB \tag{4.9}$$

A fração explicada pelo fator A pode ser representada pela Equação 4.10. O mesmo se aplica ao fator B e à interação AB.

$$\text{Fração da variação explicada por } A = \frac{SSA}{SST} \tag{4.10}$$

Quando expressada em porcentagem, essa fração mede a importância do fator A. Os fatores com alta porcentagem de variação são considerados importantes.

Todos os experimentos apresentados nas próximas seções foram cuidadosamente planejados seguindo um padrão estatístico rigoroso, visando à geração de dados para análise que culminam com as informações sobre desempenho necessárias ao sistema (Jain, 1991;

[1] SST: *Sum of Squares Total*

Montgomery et al., 2002). Tendo conhecimento do ambiente de experimentações adotado neste trabalho e os algoritmos de criptografia utilizados, as próximas seções apresentarão detalhes do planejamento de experimentos e uma ampla discussão dos resultados será conduzida no domínio de cada estudo de caso, isoladamente.

4.4 Considerações Finais

Este capítulo apresentou o ambiente de experimentações adotado para este trabalho e também a implementação do mesmo, considerando recomendações do fabricante e necessidades específicas do contexto no qual este trabalho está focado. Foram apresentados também os algoritmos que serão avaliados no próximo capítulo, a razão pela escolha dos mesmos e detalhes de implementação de dois algoritmos de curva elíptica.

Este capítulo apresentou ainda técnicas estatísticas de avaliação de desempenho que configuram a metodologia de avaliação adotada para os testes conduzidos no próximo capítulo. Elas serão aplicadas na avaliação dos algoritmos quando executados no ambiente de experimentações supra abordado.

Capítulo

5

Avaliação de Desempenho de Algoritmos Criptográficos em Sistemas Embarcados

5.1 Considerações Iniciais

Com o ambiente de experimentações implementado e tendo conhecimento de algumas técnicas estatísticas para avaliar e comparar desempenho em sistemas computacionais detalhados no Capítulo 4, foi possível elaborar alguns estudos de caso para melhor avaliar os resultados obtidos neste trabalho de mestrado.

Os estudos de caso apresentados a seguir foram conduzidos de modo a efetuar uma avaliação ampla dos algoritmos abordados e também citados no Capítulo 4. Além disso, avaliações específicas como comparações de desempenho entre algoritmos de criptografia que pudessem retornar resultados relevantes à pesquisa foram executadas, com vista à obtenção de dados que comprovem as vantagens de alguns algoritmos em relação a outros.

O primeiro estudo de caso avalia oito algoritmos criptográficos simétricos quanto a tempo e *throughput* médios de execução. O segundo e o terceiro estudos de caso aprofundam os testes para os algoritmos AES e DES, respectivamente, efetuando uma avaliação de desempenho com a variação de alguns fatores relevantes para cada caso. O quarto estudo de caso avalia o algoritmo criptográfico assimétrico de curva elíptica. E o quinto estudo de caso apresenta uma avaliação de desempenho considerando o processo de comunicação entre duas entidades somado ao processo de criptografia.

5.2 Estudo de Caso 1: Experimentos com Algoritmos Simétricos

Esta seção tem como objetivo apresentar os experimentos realizados com algoritmos simétricos no ambiente de experimentações apresentado na seção 4.2, bem como os resultados obtidos e suas análises.

5.2.1 Domínio da Aplicação

O objetivo deste estudo de caso é verificar o desempenho dos processos de criptografia e decriptografia de diferentes quantidades de dados, descartando o processo de envio de dados entre uma entidade e outra a fim de eliminar influências de comunicação e efetuar uma avaliação específica da execução dos algoritmos. É importante observar que o tipo de arquivo não é levado em consideração como um fator de influência nos resultados devido à constatação apresentada por Elminaam et al. (2010), que conduziram testes com arquivos de áudio, vídeo, texto e imagem e reportaram haver comportamentos similares para os processos de criptografia e decriptografia de todos eles.

5.2.2 Configuração do Ambiente de Testes

Uma avaliação de desempenho deve sempre ser conduzida com total conhecimento do ambiente de testes onde os experimentos serão executados, podendo controlar ou limitar fatores que possam influenciar nos resultados (Jain, 1991). Sendo assim, a caracterização do ambiente onde os testes são executados deve ser explicitada em trabalhos desta natureza.

O ambiente selecionado para realização destes experimentos é um *kit* Gumstix Overo EVM rodando uma versão otimizada para ARM do sistema operacional Ubuntu (ver seção 4.2).

5.2.3 Planejamento de Experimentos

Para o planejamento de experimentos apresentado nesta seção será utilizado o modelo fatorial completo. Como mencionado anteriormente na seção 4.3, para obter validação estatística os experimentos devem ser replicados um número de vezes que atinja uma variação baixa entre os resultados obtidos em cada execução, reduzindo o desvio padrão. Para isso, iniciam-se os experimentos com um número de execuções baixo, normalmente algo em torno de 10 e efetua-se uma avaliação dos resultados obtidos buscando verificar se o intervalo de confiança, calculado com base no desvio padrão, está baixo ou se a variação entre os resultados ficou muito discrepante. Com base nesta análise, será possível definir

a necessidade de execução de um número maior de experimentos ou se a amostragem é suficiente.

No estudo de caso em questão, cada configuração de experimento foi executada 30 vezes. A Tabela 5.1 apresenta os fatores e níveis elencados para avaliação neste primeiro estudo de caso.

Tabela 5.1: Configurações dos experimentos realizados no Estudo de Caso 1

Fatores	**Níveis**
Algoritmos	AES 128, AES 192, AES 256, *Blowfish*, DES, 3DES, RC2 40, RC2 64
Dados	T1 (1MB), T2 (3MB), T3 (7MB), T4 (10MB)

O fator Algoritmos pode assumir oito níveis, sendo cada um deles um algoritmo de criptografia. Já o fator Dados pode variar entre quatro tamanhos de mensagem, escolhidos para verificar o impacto da mudança na quantidade de dados a ser processada. T1, T2, T3 e T4 têm os tamanhos 1MB, 3MB, 7MB e 10MB, respectivamente.

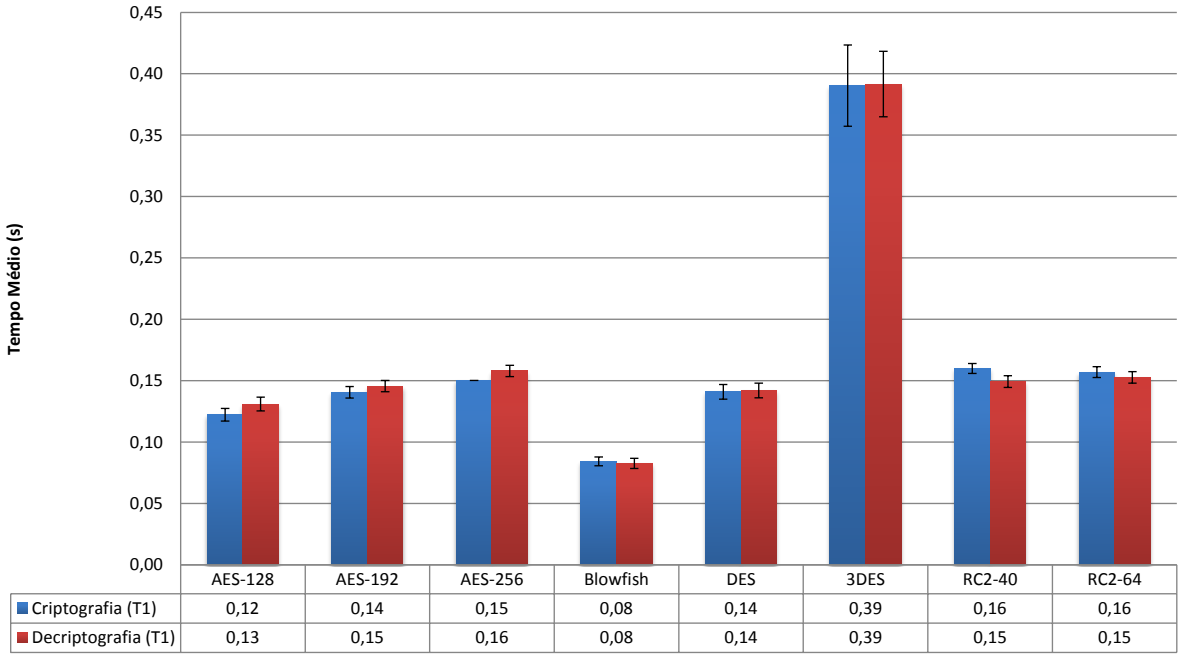
As métricas selecionadas para avaliação são duas: tempo médio de resposta em segundos e *throughput* médio em MB/s. O **tempo médio de resposta** considera o tempo que um algoritmo de criptografia leva para produzir um texto cifrado a partir de um texto simples (processo de criptografia) e vice-versa (processo de decriptografia). Este tempo foi medido com o comando `time` do Linux. O comando `time` monitora a execução de um processo e, ao fim, retorna para a saída padrão estatísticas de tempo de execução. Uma delas é o tempo de usuário, métrica utilizada nestes experimentos.

O ***throughput* médio** é calculado com base no tamanho do arquivo de saída e do tempo despendido no processo. O cálculo efetua a divisão do tamanho do arquivo em *MBytes* pelo tempo da operação em segundos, obtendo assim, o *throughput* médio em MB/s. É importante avaliar o *throughput* pois ele representa a velocidade de processamento de um algoritmo de criptografia (Elminaam et al., 2010; Umaparvathi e Varughese, 2010).

5.2.4 Análise de Resultados

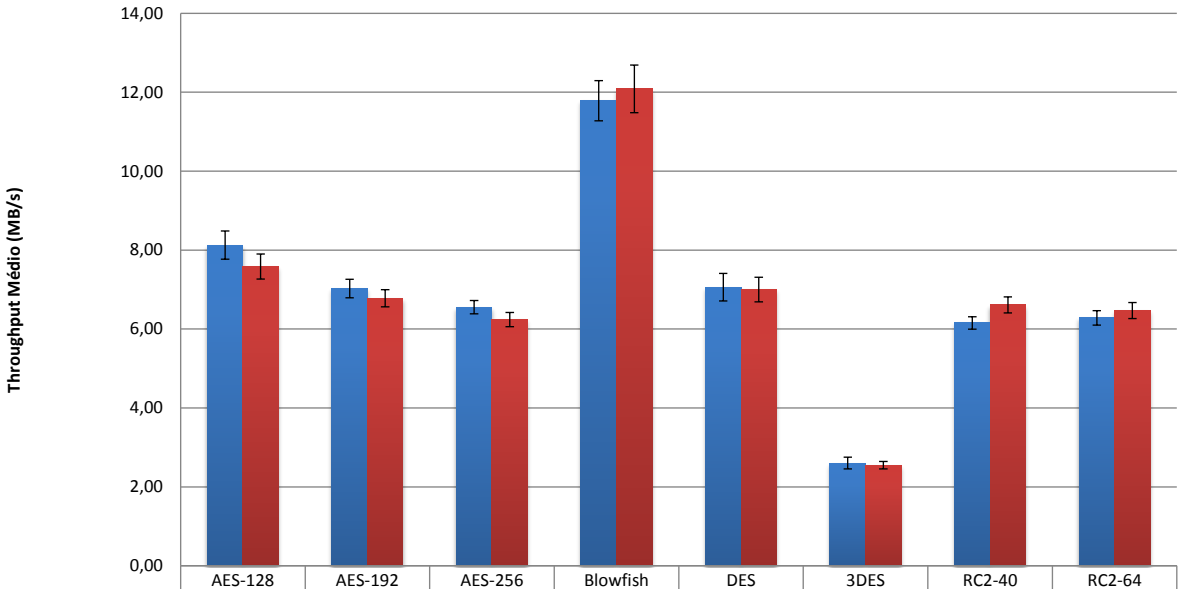
A análise de resultados será conduzida de forma isolada para cada tamanho de mensagem, apresentando, ao fim, uma visão geral de todas elas. Nos gráficos apresentados nas Figuras 5.1 e 5.2 são ilustrados o tempo médio de resposta e o *throughput* médio para o tamanho de mensagem T1 de 1MB.

No gráfico da Figura 5.1 é ilustrado que os tempos de criptografia e decriptografia para cada caso são muito próximos. Em alguns casos é possível afirmar estatisticamente que os tempos são diferentes, uma vez que os intervalos de confiança não se sobrepõem, como foi



	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
Criptografia (T1)	0,12	0,14	0,15	0,08	0,14	0,39	0,16	0,16
Decriptografia (T1)	0,13	0,15	0,16	0,08	0,14	0,39	0,15	0,15

Figura 5.1: Tempo médio de resposta para as operações de criptografia e decriptografia de T1



	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
Criptografia (T1)	8,13	7,03	6,55	11,78	7,06	2,60	6,15	6,28
Decriptografia (T1)	7,58	6,78	6,24	12,08	7,00	2,55	6,61	6,47

Figura 5.2: *Throughput* médio para as operações de criptografia e decriptografia de T1

o caso dos algoritmos AES 256 *bits* e RC2 40 *bits*, que apresentam uma pequena diferença entre os tempos de cada processo.

Efetuando uma análise comparativa entre os algoritmos nestas condições, pode-se notar que o *Blowfish* é o que apresenta menor tempo de criptografia (80 ms) e que o 3DES

apresentou tempo bastante elevado em relação aos demais (390 ms). É importante ressaltar que o 3DES opera com três chaves de 64 *bits*, `k1`, `k2` e `k3` podendo `k1` e `k3` serem iguais. O processo de criptografia do 3DES criptografa com `k1`, decriptografa com `k2` e criptografa novamente com `k3`. Isso faz com que o algoritmo seja mais seguro que o DES, porém, esse processo em três etapas gera um aumento natural no tempo de processamento dos dados. A seção 5.4 vai mostrar uma comparação de desempenho entre o DES e o 3DES com as respectivas influências de fatores, efetuando uma análise mais detalhada.

O gráfico da Figura 5.1 ilustra ainda que há uma pequena elevação no tempo de execução do AES 256 *bits* em relação ao AES 192 *bits* e, da mesma forma, do AES 192 *bits* quando comparado ao AES 128 *bits*. O estudo de caso apresentado na seção 5.3 vai abordar o AES exclusivamente e mostrar uma avaliação de desempenho entre os diferentes tamanhos de chave.

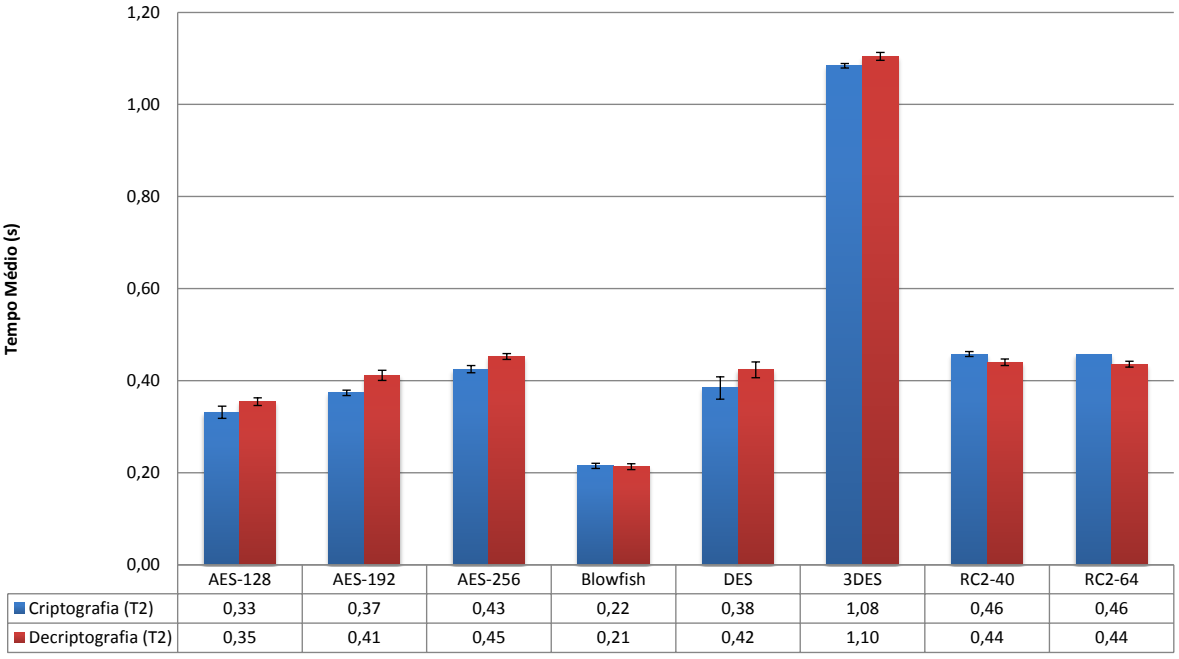
A Figura 5.2 mostra o *throughput* médio de cada operação (criptografia e decriptografia) para cada algoritmo avaliado. Naturalmente, o algoritmo com menor tempo médio de resposta apresenta o maior *throughput* médio, devido à operação base do cálculo do *throughput* médio. O *Blowfish* tem o maior *throughput* médio, aproximadamente cinco vezes maior que o apresentado pelo 3DES.

Partindo do pressuposto de que 1MB de informação pode ser considerado um montante baixo de dados para criptografia com algoritmos simétricos, outros três tamanhos foram selecionados, como mencionado anteriormente, para compor esta avaliação. Os experimentos realizados com a mensagem T2 são apresentados nos gráficos das Figuras 5.3 e 5.4. O tamanho de T2 é 3MB.

O comportamento observado nas Figuras 5.3 e 5.4 é similar ao que se pode observar nos resultados de T1, com uma elevação natural do tempo de resposta devido ao aumento da quantidade de dados a serem processados. É possível verificar também uma estabilização maior dos resultados, o que deve-se ao aumento de dados para processamento, o que consequentemente faz com que haja um ganho no *throughput* médio. Isto será melhor discutido com os gráficos apresentados no final deste estudo de caso.

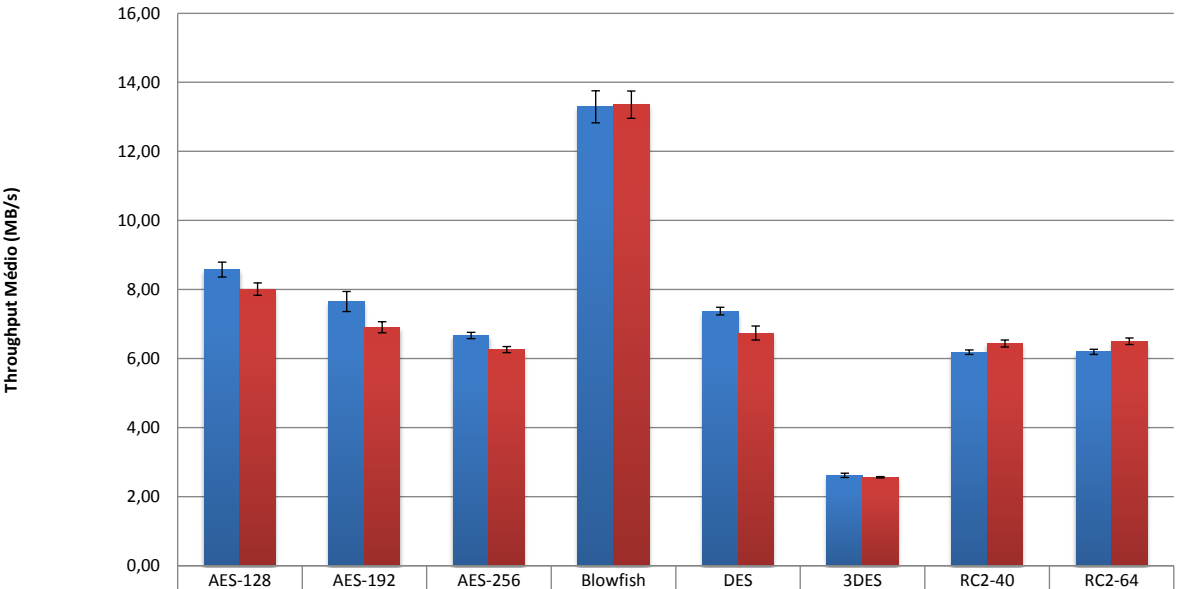
Nas Figuras 5.5 e 5.6 são ilustrados os gráficos gerados com os dados coletados no processamento da mensagem T3 de tamanho 7MB. Novamente, o comportamento dos algoritmos foi similar aos casos de T1 e T2 com uma elevação de tempo devido ao aumento dos dados a serem processados.

Não é possível afirmar, de modo geral, que o processo de criptografia é mais lento ou mais rápido que o processo de decriptografia. Houve uma variação nestes tempos durante os experimentos realizados neste estudo de caso para a maioria dos algoritmos, apesar de haver uma tendência do processo de criptografia ser mais rápido. Em muito casos, os tempos das duas operações são estatisticamente iguais e, mesmo que haja situações em que um dos processos é mais rápido, ainda assim a diferença é muito pequena.



	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
Criptografia (T2)	0,33	0,37	0,43	0,22	0,38	1,08	0,46	0,46
Decriptografia (T2)	0,35	0,41	0,45	0,21	0,42	1,10	0,44	0,44

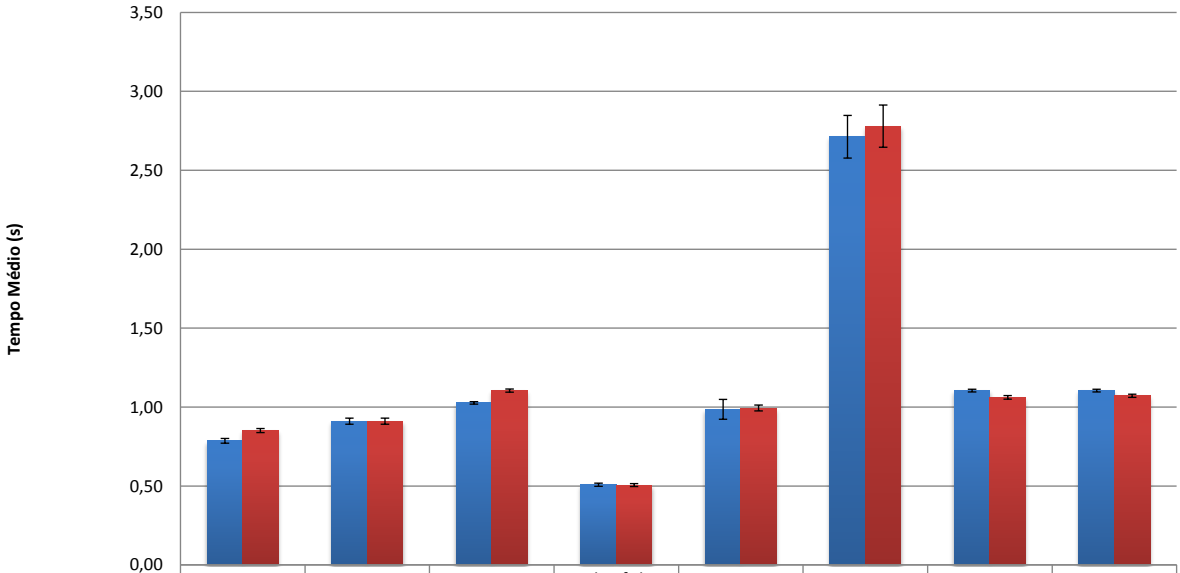
Figura 5.3: Tempo médio de resposta para as operações de criptografia e decriptografia de T2



	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
Criptografia (T2)	8,58	7,65	6,67	13,29	7,38	2,62	6,18	6,19
Decriptografia (T2)	8,01	6,91	6,26	13,35	6,74	2,56	6,44	6,50

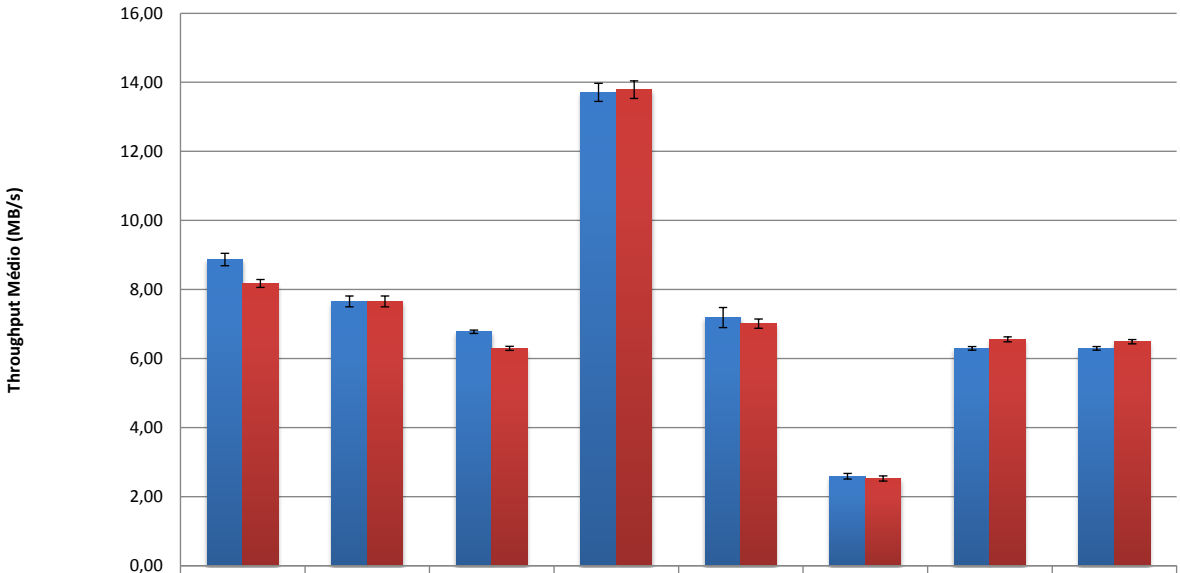
Figura 5.4: *Throughput* médio para as operações de criptografia e decriptografia de T2

Nas Figuras 5.7 e 5.8 são apresentados os resultados obtidos para o tamanho de mensagem T4 de 10MB. De modo geral, efetuando uma análise dos gráficos apresentados para T1, T2, T3 e T4, percebe-se um aumento natural do tempo médio de resposta uma vez que o tamanho do arquivo de dados a serem processados também foi aumentado. Já o *through-*



	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
Criptografia (T3)	0,79	0,91	1,03	0,51	0,99	2,71	1,10	1,10
Decriptografia (T3)	0,85	0,91	1,10	0,51	0,99	2,78	1,06	1,07

Figura 5.5: Tempo médio de resposta para as operações de criptografia e decriptografia de T3



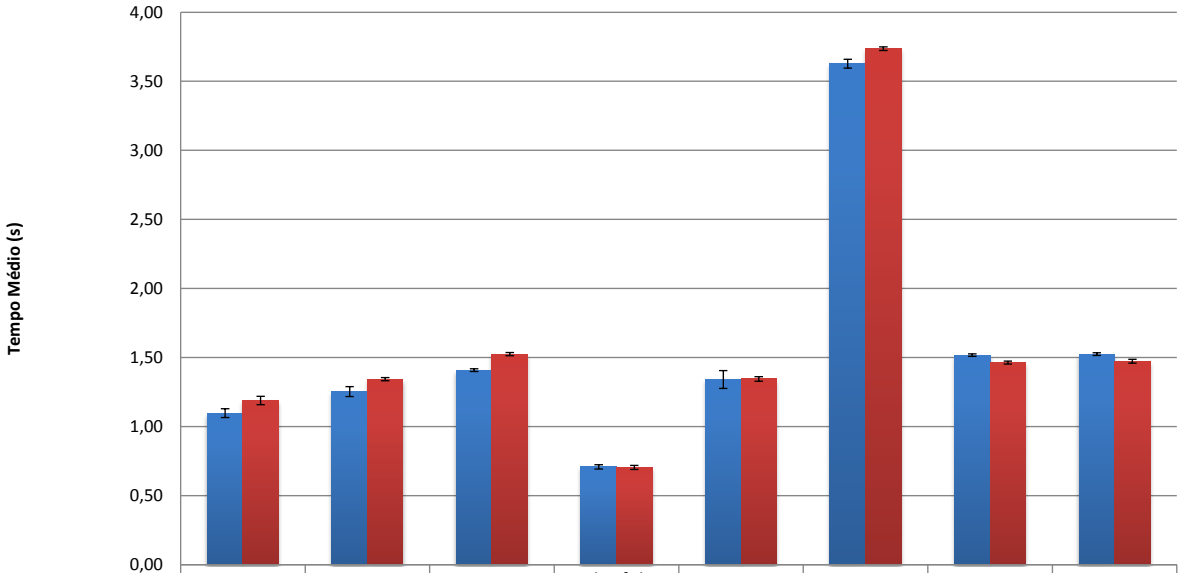
	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
Criptografia (T3)	8,87	7,66	6,77	13,71	7,19	2,59	6,30	6,30
Decriptografia (T3)	8,18	7,66	6,30	13,79	7,01	2,53	6,56	6,49

Figura 5.6: *Throughput* médio para as operações de criptografia e decriptografia de T3

put médio sofre pequenas mudanças e apresenta ganhos conforme aumenta-se o tamanho da mensagem. Isso se dá provavelmente pelo fato de que todos os algoritmos gastam um determinado tempo inerente a ações comuns a todos como manipulação dos arquivos de

entrada e de saída, por exemplo. Esse tempo é igual para todos, independentemente da quantidade de dados a serem processados.

Os tempos médios obtidos para T4 ficaram abaixo de 1,6 s para todos os algoritmos, exceto no 3DES. Estes tempos são baixos e viáveis de se aplicar na comunicação de dois sistemas embarcados críticos, que podem transferir mensagens pequenas como sinais de controle (mudança de direção, avisos de presença etc.) ou até mesmo grandes quantidades de dados, como um *streaming* de vídeo em tempo real ou fotografias em alta resolução que sejam confidenciais e, por isso, seja desejável o uso de criptografia durante o envio ou armazenagem.



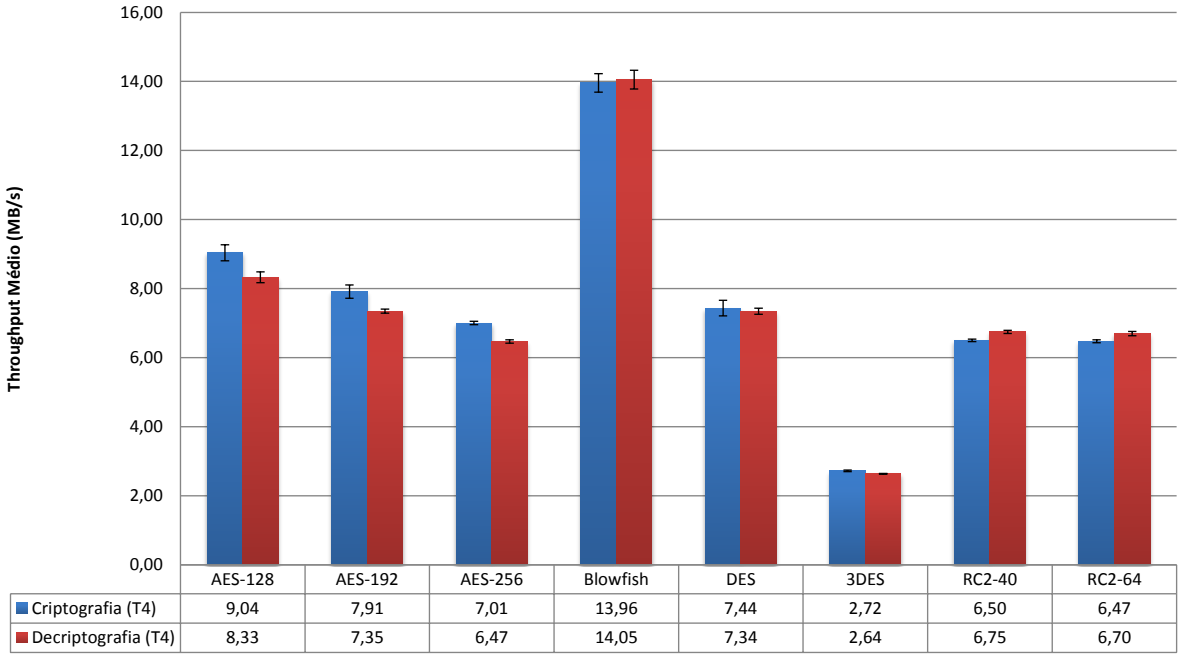
	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
Criptografia (T4)	1,10	1,25	1,41	0,71	1,34	3,63	1,52	1,53
Decriptografia (T4)	1,19	1,34	1,53	0,70	1,35	3,74	1,46	1,47

Figura 5.7: Tempo médio de resposta para as operações de criptografia e decriptografia de T4

Para melhor avaliar os resultados apresentados neste estudo de caso, no gráfico da Figura 5.9 estão ilustrados os resultados obtidos no processo de criptografia dos quatro tamanhos de mensagem abordados até agora: T1, T2, T3 e T4.

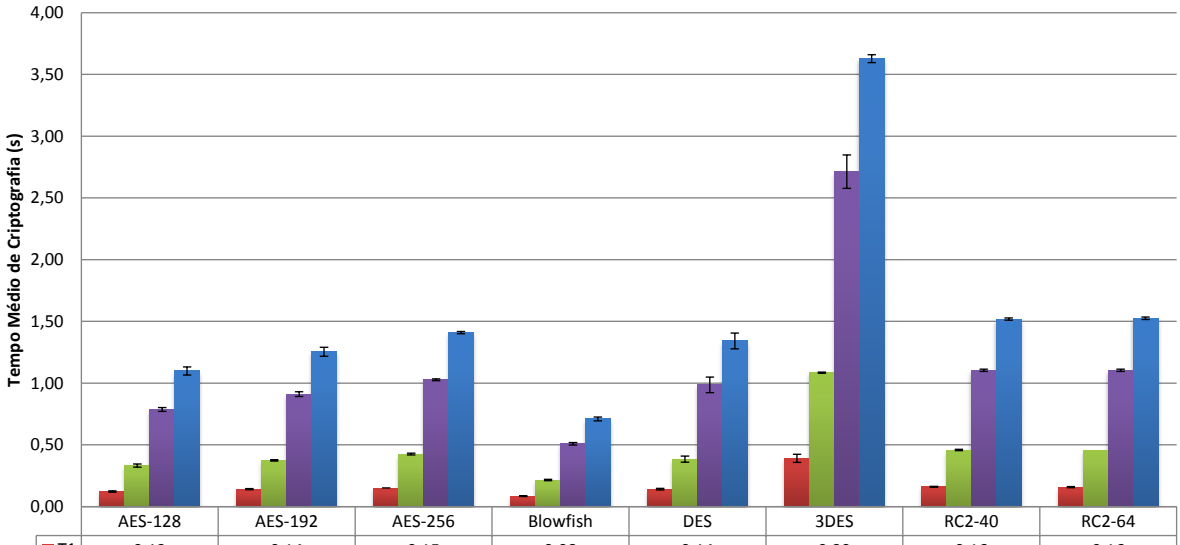
De modo geral, houve um aumento proporcional no tempo de execução de cada tamanho de mensagem. O *Blowfish* é o algoritmo com o melhor desempenho e o 3DES é o algoritmo com o maior tempo de execução, tanto para criptografia quanto para decriptografia. Todos eles, contudo, apresentam tempos relativamente baixos quando comparados aos resultados apresentados em Elminaam et al. (2010), onde o ambiente de experimentações era um computador *laptop* de 2.4 GHz.

No gráfico da Figura 5.10 estão ilustrados os valores de *throughput* médio de criptografia de todos os algoritmos para todos os tamanhos de mensagem. No caso do algoritmo AES 128 *bits*, por exemplo, há uma pequena diferença entre o *throughput* médio obtido



	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
Criptografia (T4)	9,04	7,91	7,01	13,96	7,44	2,72	6,50	6,47
Decriptografia (T4)	8,33	7,35	6,47	14,05	7,34	2,64	6,75	6,70

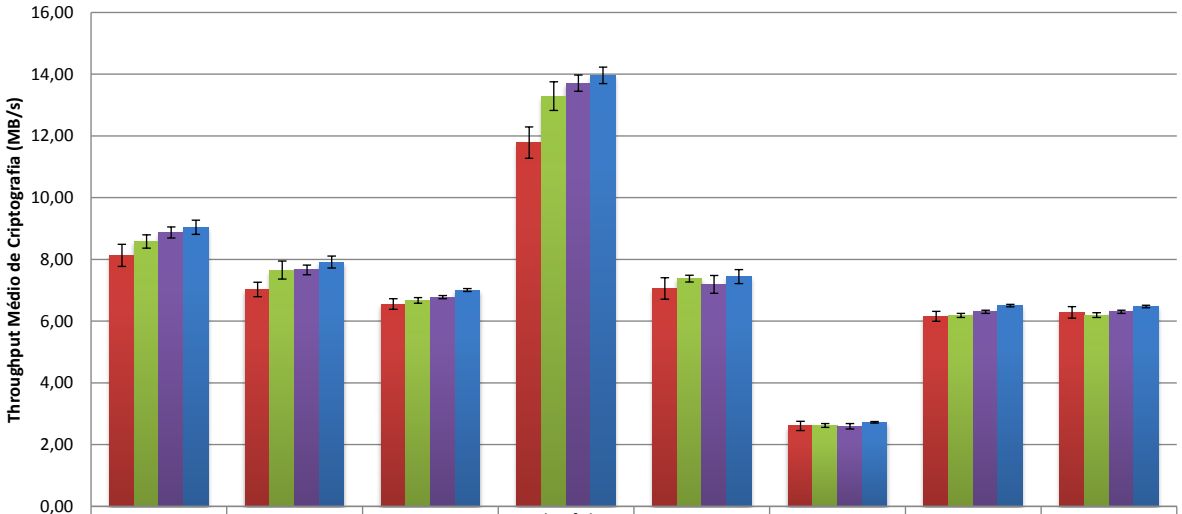
Figura 5.8: *Throughput* médio para as operações de criptografia e decriptografia de T4



	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
T1	0,12	0,14	0,15	0,08	0,14	0,39	0,16	0,16
T2	0,33	0,37	0,43	0,22	0,38	1,08	0,46	0,46
T3	0,79	0,91	1,03	0,51	0,99	2,71	1,10	1,10
T4	1,10	1,25	1,41	0,71	1,34	3,63	1,52	1,53

Figura 5.9: Tempo médio de resposta para a operação de criptografia das mensagens T1, T2, T3 e T4

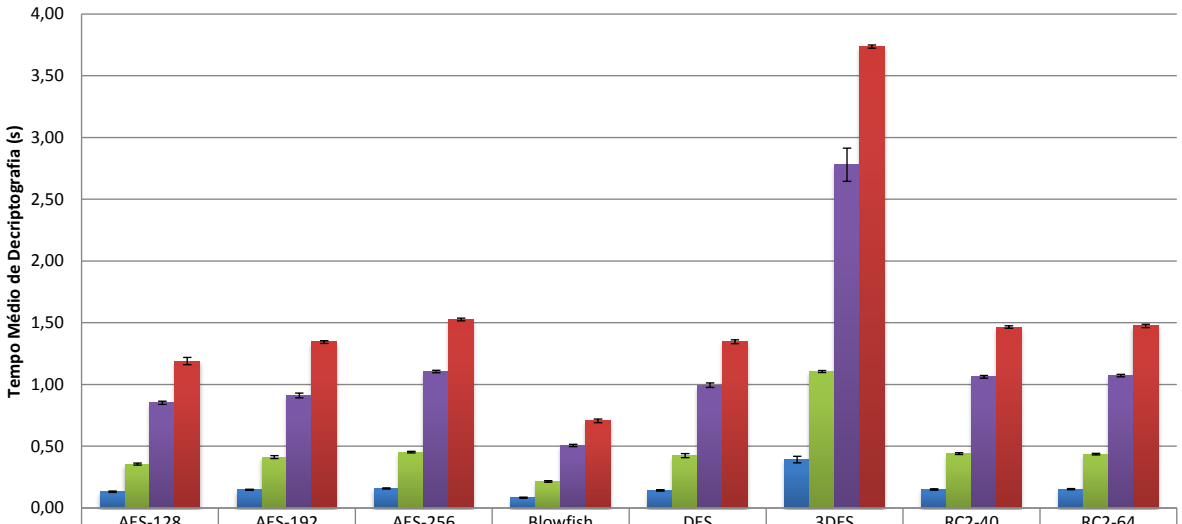
para cada tamanho de mensagem. Em alguns casos, os resultados são estatisticamente iguais, devido à sobreposição dos intervalos de confiança. Comportamentos similares são observados nos demais algoritmos. Como já mencionado, o *Blowfish* tem o maior *throughput* e o 3DES o menor.



	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
T1	8,13	7,03	6,55	11,78	7,06	2,60	6,15	6,28
T2	8,58	7,65	6,67	13,29	7,38	2,62	6,18	6,19
T3	8,87	7,66	6,77	13,71	7,19	2,59	6,30	6,30
T4	9,04	7,91	7,01	13,96	7,44	2,72	6,50	6,47

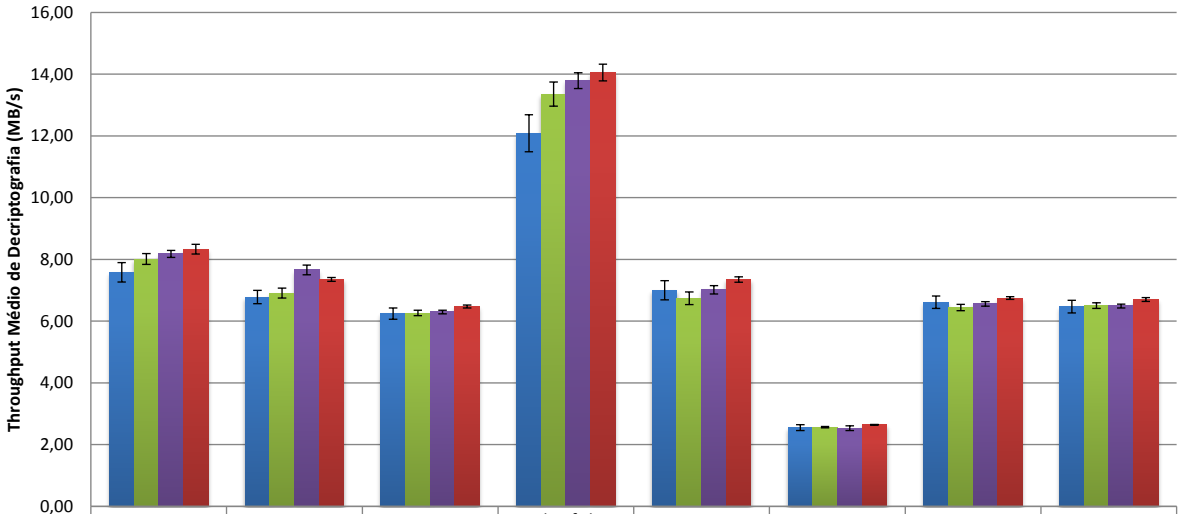
Figura 5.10: *Throughput* médio para a operação de criptografia das mensagens T1, T2, T3 e T4

Nos gráficos das Figuras 5.11 e 5.12 são ilustrados, respectivamente, o tempo médio de resposta e o *throughput* médio dos algoritmos criptográficos durante o processo de decriptografia de todos os tamanhos de mensagem.



	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
T1	0,13	0,15	0,16	0,08	0,14	0,39	0,15	0,15
T2	0,35	0,41	0,45	0,21	0,42	1,10	0,44	0,44
T3	0,85	0,91	1,10	0,51	0,99	2,78	1,06	1,07
T4	1,19	1,34	1,53	0,70	1,35	3,74	1,46	1,47

Figura 5.11: Tempo médio de resposta para a operação de decriptografia das mensagens T1, T2, T3 e T4



	AES-128	AES-192	AES-256	Blowfish	DES	3DES	RC2-40	RC2-64
T1	7,58	6,78	6,24	12,08	7,00	2,55	6,61	6,47
T2	8,01	6,91	6,26	13,35	6,74	2,56	6,44	6,50
T3	8,18	7,66	6,30	13,79	7,01	2,53	6,56	6,49
T4	8,33	7,35	6,47	14,05	7,34	2,64	6,75	6,70

Figura 5.12: *Throughput* médio para a operação de decriptografia das mensagens T1, T2, T3 e T4

De modo geral, o *Blowfish* apresentou o melhor desempenho, seguido do AES 128 *bits* e do AES 192 *bits*. O quarto melhor desempenho foi do DES, seguido pelo AES 256 *bits*, pelo RC2 40 *bits* e 64 *bits* e, por fim, pelo 3DES, que apresenta um desempenho consideravelmente inferior aos demais. Como já discutido, essa característica do 3DES não o torna inviável, uma vez que o processo de criptografia dele em comparação aos demais é diferente e envolve criptografia tripla dos dados.

O comparativo apresentado neste estudo de caso considera os tempos médios de operação de algoritmos simétricos mais utilizados na literatura. Em aplicações reais é necessário levar em consideração que cada algoritmo utiliza modos diferentes de operação e, por isso, fornecem níveis de segurança que dependem unicamente da forma como eles criptografam/decriptografam os dados e do tamanho da chave utilizada.

Os estudos de caso 2 e 3 (seções 5.3 e 5.4) conduzirão a uma avaliação de desempenho mais específica e detalhada dos algoritmos AES e DES, respectivamente.

5.3 Estudo de Caso 2: Avaliação de Desempenho do AES

Este estudo de caso será dividido em três partes de modo a comparar pares de tamanhos de chaves do AES em cada etapa. O domínio da aplicação e a configuração do ambiente de testes são comuns aos três cenários e serão abordados a seguir.

5.3.1 Domínio da Aplicação

O objetivo principal deste estudo de caso é avaliar o algoritmo AES quando operante com chaves de tamanho 128, 192 e 256 *bits*. O estudo de caso apresentado na seção 5.2 abordou diversos algoritmos simétricos, inclusive o AES, porém, com os gráficos apresentados no estudo não é possível avaliar, de fato, o efeito do tamanho da chave nos resultados. Por meio de técnicas de avaliação de desempenho, será possível visualizar o impacto do tamanho da chave e se ele é um fator relevante a ser considerado em aplicações práticas com sistemas embarcados.

5.3.2 Configuração do Ambiente de Testes

Assim como no primeiro estudo de caso, o ambiente selecionado para realização destes experimentos é um *kit* Gumstix Overo EVM rodando uma versão otimizada para ARM do sistema operacional Ubuntu (ver seção 4.2).

5.3.3 Planejamento de Experimentos

O planejamento de experimentos considera o modelo fatorial completo e visa analisar dois fatores: tamanho da chave e tamanho da mensagem. Este estudo de caso será conduzido em três partes, como já mencionado, e o fator tamanho da chave é variante em cada uma delas. As Tabelas 5.2, 5.3 e 5.4 mostram as configurações de experimentos para cada parte da avaliação, sendo a primeira delas entre o AES 128 *bits* e o AES 192 *bits*, a segunda entre o AES 192 *bits* e o AES 256 *bits* e a terceira entre o AES 128 *bits* e o AES 256 *bits*.

Tabela 5.2: Configurações dos experimentos de comparação do AES com chaves de 128 e 192 *bits*

Exp.	Fator A (Tamanho da chave)	Fator B (Tamanho da mensagem)
1	AES 128	T3
2	AES 128	T4
3	AES 192	T3
4	AES 192	T4

Os valores T3 e T4 que podem ser assumidos pelo fator B correspondem aos mesmos tamanhos utilizados no estudo de caso 1 (seção 5.2): 7MB e 10MB. A variável de resposta (saída) dos experimentos é o tempo médio. Este tempo corresponde ao processo completo de criptografia e decriptografia de cada arquivo, ou seja, não há separação entre os tempos de cada processo. Cada experimento foi replicado 30 vezes.

Tabela 5.3: Configurações dos experimentos de comparação do AES com chaves de 192 e 256 *bits*

Exp.	Fator A (Tamanho da chave)	Fator B (Tamanho da mensagem)
1	AES 192	T3
2	AES 192	T4
3	AES 256	T3
4	AES 256	T4

Tabela 5.4: Configurações dos experimentos de comparação do AES com chaves de 128 e 256 *bits*

Exp.	Fator A (Tamanho da chave)	Fator B (Tamanho da mensagem)
1	AES 128	T3
2	AES 128	T4
3	AES 256	T3
4	AES 256	T4

5.3.4 Análise de Resultados

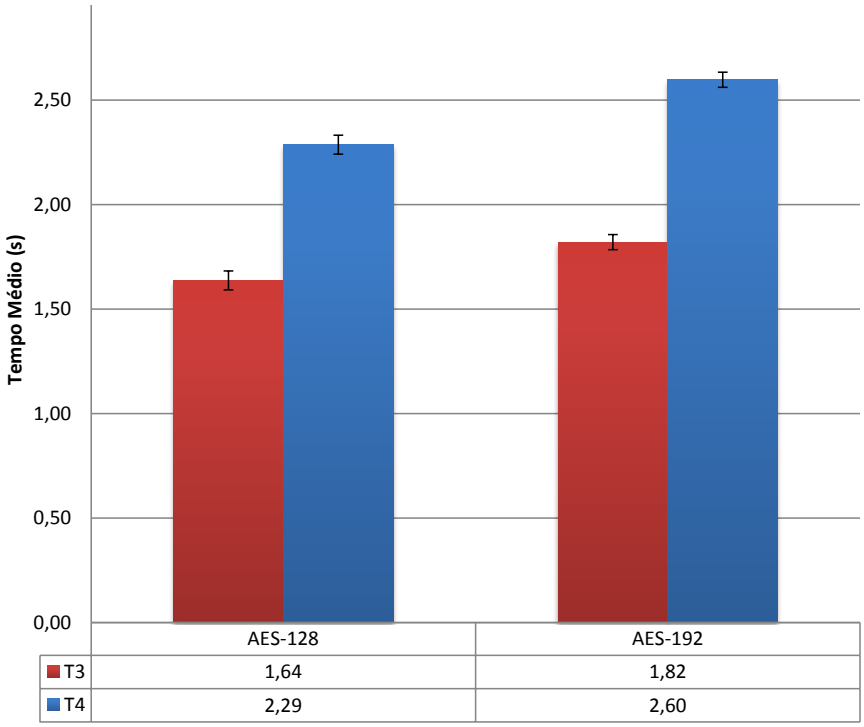


Figura 5.13: Tempo médio dos processos de criptografia e decriptografia de T3 (7MB) e T4 (10MB) pelo AES 128 e 192 *bits*

No gráfico da Figura 5.13 estão ilustrados os resultados obtidos para T3 e T4 quando processados pelo algoritmo AES com os tamanhos de chave 128 e 192 *bits* (ver Tabela 5.2).

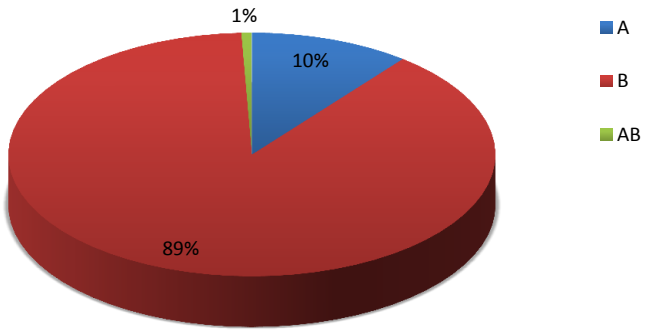


Figura 5.14: Influências dos fatores para os processos de criptografia e decriptografia de T3 e T4 pelo AES 128 e 192 *bits* (A – Tamanho da chave; B – Tamanho da mensagem)

Nota-se uma pequena diferença entre o tempo de execução do AES 192 *bits* em relação ao AES 128 *bits* devido à elevação do tamanho da chave. Este comportamento ocorre para ambos os tamanhos de mensagem adotados. Nota-se também que o intervalo de confiança é baixo, o que demonstra que houve uma baixa variação entre os resultados obtidos nas 30 replicações.

Como mencionado anteriormente, a importância de um fator é medida pela proporção na variação total da variável de resposta que é influenciada por esse fator. No gráfico da Figura 5.14 estão ilustradas as parcelas de influência de cada fator. No caso destes experimentos, o fator de maior influência foi o tamanho da mensagem (fator B) com 89%. Em seguida está o tamanho da chave (fator A) com 10% de influência. A associação dos dois fatores resulta em uma pequena influência de 1%. Estes resultados mostram que a variação da chave, nestas condições, não foi fator de muita relevância e que o tempo de resposta depende principalmente da quantidade de dados a serem processados.

O gráfico da Figura 5.15 refere-se à configuração apresentada na Tabela 5.3. Ele ilustra um comparativo dos tempos médios de processamento dos arquivos T3 e T4 pelo algoritmo AES operando com chaves de 192 e 256 *bits*. As influências dos fatores para este caso podem ser observadas no gráfico da Figura 5.16. Percebe-se que a influência do tamanho da chave (fator A) é de 86%, do tamanho da mensagem (fator B) é de 14% e que os fatores A e B associados exercem influência próxima de zero.

A terceira parte deste estudo de caso avalia o algoritmo AES com variação do tamanho de chave entre 128 e 256 *bits*. Devido à diferença de tamanho entre as chaves ser maior, observa-se uma elevação na diferença de tempo médio para execução dos processos de criptografia e decriptografia como pode ser visto na Figura 5.17, que ilustra o resultados dos experimentos planejados e apresentados na Tabela 5.4.

No gráfico da Figura 5.18 está ilustrada que a maior influência é explicada pelo tamanho da mensagem (fator A) com 61% quando comparados o AES 128 *bits* com o AES 256 *bits*. Apesar de continuar sendo a influência dominante, houve uma diminuição da por-

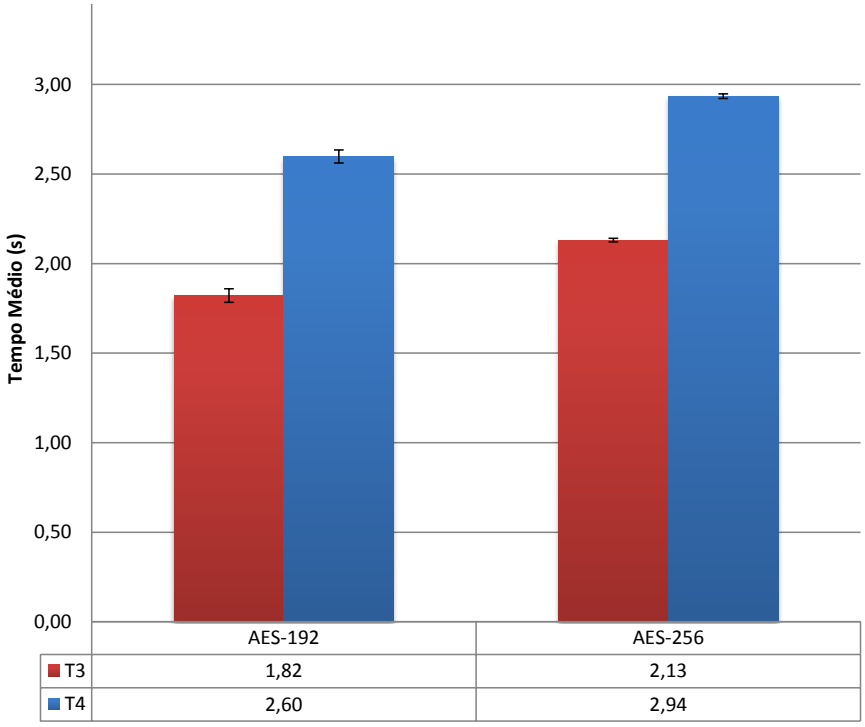


Figura 5.15: Tempo médio dos processos de criptografia e decriptografia de T3 (7MB) e T4 (10MB) pelo AES 192 e 256 *bits*

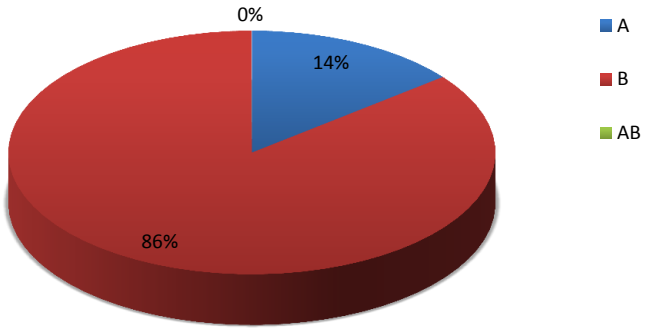


Figura 5.16: Influências dos fatores para os processos de criptografia e decriptografia de T3 e T4 pelo AES 192 e 256 *bits* (A – Tamanho da chave; B – Tamanho da mensagem)

centagem devido ao crescimento da diferença de tamanho entre as chaves, o que fez com que o tamanho da chave (fator A) tivesse uma maior influência nos resultados, totalizando 38%. Os fatores A e B associados influenciaram 1% nos resultados.

Estes números ajudam a perceber que o tamanho da chave de criptografia utilizada no AES influencia nos resultados, mas em determinados contextos pode não ser um fator crítico. É sempre aconselhável analisar o contexto de aplicação de uma solução de segurança e avaliar se há necessidade de fortalecer a criptografia utilizada com o uso de chaves de tamanhos maiores. Quando o tempo para quebra de uma cifra ultrapassa um determinado prazo, torna-se viável o uso de algoritmos operantes com chaves de tamanhos menores,

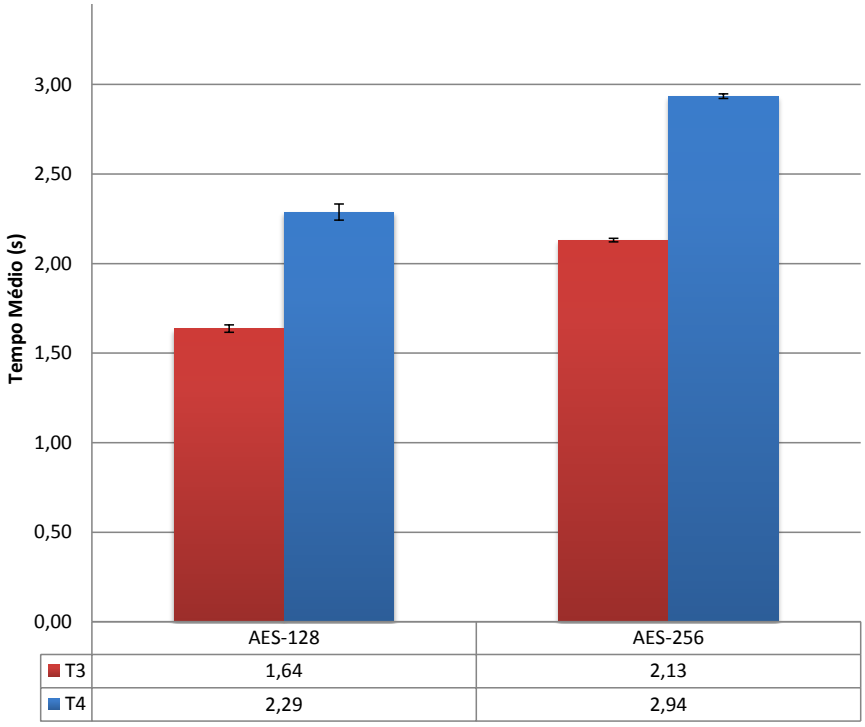


Figura 5.17: Tempo médio dos processos de criptografia e decriptografia de T3 (7MB) e T4 (10MB) pelo AES 128 e 256 *bits*

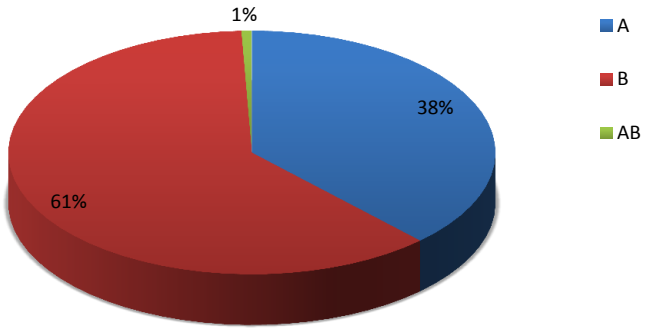


Figura 5.18: Influências dos fatores para os processos de criptografia e decriptografia de T3 e T4 pelo AES 128 e 256 *bits* (A – Tamanho da chave; B – Tamanho da mensagem)

uma vez que a afirmação de que um sistema é seguro deve variar de acordo com o tempo de quebra daquela segurança. Supondo que uma operação de ataque leve 2 meses para quebrar uma cifra, escolhe-se uma frequência de troca de chave que seja menor do que este período. O principal benefício de se usar algoritmos com chaves menores que atendam aos requisitos de segurança, é a geração de um *overhead* menor no sistema, o que vai ao encontro dos objetivos de projetistas e desenvolvedores de *software* e *hardware* para sistemas embarcados críticos.

Neste estudo de caso foi possível perceber que a escolha do melhor tamanho de chave para o algoritmo AES vai depender do contexto onde esta criptografia será aplicada. É

necessário avaliar a criticidade do sistema e verificar se o tempo gasto pelo algoritmo para criptografar e decriptografar os dados é aceitável ou se ele extrapola o limite estabelecido de acordo com o contexto da aplicação.

5.4 Estudo de Caso 3: Avaliação de Desempenho do DES

Da mesma forma que o anterior, este estudo de caso apresenta uma avaliação de desempenho entre dois algoritmos: o DES e sua variação 3DES. Como explicado anteriormente, o 3DES opera com três chaves, efetuando o processo de criptografia em três etapas: criptografia dos dados com a chave `k1`, decriptografia destes dados com a chave `k2` e uma nova criptografia dos dados com `k3`. Da mesma forma o processo de decriptografia é executado em três etapas, porém no sentido contrário.

Sendo assim, um comportamento esperado para os resultados é que o 3DES leve aproximadamente três vezes o tempo necessário ao DES para criptografar a mesma quantidade de dados.

5.4.1 Domínio da Aplicação

O objetivo deste estudo de caso é verificar a influência dos algoritmos DES e 3DES nos resultados quando criptografando dois tamanhos diferentes de mensagens. Para extrair maiores informações, técnicas de avaliação de desempenho serão novamente empregadas, obtendo assim a influência dos fatores elencados nestes experimentos.

5.4.2 Configuração do Ambiente de Testes

O ambiente selecionado para realização destes experimentos é um *kit* Gumstix Overo EVM rodando uma versão otimizada para ARM do sistema operacional Ubuntu (ver seção 4.2), o mesmo adotado nos estudos de caso 1 e 2.

5.4.3 Planejamento de Experimentos

Os experimentos seguem o modelo fatorial completo, que avalia todas as combinações possíveis em todos os níveis de todos os fatores. Um dos fatores é o algoritmo, variante entre DES e 3DES, e o segundo fator é o tamanho da mensagem, variante entre T3 (7MB) e T4 (10MB), os mesmos arquivos utilizados no estudo de caso 2. Na Tabela 5.5 são ilustradas as quatro configurações de experimentos executadas. É importante ressaltar que cada uma delas foi replicada 30 vezes.

A variável de resposta avaliada é o tempo médio de resposta medido em segundos. Novamente, o tempo médio refere-se à soma dos tempos de criptografia e decriptografia.

Tabela 5.5: Configurações dos experimentos de comparação entre o DES e o 3DES

Exp.	Fator A (Algoritmo)	Fator B (Tamanho da mensagem)
1	DES	T3
2	DES	T4
3	3DES	T3
4	3DES	T4

5.4.4 Análise de Resultados

Os resultados obtidos para estes experimentos estão expressados em gráfico na Figura 5.19. Comparando-se os tempos médios obtidos para T3 e T4 é possível perceber uma diferença devido à quantidade de dados a serem processados. Porém, comparando os tempos médios obtidos por cada algoritmo, percebe-se uma diferença maior, o que leva a acreditar que o fator mais influente neste caso foi o algoritmo. Para comprovar, o cálculo das influências dos fatores foi efetuado e está representado graficamente na Figura 5.20.

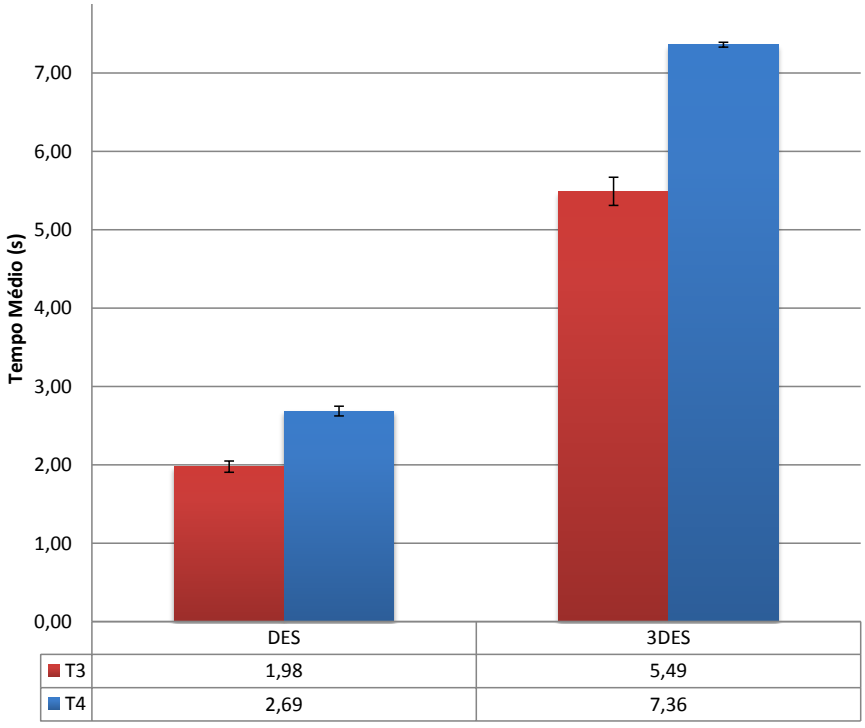


Figura 5.19: Tempo médio dos processos de criptografia e decriptografia de T3 (7MB) e T4 (10MB) pelo DES e pelo 3DES

O fator A (algoritmo) foi o mais influente com uma participação de 89% nos resultados. Isso pode ser explicado pela criptografia em três etapas, o que acaba gerando um processamento consideravelmente maior. O fator B (tamanho da mensagem) teve influência de 9% e os fatores A e B associados de apenas 2%.

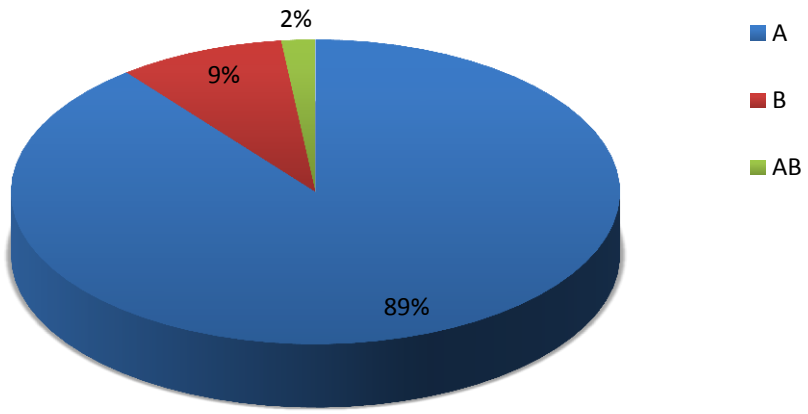


Figura 5.20: Influências dos fatores para os processos de criptografia e decriptografia de T3 e T4 pelo DES e pelo 3DES (A – Algoritmo; B – Tamanho da mensagem)

O DES, assim como todos os demais algoritmos de criptografia, é vulnerável ao ataque de força bruta. O tempo de quebra de uma cifra que utiliza chave de 56 *bits* é baixo e viabiliza ataques. O 3DES, por sua vez, resolve este problema criptografando três vezes o texto plano e por utilizar duas ou três chaves durante este processo, ampliando o tempo de quebra e descoberta da chave secreta. Apesar de apresentar um tempo elevado em relação ao DES, ele é empregado por prover um nível de segurança maior.

5.5 Estudo de Caso 4: Comparação de Desempenho entre Implementações de Criptografia de Curva Elíptica

Esta seção tem como objetivo apresentar a metodologia utilizada nos experimentos realizados com duas implementações do algoritmo ECC, bem como os resultados obtidos e suas análises. É importante ressaltar que parte dos resultados descritos neste estudo de caso deram origem a duas publicações: Pigatto et al. (2011) Pigatto et al. (2012).

5.5.1 Domínio da Aplicação

O objetivo deste estudo de caso é efetuar uma avaliação de desempenho do algoritmo assimétrico conhecido como ECC. Os resultados serão apresentados em duas etapas. No primeiro momento, uma comparação entre o ECC implementado com as bibliotecas Miracl e Relic será apresentada. Com base nisso, o algoritmo com melhor desempenho é utilizado para uma nova avaliação de desempenho após a compilação da biblioteca para a arquitetura ARM.

5.5.2 Configuração do Ambiente de Testes

A primeira parte dos testes apresentará a avaliação de desempenho entre os algoritmos implementados nas bibliotecas Miracl e Relic sendo executadas em um computador *desktop* Pentium Dual-Core CPU T4300 2.10GHz, com 2GB de RAM e sistema operacional Linux Ubuntu 10.04 LTS. Esta etapa da avaliação visa identificar qual é a melhor biblioteca para ser utilizada na implementação do ECC.

Já na segunda etapa desta avaliação, a biblioteca com o melhor desempenho é portada e executada em um *kit* Gumstix Overo EVM rodando uma versão otimizada para ARM do sistema operacional Ubuntu (ver seção 4.2) e, posteriormente, comparada com a execução no computador *desktop* citado no parágrafo anterior.

5.5.3 Planejamento de Experimentos

Os experimentos da primeira etapa consideram três fatores, cada um com dois níveis: biblioteca, tamanho da chave e tamanho da mensagem. A Tabela 5.6 apresenta as oito combinações possíveis de acordo com o modelo fatorial completo.

Tabela 5.6: Configurações dos experimentos de comparação entre o ECC com as bibliotecas Miracl e Relic

Exp.	Fator A (Biblioteca)	Fator B (Tam. chave)	Fator C (Tam. mensagem)
1	Miracl	160	T5
2	Miracl	160	T6
3	Miracl	256	T5
4	Miracl	256	T6
5	Relic	160	T5
6	Relic	160	T6
7	Relic	256	T5
8	Relic	256	T6

As mensagens T5 e T6 têm tamanhos 50KB e 100KB, respectivamente. Estes tamanhos podem parecer pequenos quando comparados aos demais estudos de caso que foram executados com mensagens de tamanhos bem maiores, contudo, a razão pelo uso de mensagens de tamanho reduzido se deve ao emprego comum de algoritmos assimétricos. Normalmente eles são utilizados para troca de chaves simétricas ou para assinatura digital, não sendo empregados para trocas de dados propriamente ditos devido ao desempenho reduzido quando comparados a algoritmos simétricos. Segundo Xu et al. (2004), em VANETs quando há trocas de mensagens de controle, um tamanho de mensagem aceito como comum é 250 *bytes*. Os experimentos utilizam mensagens de tamanhos considera-

velmente maiores, sendo elas 50 e 100KB, o que gera uma carga de trabalho maior para esta avaliação.

Os experimentos da primeira etapa deste estudo de caso foram executados 15 vezes, o que garantiu uma variação baixa no desvio padrão e foi, portanto, suficiente para a avaliação. O tempo médio de execução corresponde à soma dos tempos de criptografia e decriptografia de cada mensagem, sendo esta a métrica selecionada para avaliação.

A partir dos resultados da primeira avaliação que serão apresentados na próxima seção, é possível definir o melhor algoritmo e torná-lo um fator fixo da avaliação desempenho da segunda etapa do estudo de caso, cujo planejamento de experimentos segue a configuração expressa na Tabela 5.7.

Tabela 5.7: Configurações dos experimentos de comparação do ECC em ambiente *Desktop* e no *kit* Gumstix Overo EVM

Exp.	Fator A (Ambiente)	Fator B (Tam. chave)	Fator C (Tam. mensagem)
1	Desktop	160	T5
2	Desktop	160	T6
3	Desktop	256	T5
4	Desktop	256	T6
5	Gumstix	160	T5
6	Gumstix	160	T6
7	Gumstix	256	T5
8	Gumstix	256	T6

A métrica de desempenho destes experimentos continua sendo o tempo médio de execução e o número de replicações também foi de 15 vezes.

5.5.4 Análise de Resultados

As bibliotecas utilizadas nas implementações dos algoritmos abordados neste estudo de caso foram apresentadas na seção 4.2.3. Os parâmetros que definem a curva elíptica e o ponto utilizado em comum pelos usuários foram fixados para que os algoritmos pudessem ser avaliados de forma justa. Os resultados relativos à comparação entre os algoritmos implementados com as bibliotecas Miracl e Relic (ver Tabela 5.6) estão apresentados na Figura 5.21.

Efetuando uma análise dos resultados, pode-se notar que a biblioteca Relic apresenta um desempenho superior em relação à biblioteca Miracl. Há uma clara degradação do tempo de execução quando o tamanho da chave é de 256 *bits*. É necessário lembrar que um algoritmo de curva elíptica com uma chave de 160 *bits* é equivalente ao RSA com 1024 *bits*, o que, portanto, infere que uma chave de 256 *bits* no ECC é suficiente para a maioria das aplicações.

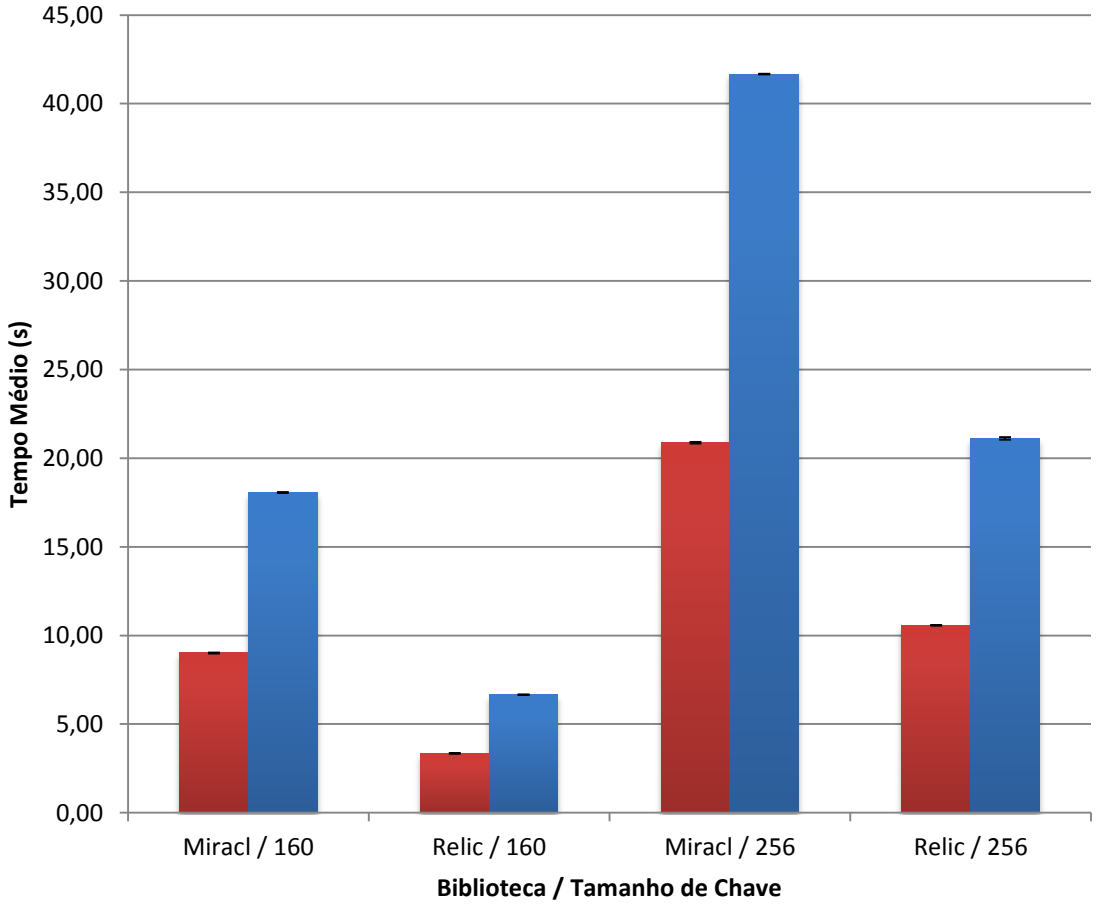


Figura 5.21: Tempo médio da soma das execuções dos processos de criptografia e decriptografia com o algoritmo ECC baseado nas bibliotecas Miracl e Relic

Os tempos médios de resposta do ECC são altos em relação aos algoritmos de criptografia simétrica, como esperado. Isso se deve ao já mencionado fator que caracteriza algoritmos de criptografia assimétrica, os quais operam com cálculos mais intensos do que os algoritmos classificados como simétricos.

No gráfico da Figura 5.22 estão ilustradas as influências dos fatores para esta primeira avaliação de desempenho deste estudo de caso. O fator B (tamanho da chave) foi a maior influência nos resultados finais, com uma parcela de 40%, o que é bastante relevante. Em seguida, o fator A (biblioteca) influenciou em 28% os resultados dos experimentos. A terceira maior influência é atribuída ao fator C (tamanho da mensagem), com 23%. A associação dos fatores BC teve influência de 4%, AC de 3%, AB de 2% e ABC não exerceu influência.

Após esta primeira avaliação de desempenho, foi possível descobrir que o algoritmo baseado na biblioteca Relic apresentou um melhor desempenho em relação ao baseado na Miracl. Sendo assim, a biblioteca Relic foi portada para a arquitetura ARM de modo a efetuar uma comparação entre a execução em um computador *desktop* e a execução em um

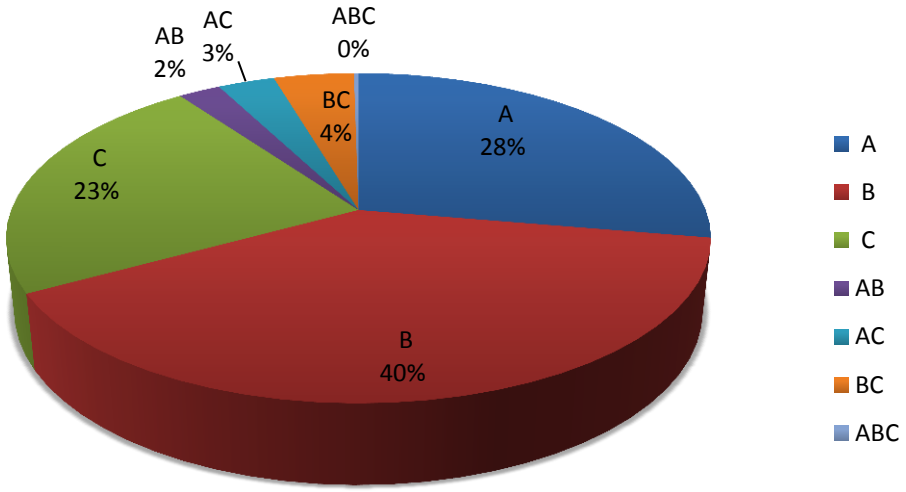


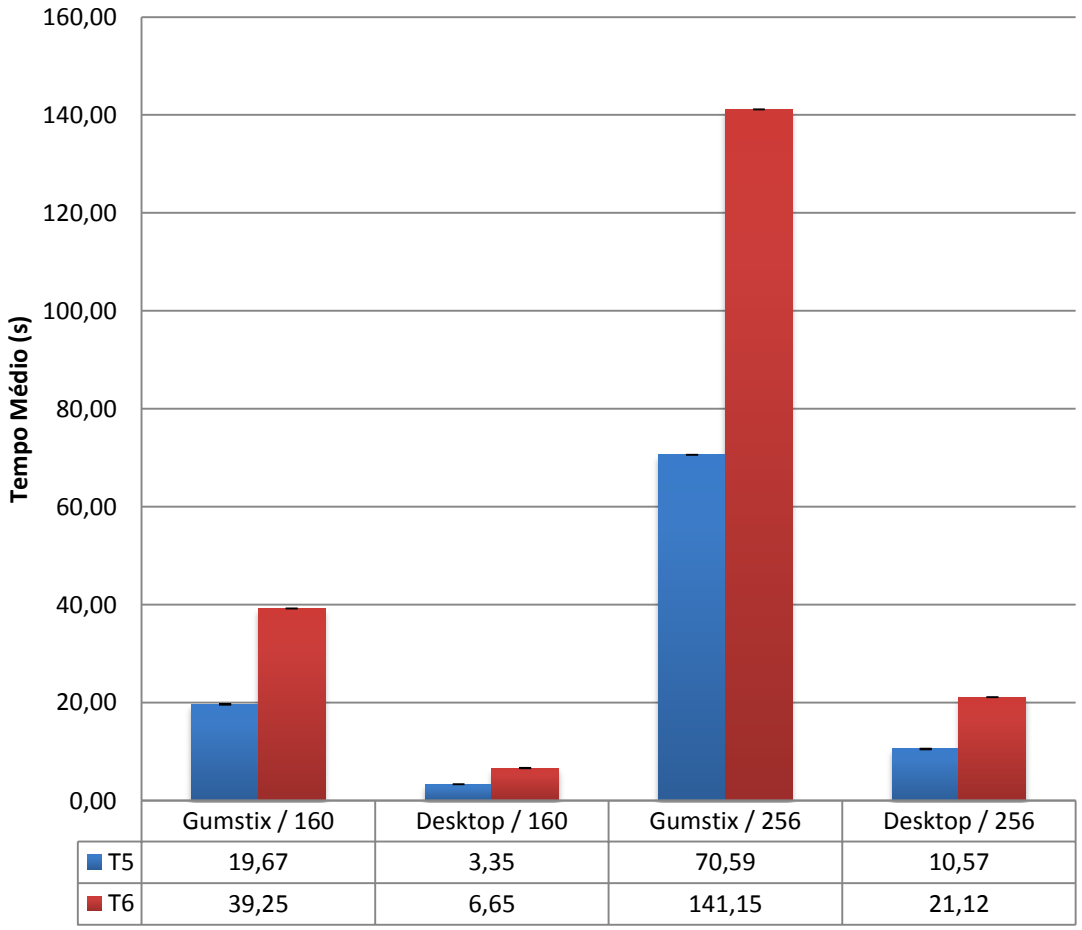
Figura 5.22: Influências dos fatores da avaliação do ECC baseado nas bibliotecas Miracl e Relic (A – Biblioteca; B – Tamanho da chave; C – Tamanho da mensagem)

kit Gumstix Overo EVM (ver Tabela 5.7). Para efetuar este processo, uma *flag* específica para compilação para ARM foi utilizada, a qual consta na documentação da biblioteca Relic (Aranha e Gouvêa, 2011).

O comportamento dos resultados para os dois ambientes estão ilustrados no gráfico da Figura 5.23. Houve uma notável degradação no tempo de execução do algoritmo quando portado para a arquitetura ARM. Considerando que o ECC é um algoritmo de chave pública (ou assimétrico), este comportamento é aceitável. Segundo Coulouris et al. (2005), algoritmos assimétricos podem exigir de 100 a 1000 vezes mais processamento do que algoritmos simétricos. Sendo assim, há uma maior facilidade de se extrapolar recursos de processamento e memória em um ambiente embarcado em que estes elementos são bastante limitados, como discutido anteriormente.

O algoritmo de curva elíptica é bastante indicado para o processo de troca de chaves durante a fase de estabelecimento de uma conexão, já que ele utiliza um par de chaves (pública e privada) como todo algoritmo assimétrico. O ECC pode ser empregado, portanto, em situações onde se busque autenticidade e/ou confidencialidade.

A análise das influências dos fatores permite uma análise mais detalhada dos resultados (ver Figura 5.24). O ambiente (fator A) foi o que mais influenciou nos resultados com um total de 43%. Como já mencionado, houve uma degradação no desempenho quando executando no *kit* Gumstix Overo EVM devido à limitação de recursos comum neste tipo de ambiente. O segundo fator de maior influência foi o tamanho da chave (fator B), que explica por 25% dos resultados obtidos. Aqui é importante observar que uma chave de 160 *bits* é tida como equivalente a uma chave de 1024 *bits* no RSA, sendo a chave de 256 *bits* considerada, portanto, de uma segurança bastante alta, uma vez que equivale a uma chave de tamanho 3072 *bits* no RSA (Lenstra e Verheul, 1999).



	Gumstix / 160	Desktop / 160	Gumstix / 256	Desktop / 256
T5	19,67	3,35	70,59	10,57
T6	39,25	6,65	141,15	21,12

Figura 5.23: Tempo médio da soma das execuções dos processos de criptografia e decriptografia com o algoritmo ECC baseado na biblioteca Relic

A associação dos fatores A e B gerou uma influência de 14% nos resultados, seguida pelo fator C (tamanho da mensagem) com influência de 9%. Os fatores A e C associados tiveram influência de 5%, B e C de 3% e a associação de todos os três fatores exerceu influências de apenas 1%.

Este estudo de caso contribui com a verificação de que existem limitações de recursos em ambientes embarcados e elas devem ser levadas em consideração na hora da escolha da chave e do algoritmo mais apropriados, tendo em vista a degradação no desempenho geral do sistema e também a criticidade das informações transmitidas, fator que vai refletir diretamente na escolha da melhor solução criptográfica. Há ainda a possibilidade de se reduzir o *overhead* com a aplicação de assinatura digital, a qual pode ser realizada com o ECC.

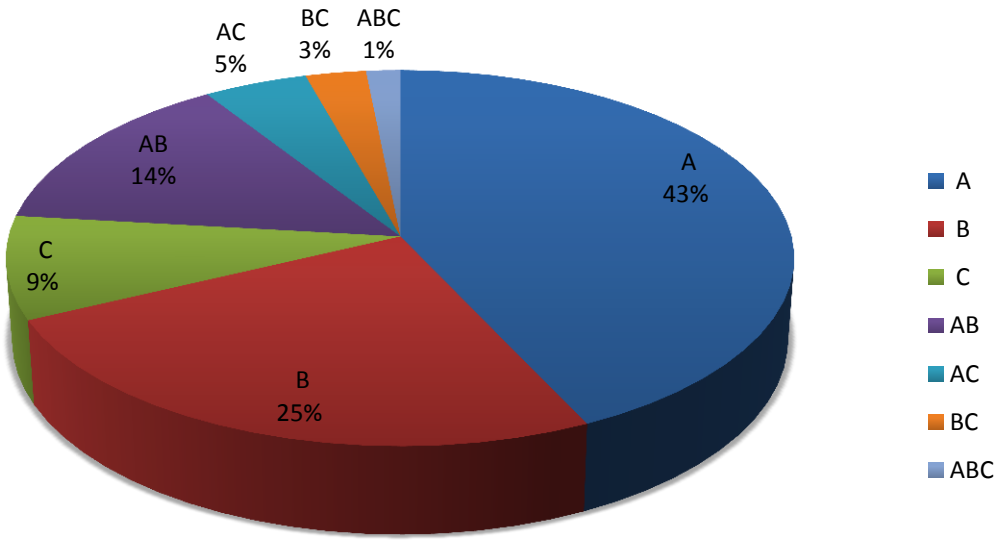


Figura 5.24: Influências dos fatores da avaliação do ECC baseado na biblioteca Relic em diferentes ambientes de execução (A – Ambiente; B – Tamanho da chave; C – Tamanho da mensagem)

5.6 Estudo de Caso 5: Criptografia e Comunicação

Os estudos de caso apresentados anteriormente não abordavam comunicação pois o objetivo era avaliar apenas o desempenho de algoritmos criptográficos com fatores que pudessem influenciar no seu tempo médio de execução. Sabe-se que comunicação sem fio está condicionada a diversas influências do ambiente no qual está inserida. Obstáculos como paredes, aparelhos eletrônicos, motores elétricos e antenas são alguns exemplos de elementos que podem influenciar no sinal de uma rede sem fio.

Este estudo de caso conduzirá testes com dois algoritmos simétricos avaliados no estudo de caso 1 (seção 5.2), porém criptografando uma troca de mensagens entre dois *kits* Gumstix Overo EVM. É importante ressaltar que os resultados apresentados neste estudo de caso foram aceitos para publicação em um evento brasileiro da área: Salla et al. (2012).

5.6.1 Domínio da Aplicação

O objetivo deste estudo de caso é verificar a influência da comunicação no tempo médio de execução do envio de mensagens entre duas entidades comunicantes. O ambiente de testes será melhor detalhado na seção 5.6.2. Outro quesito a ser avaliado é o desempenho dos algoritmos, que de acordo com os gráficos apresentados no estudo de caso 1 (seção 5.2), apresentaram resultados similares: AES 256 *bits* e RC2 64 *bits*. O segundo teve desempenho levemente inferior ao primeiro como mostrou a Figura 5.11.

Antes de prosseguir, será necessário apresentar o ambiente de testes e explicar como funciona a comunicação entre as duas entidades.

5.6.2 Configuração do Ambiente de Testes

O ambiente de testes destes experimentos utiliza duas unidades do *kit* Gumstix Overo EVM com as mesmas características de *hardware* e *software*. O sistema operacional utilizado é o mesmo dos demais estudos de caso, que pode ser visto na seção 4.2. O que difere este estudo dos demais é que os testes foram executados enviando informações criptografadas de uma entidade para a outra, nomeadas no escopo deste estudo de caso como E1 e E2.

A comunicação se da via *sockets* implementados em C. E1 assume o papel de servidor, cria e ativa um *socket* em um endereço específico, busca por conexões e as aceita caso estas façam uma requisição. E2 assume o papel de cliente, criando um *socket* e conectando-se ao servidor E1. A partir deste momento E1 e E2 podem trocar informações com funções de leitura, escrita e envio.

Os experimentos executados descartam o tempo de estabelecimento de uma conexão entre E1 e E2 e prendem-se à avaliação do tempo necessário para criptografar e enviar as mensagens confidenciais. Isso será melhor explicado na próxima seção.

5.6.3 Planejamento de Experimentos

Nesta avaliação de desempenho serão considerados dois fatores. O primeiro deles é o algoritmo, que terá dois níveis: AES 256 *bits* e RC2 64 *bits*. O segundo fator é a distância entre as entidades E1 e E2, que pode assumir os valores nomeados como Próximas e Distantes, os quais correspondem a 1m e 9m de distância, respectivamente. A Tabela 5.8 mostra a configuração destes experimentos.

Tabela 5.8: Configurações dos experimentos de avaliação de desempenho de comunicação sem fio segura

Exp.	Fator A (Algoritmo)	Fator B (Distância)
1	AES 256 *bits*	Próximas
2	AES 256 *bits*	Distantes
3	RC2 64 *bits*	Próximas
4	RC2 64 *bits*	Distantes

O tamanho da mensagem trocada pelas entidades é de 100KB. A adoção de um tamanho pequeno de mensagem busca simular o envio de instruções de controle para um VANT ou VTNT (Veículo Terrestre Não Tripulado), ou ainda entre dois veículos em uma VANET,

por exemplo, situando o estudo de caso em cenários mais próximos da realidade. Os experimentos foram executados 15 vezes e a variável de resposta é o tempo médio necessário para criptografar os dados em E1, enviar o texto cifrado para E2 e decriptografá-lo em E2.

É importante ressaltar que as influências de fatores externos (elementos do ambiente) não foram consideradas. Os testes foram executados em um laboratório e, portanto, o ambiente e as condições de fatores externos foram as mesmas para todos os experimentos.

5.6.4 Análise de Resultados

Os resultados destes experimentos estão ilustrados no gráfico da Figura 5.25.

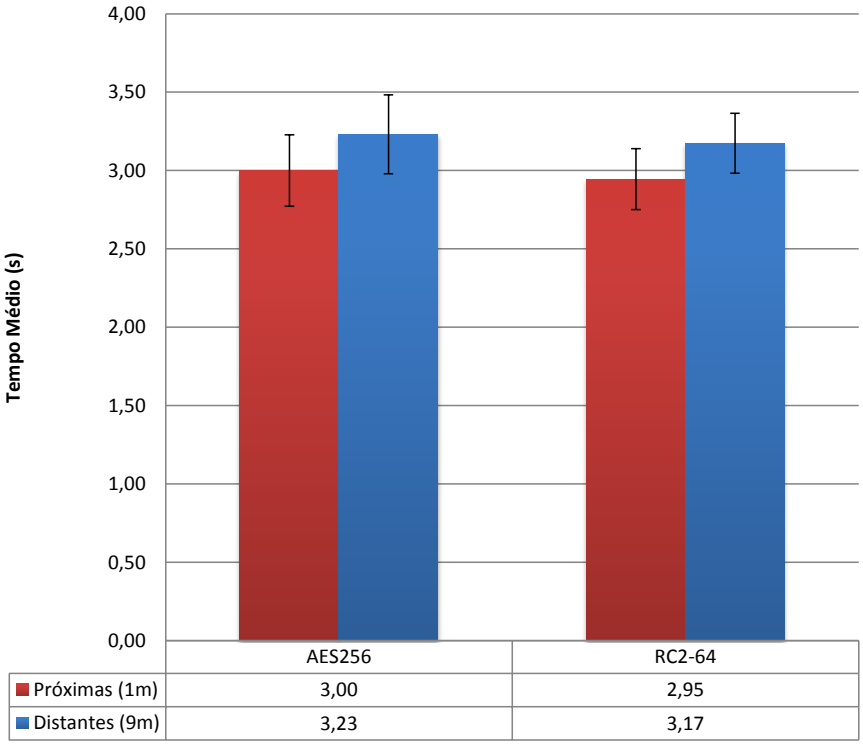


Figura 5.25: Tempo médio de execução de experimentos com comunicação sem fio segura via *Wi-Fi*

Estatisticamente não é possível afirmar que há diferenças entre os resultados apresentados, apesar de o tempo médio de execução de ambos os algoritmos ter ficado maior para os testes em que as entidades E1 e E2 estão mais distantes uma da outra. Esta grande variação dos resultados que fez com que o intervalo de confiança ficasse grande se deve justamente à influência de fatores externos ao ambiente de experimentações, que degradam o sinal de redes sem fio.

Estes aspectos relativos a influências de redes sem fio não foram abordados, pois fogem dos objetivos do trabalho que se preocupa em avaliar formas de se assegurar a comunicação. Influências externas precisam ser consideradas em aplicações finais, pois as condições do local de execução dos sistemas embarcados deverá influenciar no sinal *Wi-Fi*.

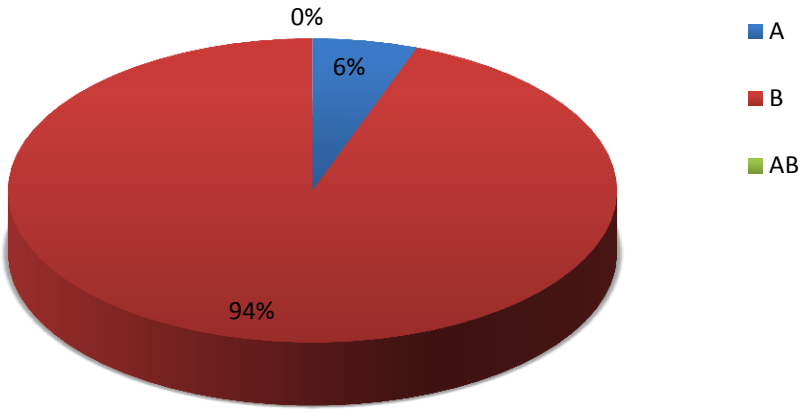


Figura 5.26: Influências dos fatores na avaliação de desempenho de comunicação sem fio segura (A – Algoritmo; B – Distância)

No tocante aos algoritmos de criptografia, os tempos médios de execução não apresentaram variações grandes, como visto no estudo de caso 1, o que leva a concluir que, no estudo de caso atual, a responsabilidade pelo aumento na variação dos resultados é, em grande parte, da rede.

O cálculo da influência dos fatores, apresentado na Figura 5.26, aponta que a distância (fator B) foi o que mais influenciou os resultados, com 94%, um valor considerável. Com este fator certamente surgem elementos externos ao ambiente no qual a comunicação foi estabelecida e uma avaliação detalhada das principais influência em comunicações que se utilizam de redes sem fio poderia ser conduzida em trabalhos futuros (ver seção 6.4. O segundo fator a influenciar foi o algoritmo com 6%. A associação dos fatores A e B não exerceu influência.

5.7 Considerações Finais

Este capítulo apresentou estudos de caso que avaliaram conjuntos de algoritmos de criptografia quanto a aspectos de desempenho. Em alguns casos, avaliações e comparações de desempenho foram conduzidas, variando-se fatores inerentes aos contexto aos quais se aplicam os algoritmos e apresentando as influências de cada fator nos resultados obtidos.

Discussões a respeito da escolha dos algoritmos mais apropriados em cada tipo de situação foram apresentadas. Algoritmos simétricos provaram ser melhores em termos de desempenho quando comparados a algoritmos assimétricos, como preconizado pela literatura. Todos os experimentos foram realizados com vista a ambientes embarcados, arquitetura foco deste trabalho.

Capítulo

6

Conclusão

A popularização e a disseminação do uso de sistemas embarcados é bastante notável atualmente. A motivação para o uso destes sistemas vai desde questões de conforto pessoal, residencial e urbano, até operações mais sofisticadas, que demandam sistemas desenvolvidos especificamente para atender a certos requisitos e que facilitam operações que antes eram desempenhadas quase que completamente por seres humanos. Neste último caso tem-se os sistemas autônomos como exemplo, que vêm crescendo em utilização devido à capacidade de auxiliar e automatizar operações militares, ambientais e de segurança.

Sistemas embarcados com características que incluem altos custos de fabricação e riscos de perdas financeiras e humanas são conhecidos como críticos. Estes sistemas costumam operar sob condições climáticas extremas, têm normalmente limitações de recursos computacionais e operam, na maioria das vezes, com tarefas de tempo real. Por estes motivos é essencial que seu funcionamento seja assegurado, evitando prejuízos e permitindo a execução de tarefas pré-programadas de maneira completa e bem sucedida. Um ponto crítico nestes sistemas é a troca segura de informações entre dois ou mais sistemas, bem como com suas bases de controle.

Tendo em vista que uma das mais frequentes formas de se assegurar uma comunicação se dá por meio do emprego de criptografia e esta, por sua vez, gera uma sobrecarga computacional devido aos intensos cálculos executados, avaliações de algoritmos de segurança em ambientes com limitações de processamento são necessárias para se obter resultados que auxiliem na escolha dos algoritmos e tamanhos de chave mais apropriados. Também é

necessário levar em consideração características específicas de cada cenário onde se busca inserir criptografia em canais de comunicação.

Sendo assim, o objetivo principal desta dissertação de mestrado incluiu a aplicação de técnicas estatísticas de avaliação de desempenho para verificar a degradação no tempo de execução de tarefas de criptografia e identificar os algoritmos que apresentam melhor *throughput* médio em ambientes embarcados. Para isso, um levantamento dos principais algoritmos de criptografia foi realizado com base na literatura recente. O ambiente de testes adotado, um *kit* Gumstix Overo EVM, tem características muito próximas de um sistema embarcado crítico real, tendo sido inclusive aplicado em sistemas reais, o que auxiliou a determinar um ambiente de experimentações que vai ao encontro da proposta deste trabalho.

Os resultados foram analisados em conjuntos organizados em estudos de caso. Cada estudo de caso apresenta comparações ou avaliações de desempenho de algoritmos simétricos e assimétricos, variando os tamanhos de chave para os algoritmos aplicáveis, variando o montante de dados para processamento e variando fatores específicos para cada cenário assumido nos estudos de caso. Ao fim de cada avaliação/comparação de desempenho, cálculos para determinar a influência exercida por cada fator nos resultados obtidos foram efetuados de modo a visualizar quais são os que mais geram impacto nas condições propostas.

6.1 Dificuldades Relacionadas ao Projeto

A principal dificuldade quando se trabalha com segurança de redes de computadores e mais especificamente com criptografia está na validação de algoritmos desenvolvidos. Quando um algoritmo novo é criado, normalmente submete-se o mesmo à avaliação de uma comunidade, que vai identificar possíveis falhas de segurança naquele algoritmo. Este procedimento é muito demorado e inviável para uma dissertação de mestrado, principalmente devido ao curto período de tempo definido para o desenvolvimento do trabalho. Sendo assim, uma saída para este problema foi a adoção de uma ferramenta que disponibilizasse algoritmos de criptografia já implementados e validados. No caso deste trabalho adotou-se o *toolkit* OpenSSL, o qual é amplamente empregado em trabalhos da literatura e apresenta um bom desempenho.

Uma segunda dificuldade durante o projeto foi a instalação do sistema operacional no *kit* Gumstix Overo EVM. A documentação fornecida pela empresa fabricante é falha em alguns aspectos fundamentais, o que exigiu que materiais de apoio externos aos recomendados pela empresa fossem buscados de modo a sanar alguns problemas durante a fase de preparação do ambiente de experimentações.

Outra dificuldade encontrada foi na obtenção de implementações de criptografia de curva elíptica que já estivessem validadas. Como isto não foi possível, adotou-se dois algoritmos desenvolvidos durante o trabalho de conclusão de curso de Silva (2011), que provê implementações de um mesmo algoritmo com bibliotecas de apoio diferentes.

Na etapa experimental, não foi possível efetuar um estudo de caso que contemplasse a movimentação dos *kits* Gumstix Overo EVM, de modo a simular situações reais onde os elementos estão em movimento. O que impediu a realização destes testes foi o tempo necessário para aquisição de baterias apropriadas para proverem alimentação durante os testes. Até a conclusão desta dissertação, as baterias não puderam ser adquiridas e, portanto, o estudo de caso que assumiria o movimento como um fator de influência nos resultados não foi possível de ser realizado.

E um último problema de ordem menor foi relacionado ao *software* que auxiliava na comunicação serial com os *kits*. O Minicom não exibe com precisão alguns caracteres, ocultando parte deles e apresentando algumas falhas após um certo tempo de uso. O problema foi resolvido adotando-se o *software* Kermit, também sugerido pela *Wiki* disponibilizada pela empresa fabricante em seu site oficial.

6.2 Contribuições

Este trabalho apresentou contribuições para a área de segurança em sistemas embarcados, levando em consideração principalmente aqueles que se encaixam na categoria de ordem crítica. Veículos aéreos e terrestres não tripulados são hoje muito utilizados para operações em áreas de risco ou de difícil acesso e, portanto, encontrar soluções de segurança da comunicação para o contexto deste tipo de sistemas é fundamental para auxiliar em aplicações futuras.

A principal contribuição deste trabalho é a realização de experimentos em protótipo real. Quando adota-se simulação, a fidelidade dos resultados pode não ser a mesma de acordo com a ferramenta utilizada. Por esse motivo adotou-se o *kit* Gumstix Overo EVM, que tem aplicação prática na área de sistemas embarcados críticos (como visto na seção 4.2) para ser o ambiente de experimentações.

Durante a fase de preparação do ambiente de experimentações, identificou-se algumas falhas na documentação oferecida pela empresa fabricante do *kit* Gumstix Overo EVM, como mencionado anteriormente. Devido a isto, um tutorial próprio de implementação do ambiente foi desenvolvido de modo a documentar os passos fundamentais para se obter a mesma configuração utilizada neste trabalho e permitir a reprodução do mesmo em trabalhos futuros.

Os resultados apresentados em forma de estudos de caso foram dispostos em gráficos que auxiliam na percepção dos algoritmos que apresentam melhor desempenho. Estas

técnicas estatísticas ajudam a extrair informações mais apuradas dos resultados obtidos, apresentando desvio padrão e influências dos fatores elencados para avaliação, o que auxilia na identificação de possíveis gargalos ou problemas para os quais pode-se direcionar maior atenção.

Esta dissertação de mestrado proveu também resultados concretos que podem auxiliar na escolha dos algoritmos mais indicados para o propósito final de determinados cenários reais. Estes resultados podem ser incorporados pelo INCT-SEC (Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos) para a resolução de problemas inerentes a sistemas embarcados críticos, área direta de atuação do instituto, órgão ao qual o Laboratório de Sistemas Embarcados Críticos (local de desenvolvimento deste projeto) está vinculado.

E por fim, este trabalho teve contribuições na formação do aluno de mestrado no tocante à gerência de projetos acadêmicos. Foi possível ao aluno participar ativamente na elaboração de seis projetos de iniciação científica (IC) e dois projetos de trabalho de conclusão de curso (TCC), havendo um acompanhamento de todas as fases de execução dos mesmos, até a obtenção de dados para escrita de artigos científicos que geraram publicações e participações em eventos. Esta relação próxima com alunos de graduação ajuda na formação de um mestre que tenha condições de auxiliar nas principais questões relativas às fases inerentes ao desenvolvimento de projetos de IC e TCC.

6.3 Produção Científica

Esta seção visa apresentar a produção científica proveniente deste trabalho e também de trabalhos de iniciação científica e de conclusão de curso correlacionados. Mostra ainda uma palestra e um minicurso com temas ligados à proposta desta dissertação que foram apresentados no ano de 2011.

6.3.1 Artigos

- SALLA, G. C.; SARTIN, A. M.; SILVA, N. B. F.; PIGATTO, D. F.; BRANCO, K. R. L. J. C. "Performance Evaluation of Security Communication in Critical Embedded Systems". In: II Brazilian Conference on Critical Embedded Systems (CBSEC), 2012, Campinas, SP. (Aceito para publicação).

- PIGATTO, D. F.; SILVA, N. B. F.; BRANCO, K. R. L. J. C. "Performance Evaluation and Comparison of Algorithms for Elliptic Curve Cryptography with El-Gamal based on MIRACL and RELIC Libraries". Journal of Applied Computing Research (JACR), v. 1, p. 95-103, 2011.

- PIGATTO, D. F.; SILVA, N. B. F.; SIKANSI, F. E. G.; BRANCO, K. R. L. J. C. "Aplicação de Criptografia e Assinatura Digital para Prover Comunicação Segura em Veículos Terrestres Não Tripulados". In: II Escola Regional de Alto Desempenho de São Paulo (ERAD-SP 2011), 2011, São José dos Campos.

- PIGATTO, D. F.; SILVA, N. B. F.; BRANCO, K. R. L. J. C. "Avaliação e Comparação de Desempenho entre Algoritmos de Criptografia de Curva Elíptica com El-Gamal baseados nas Bibliotecas MIRACL e RELIC". In: Escola Regional de Redes de Computadores (ERRC), 2011, São Leopoldo, RS.

- RODRIGUES, D. ; ESTRELLA, J. C. ; PIGATTO, D. F. ; BRANCO, K. R. L. J. C. . Performance Evaluation of Security Techniques in Web Services. In: 13th International Conference on Information Integration and Web-based Applications and Services (iiWAS), 2011, Ho Chi Minh City, Vietnam. 13th International Conference on Information Integration and Web-based Applications and Services (iiWAS), 2011.

- SCHOABA, V. ; SIKANSI, F. E. G. ; PIGATTO, D. F. ; BRANCO, K. R. L. J. C. ; BRANCO, L. C. . Digital Signature for Mobile Devices: A New Implementation and Evaluation. International Journal of Future Generation Communication and Networking, v. 4, p. 3-3, 2011.

- SCHOABA, V. ; SIKANSI, F. E. G. ; PIGATTO, D. F. ; BRANCO, K. R. L. J. C. ; BRANCO, L. C. . DISIMOD Digital Signature for Mobile Devices. In: International Conference on Convergence and Hybrid Information Technology (IJFGCN 2010), 2010, Daejeon. International Conference on Convergence and Hybrid Information Technology, 2010.

6.3.2 Minicurso e Palestra

- PIGATTO, D. F.; EUSTAQUIO, P. S. F. "Avaliação de Desempenho em Sistemas Computacionais: técnicas estatísticas de planejamento, análise e apresentação de resultados". In: XV Semana Acadêmica de Ciência da Computação, URI, Erechim, RS, 2011. (Minicurso).

- PIGATTO, D. F. Segurança da Informação para Sistemas Embarcados Críticos. In: XV Semana Acadêmica de Ciência da Computação, URI, Erechim, RS, 2011. (Palestra).

6.3.3 Resumos

- SILVA, N. B. F.; PIGATTO, D. F.; BRANCO, K. R. L. J. C. "Implementações de Criptografia Utilizando Curvas Elípticas". In: XIX Congresso de Iniciação Científica da UFSCar (Universidade Federal de São Carlos), 2011, São Carlos, SP.

- SIKANSI, F. E. G.; PIGATTO, D. F.; BRANCO, K. R. L. J. C. "Aplicação de Criptografia e Assinatura Digital para Comunicação Segura em Veículos Terrestres Não Tripulados (VTNTs)". In: V Workshop de Iniciação Científica e Tecnológica de Computação, 2011, São Carlos, SP.

- SILVA, N. B. F.; BRANCO, K. R. L. J. C.; PIGATTO, D. F. "Comparação entre Implementações de Criptografia de Curva Elíptica baseada no método de El-Gamal com as Bibliotecas MIRACL e RELIC". In: 19º Simpósio Internacional de Iniciação Científica (SIICUSP), 2011, São Carlos, SP.

- RODRIGUES, D. ; PIGATTO, D. F. ; ESTRELLA, J. C. ; BRANCO, K. R. L. J. C. . Comparison and Analysis of Cryptographic Algorithms Aiming Performance Improvement in Secure Web Services. IEEE 13th International Symposium on High-Assurance Systems Engineering (HASE), 2011, 2011 (Pôster).

6.4 Trabalhos Futuros

Sistemas embarcados críticos compõem uma área de intensas pesquisas atualmente, e por isso, várias são as possibilidades de extensão deste trabalho. Algumas sugestões são:

- Ampliar a avaliação de desempenho para algoritmos de criptografia menos utilizados, mas que podem ser suficientes para alguns casos em que pode-se aplicar uma segurança menos eficiente em prol do desempenho.

- Executar novamente os testes deste trabalho porém incluindo o fator movimento enquanto dois *kits* Gumstix Overo EVM se comunicam, podendo calcular a influência do mesmo nos resultados finais. Isso aborda a questão da movimentação que é muito comum em sistemas como VANTs, VTNTs e em VANETs.

- Utilizar protocolos de comunicação como *Bluetooth* e *ZigBee* para a troca de dados entre os *kits* Gumstix Overo EVM. Um comparativo pode ser estabelecido entre eles e a *Wi-Fi* para verificar qual deles é mais suscetível a influências e qual apresenta o melhor desempenho.

- Efetuar um estudo sobre ataques de segurança específicos para sistemas embarcados críticos, efetuando testes práticos para verificar a dificuldade em se invadir um sistema como este e quais são as principais vulnerabilidades que eles apresentam, podendo inclusive efetuar um comparativo com a computação tradicional.

Referências

AL-ROUSAN, M.; RJOUB, A.; BASET, A. A Low-Energy Security Algorithm for Exchanging Information in Wireless Sensor Networks. *Journal of Information Assurance and Security 4*, p. 48–59, 2009.

ARANHA, D. F.; GOUVÊA, C. P. L. RELIC is an Efficient LIbrary for Cryptography. 2011. Disponível em: `http://code.google.com/p/relic-toolkit/`

ARM Classic Processors. 2012.
Disponível em: `http://www.arm.com/products/processors/classic/index.php`

ARMOUSH, A.; SALEWSKI, F.; KOWALEWSKI, S. Design Pattern Representation for Safety-Critical Embedded Systems. *Journal of Software Engineering and Applications*, v. 02, n. 01, p. 1–12, 2009.
Disponível em: `http://www.scirp.org/Journal/PaperDownload.aspx?paperID=403&fileName=JSEA20090100001_22902584.pdf`

BARR, M. *Programming embedded systems in C and C++*. O'Reilly Series. O'Reilly, 1999.
Disponível em: `http://books.google.com.br/books?id=a2Q6U0b36rMC`

BARR GROUPS Embedded Systems Experts. 2012.
Disponível em: `http://www.barrgroup.com/Embedded-Systems/Glossary-E`

BERGER, A. S. *Embedded systems design: an introduction to processes, tools, and techniques*. Embedded Systems Series. CMP Books, 237 p., 2002.
Disponível em: `http://books.google.com.br/books?id=3vY35UkvXrAC`

BISHOP, M. A. *Introduction to computer security*. Addison-Wesley, 2005.
Disponível em: `http://books.google.com.br/books?id=Z-lQAAAAMAAJ`

BLOOMFIELD, R.; CAZIN, J.; CRAIGEN, D.; JURISTO, N.; KESSELER, E.; VOAS, J. Validation, Verification and Certification of Embedded Systems. 2005.

BRÄNDLE, M.; NAEDELE, M. Security for Process Control Systems: An Overview. *IEEE Security and Privacy*, v. 6, n. 6, p. 24–29, 2008.
Disponível em: http://dx.doi.org/10.1109/MSP.2008.150

ÇAYIRCI, E.; RONG, C. *Security in Wireless Ad Hoc and Sensor Networks*. John Wiley & Sons, 280 p., 2009.
Disponível em: http://books.google.com.br/books?id=3EvhTrocBZUC

CERTICOM CORP. *SEC 1: Elliptic Curve Cryptography*. Relatório Técnico, 2000.
Disponível em: http://www.secg.org/collateral/sec1_final.pdf

COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems: concepts and design*. International computer science series. Addison-Wesley, 927 p., 2005.
Disponível em: http://books.google.com.br/books?id=8N9QAAAAMAAJ

ELMINAAM, D. S. A.; ABDUAL-KADER, H. M.; HADHOUD, M. M. Evaluating The Performance of Symmetric Encryption Algorithms. *I. J. Network Security*, v. 10, n. 3, p. 216–222, 2010.
Disponível em: http://dblp.uni-trier.de/db/journals/ijnsec/ijnsec10.html#ElminaamKH10

ERTAUL, L.; LU, W. ECC Based Threshold Cryptography for Secure Data Forwarding and Secure Key Exchange in MANET (I). In: *NETWORKING'05*, 2005, p. 102–113.

FAN, J.; SAKIYAMA, K.; VERBAUWHEDE, I. Elliptic curve cryptography on embedded multicore systems. *Design Automation for Embedded Systems*, v. 12, n. 3, p. 231–242, 2008.
Disponível em: http://dx.doi.org/10.1007/s10617-008-9021-3http://www.springerlink.com/index/10.1007/s10617-008-9021-3

FELDHOFER, M.; DOMINIKUS, S.; WOLKERSTORFER, J. Strong Authentication for RFID Systems Using the AES Algorithm. In: *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, 2004, p. 357–370.

GARFINKEL, S.; SPAFFORD, G.; SCHWARTZ, A. *Practical Unix and Internet Security*. Practical Series. O'Reilly, 954 p., 2003.
Disponível em: http://books.google.com.br/books?id=t0IExLP-MPMC

GILL, H. Challenges for critical embedded systems. In: *Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on*, 2005, p. 7–9.

GUMSTIX Create a Bootable MicroSD Card. 2011a.
Disponível em: http://gumstix.org/create-a-bootable-microsd-card.html

GUMSTIX Gumstix User Wiki. 2011b.
Disponível em: http://wiki.gumstix.org/index.php?title=Main_Page

GUMSTIX Installing Ubuntu 10.04 on Gumstix Overo. 2011c.
Disponível em: http://wiki.gumstix.org/index.php?title=Installing_Ubuntu_10.04_on_Gumstix_Overo

GUMSTIX Setting up a Serial Connection. 2011d.
Disponível em: http://gumstix.org/connect-to-my-gumstix-system.html

GUMSTIX Overo® EVM pack. 2012.
Disponível em: https://www.gumstix.com/store/product_info.php?products_id=253

GURA, N.; PATEL, A.; WANDER, A.; EBERLE, H.; SHANTZ, S. C. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. *Lecture notes in computer science*, p. 119–132, 2004.
Disponível em: http://cat.inist.fr/?aModele=afficheN&cpsidt=16075869

HABIB, M.; MEHMOOD, T.; ULLAH, F.; IBRAHIM, M. Performance of WiMAX Security Algorithm (The Comparative Study of RSA Encryption Algorithm with ECC Encryption Algorithm). In: *Computer Technology and Development, 2009. ICCTD '09. International Conference on*, 2009, p. 108–112.

HARTENSTEIN, H.; LABERTEAUX, K. P. A tutorial survey on vehicular ad hoc networks. *Communications Magazine, IEEE*, v. 46, n. 6, p. 164–171, 2008.

HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 856 p., 2011.
Disponível em: http://books.google.com.br/books?id=gQ-fSqbLfFoC

HENZINGER, T.; SIFAKIS, J. The embedded systems design challenge. In: *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, Springer, 2006.
Disponível em: http://chess.eecs.berkeley.edu/pubs/264.html

HODJAT, A.; HWANG, D. D.; LAI, B.; TIRI, K.; VERBAUWHEDE, I. A 3.84 gbits/s AES crypto coprocessor with modes of operation in a 0.18-μm CMOS technology. In: *Proceedings of the 15th ACM Great Lakes symposium on VLSI*, New York, NY, USA: ACM,

2005, p. 60–63 (*GLSVLSI '05*, v..).
Disponível em: http://doi.acm.org/10.1145/1057661.1057677

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley, 685 p., 1991.
Disponível em: http://books.google.com.br/books?id=CN1QAAAAMAAJ

JANUZAJ, V.; KUGELE, S.; LANGER, B.; SCHALLHART, C.; VEITH, H. New Challenges in the Development of Critical Embedded Systems—An "aeromotive" Perspective. In: MARGARIA, T.; STEFFEN, B., eds. *Leveraging Applications of Formal Methods, Verification, and Validation*, v. 6415 de *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 1–2, 2010.
Disponível em: http://dx.doi.org/10.1007/978-3-642-16558-0_1

JENA, D.; JENA, S. K. A novel and efficient cryptosystem for large message encryption. *Int. J. Inf. Commun. Techol.*, v. 3, n. 1, p. 32–39, 2011.
Disponível em: http://dx.doi.org/10.1504/IJICT.2011.039521

JUST, M. Cryptography III: Symmetric Ciphers. 2012.
Disponível em: http://www.inf.ed.ac.uk/teaching/courses/cs/0910/lecs/symmetric-6up.pdf

KAKANAKOV, N. R. Experimental Analysis of Client/Server Applications in Embedded Systems. In: *ELECTRONICS'2005*, 2005, p. 97–102.

KAKANAKOV, N. R.; SPASOV, G. Adaptation of web service architecture in distributed embedded systems. In: *International Conference on Computer Systems and Technologies - CompSysTech*, 2005, p. 10–16.

KAMIENSKI, C. A.; SADOK, D.; CAVALCANTI, D. A. T.; SOUSA, D. M. T.; DIAS, K. L. Simulando a Internet: Aplicações na pesquisa e no ensino. 2002.
Disponível em: http://www.cin.ufpe.br/~cak/publications/jai2002-capitulo2.pdf

KOCHER, P.; LEE, R.; MCGRAW, G.; RAGHUNATHAN, A. Security as a new dimension in embedded system design. In: *Proceedings of the 41st annual Design Automation Conference*, New York, NY, USA: ACM, 2004, p. 753–760 (*DAC '04*, v..).
Disponível em: http://doi.acm.org/10.1145/996566.996771

KOOPMAN, P. Embedded System Security. *Computer*, v. 37, n. 7, p. 95–97, 2004.
Disponível em: http://dx.doi.org/10.1109/MC.2004.52

KUROSE, J. F.; ROSS, K. W.; MARQUES, A. S. *Redes de computadores e a Internet: uma nova abordagem*. Addison Wesley, 548 p., 2003.
Disponível em: `http://books.google.com.br/books?id=AJc8AgAACAAJ`

LENSTRA, A. K.; VERHEUL, E. R. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, v. 14, p. 255–293, 1999.

MARTINEZ, K.; BASFORD, P.; ELLUL, J.; CLARKE, R. Field Deployment of Low Power High Performance Nodes. In: *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*, 2010, p. 199–205.

MINAAM, D. S. A.; ABDUAL-KADER, H. M.; HADHOUD, M. M. Evaluating the Effects of Symmetric Cryptography Algorithms on Power Consumption for Different Data Types. *International Journal of Network Security*, v. 11, n. 2, p. 78–87, 2010.

MONTGOMERY, D. C.; CUSTER, L.; MCCARVILLE, D. R. *Design and Analysis of Experiments, Student Solutions Manual*. John Wiley & Sons, 216 p., 2002.
Disponível em: `http://books.google.co.in/books?id=N0wOAAAACAAJ`

MORENO, E. D.; CHIARAMONTE, F. D.; PEREIRA, R. B. *Criptografia em Software e Hardware*. São Paulo: Novatec, 288 p., 2005.

MORITZ, G.; PRUTER, S.; TIMMERMANN, D.; GOLATOWSKI, F. Web services on deeply embedded devices with real-time processing. In: *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, 2008, p. 432–435.

NADEEM, A.; JAVED, M. A Performance Comparison of Data Encryption Algorithms. In: *2005 International Conference on Information and Communication Technologies*, IEEE, 2005, p. 84–89.
Disponível em: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1598556`

NIST Data Encryption Standard (DES). 1993.
Disponível em: `http://www.itl.nist.gov/fipspubs/fip46-2.htm`

NIST Advanced Encryption Standard (AES). 2001.
Disponível em: `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

NIST NIST: FIPS PUB 186-3 - Digital Signature Standard (DSS). 2009.

NOERGAARD, T. *Embedded systems architecture: a comprehensive guide for engineers and programmers*. Embedded technology series. Elsevier/Newnes, 640 p., 2005.
Disponível em: `http://books.google.com.br/books?id=M0g0GjY5IWQC`

OLIVEIRA, A. S.; ANDRADE, F. S. *Sistemas Embarcados: Hardware e Firmware na Prática*. São Paulo: ERICA, 320 p., 2006.
Disponível em: http://books.google.com.br/books?id=u9KVGAAACAAJ

OPENSSL Crypto (OpenSSL cryptographic library). 2012a.
Disponível em: http://www.openssl.org/docs/crypto/crypto.html

OPENSSL OpenSSL: The Open Source toolkit for SSL/TLS. 2012b.
Disponível em: http://www.openssl.org/

PENG, Z.; FANG, J. J. Comparing and implementation of public key cryptography algorithms on smart card. In: *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, 2010, p. V12–508 –V12–510.

PHANG, S. K.; ONG, J. J.; YEO, R. T. C.; CHEN, B. M.; LEE, T. H. Autonomous Mini-UAV for indoor flight with embedded on-board vision processing as navigation system. In: *Computational Technologies in Electrical and Electronics Engineering (SIBIRCON), 2010 IEEE Region 8 International Conference on*, 2010, p. 722–727.

PIGATTO, D. F.; SILVA, N. B. F. D.; BRANCO, K. R. L. J. C. Avaliação e Comparação de Desempenho entre Algoritmos de Criptografia de Curva Elíptica com El-Gamal baseados nas Bibliotecas MIRACl e RELIC. In: *9ª Escola Regional de Redes de Computadores (ERRC 2011)*, São Leopoldo, RS: Sociedade Brasileira de Computação (SBC), 2011, p. 113–116.
Disponível em: www.unisinos.br/errc2011

PIGATTO, D. F.; SILVA, N. B. F. D.; BRANCO, K. R. L. J. C. Performance Evaluation and Comparison of Algorithms for Elliptic Curve Cryptography with El-Gamal based on MIRACL and RELIC Libraries. *Journal of Applied Computing Research*, v. 1, n. 2, p. 95–103, 2012.
Disponível em: http://www.unisinos.br/revistas/index.php/jacr/article/view/1789

PIRES, R. D. M. *Aplicação do Algoritmo Diffie-Hellman no Compartilhamento de Volumes Criptografados do TrueCrypt*. Projeto final de curso, Universidade Federal de Goiás - Campus Catalão, 2010.
Disponível em: http://www2.catalao.ufg.br/uploads/files/3/monografias/2010/Rayner2010.pdf

POTLAPALLY, N. R.; RAVI, S.; RAGHUNATHAN, A.; JHA, N. K. Analyzing the energy consumption of security protocols. In: *Proceedings of the 2003 international symposium on Low power electronics and design*, New York, NY, USA: ACM, 2003, p. 30–35 (*ISLPED*

'03, v..).
Disponível em: http://doi.acm.org/10.1145/871506.871518

RAMACHANDRAN, A.; ZHOU, Z.; HUANG, D. Computing Cryptographic Algorithms in Portable and Embedded Devices. In: *Portable Information Devices, 2007. PORTABLE07. IEEE International Conference on*, 2007, p. 1–7.

RAVI, S.; RAGHUNATHAN, A.; KOCHER, P.; HATTANGADY, S. Security in embedded systems: Design challenges. *ACM Trans. Embed. Comput. Syst.*, v. 3, n. 3, p. 461–491, 2004.
Disponível em: http://doi.acm.org/10.1145/1015047.1015049

SALLA, G. C.; SARTIN, A. M.; SILVA, N. B. F.; PIGATTO, D. F.; BRANCO, K. R. L. J. C. Performance Evaluation of Security Communication in Critical Embedded Systems. In: *II Brazilian Conference on Critical Embedded Systems*, Campinas, SP, 2012.
Disponível em: http://inct-sec.org/cbsec2012/

SHAMUS Miracl - Multiprecision Integer and Rational Arithmetic C/C++ Library. 2011.
Disponível em: http://www.shamus.ie/index.php?page=elliptic-curves

SHORT, M. Development guidelines for dependable real-time embedded systems. In: *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on*, 2008, p. 1032–1039.

SILVA, N. B. F. *Implementação do Algoritmo de Criptografia de Curvas Elípticas em C*. Trabalho de conclusão de curso, Universidade de São Paulo (USP) - Campus de São Carlos, 2011.

SLOSS, A. N.; SYMES, D.; WRIGHT, C. *Arm System Developer's Guide: Designing and Optimizing System Software*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier/ Morgan Kaufman, 689 p., 2004.
Disponível em: http://books.google.com.br/books?id=vdk4ZGRqMskC

STALLINGS, W. *Criptografia e seguran{ç}a de redes: princ{í}pios e pr{á}ticas*. PRENTICE HALL BRASIL, 2008.
Disponível em: http://books.google.com.br/books?id=LPZfPgAACAAJ

STAPKO, T. J. *Practical Embedded Security: Building Secure Resource-Constrained Systems*. Embedded Technology Series. Elsevier/Newnes, 2007.
Disponível em: http://books.google.com.br/books?id=Mly55VntuYMC

TANENBAUM, A. S. *Redes de Computadores*. Campus, 945 p., 2003.
Disponível em: http://books.google.com.br/books?id=0tjB8FbV590C

THRAMBOULIDIS, K. C.; DOUKAS, G.; KOUMOUTSOS, G. A SOA-based Embedded Systems Development Environment For Industrial Automation. *EURASIP Journal on Embedded Systems*, v. 2008, p. 3:1———3:15, 2008.
Disponível em: http://downloads.hindawi.com/journals/es/2008/312671.pdf

TIRI, K.; HWANG, D.; HODJAT, A.; LAI, B.; YANG, S.; SCHAUMONT, P.; VERBAUWHEDE, I. A side-channel leakage free coprocessor ic in 0.18um cmos for embedded aes-based cryptographic and biometric processing. In: *In Dac '05*, ACM Press, 2005, p. 222–227.

UBUNTU ARM/RootStock. 2011.
Disponível em: https://wiki.ubuntu.com/ARM/RootStock

UMAPARVATHI, M.; VARUGHESE, D. K. Evaluation of Symmetric Encryption Algorithms for MANETs. *Journal of Computer Science*, v. 10, n. 3, p. 10–12, 2010.

WOLF, W. H. *Computers as components: principles of embedded computing system design*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier/Morgan Kaufmann, 507 p., 2008.
Disponível em: http://books.google.com.br/books?id=AvkKJ0GocIsC

WOLLINGER, T.; GUAJARDO, J.; PAAR, C. Cryptography in Embedded Systems: An Overview. In: *in Proc. of the Embedded World 2003 Exhibition and Conference*, 2003, p. 18–20.

XU, Q.; MAK, T.; KO, J.; SENGUPTA, R. Vehicle-to-vehicle safety messaging in DSRC. In: *VANET '04: Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, New York, NY, USA: ACM, 2004, p. 19–28.
Disponível em: http://portal.acm.org/citation.cfm?id=1023875.1023879#

YOUSEFI, S.; MOUSAVI, M.; FATHY, M. Vehicular Ad Hoc Networks (VANETs): Challenges and Perspectives. In: *ITS Telecommunications Proceedings, 2006 6th International Conference on*, 2006, p. 761–766.
Disponível em: http://dx.doi.org/10.1109/ITST.2006.289012

ZHANG, Y.; CHEN, D.; CHOI, Y.; CHEN, L.; KO, S.-B. A high performance ECC hardware implementation with instruction-level parallelism over GF(2163). *Microprocess. Microsyst.*, v. 34, n. 6, p. 228–236, 2010.
Disponível em: http://dx.doi.org/10.1016/j.micpro.2010.04.006

Apêndice

A

Tutorial: Instalação de Ubuntu em uma Gumstix Overo EVM

Este tutorial explica como instalar uma versão específica para ARM do sistema operacional Ubuntu em uma Gumstix Overo EVM.

Preparação da imagem do sistema operacional

O primeiro passo é realizar a instalação do *software* que efetuará a geração da imagem do sistema operacional Ubuntu especificamente para uma arquitetura ARM:

```
$ sudo apt-get install rootstock qemu
```

Em seguida, executa-se o comando de geração da imagem com os parâmetros adequados, como exemplificado a seguir. É interessante incluir o máximo de aplicativos possível nesta etapa. Ex.: samba, x11vnc, entre outros.

```
$ sudo rootstock --serial ttyS2 --dist lucid --fqdn "gumstix" --seed lxde,
gdm,openssh-server,gcc,g++,build-essential,apt --login gumstix --password
gumstix
```

Onde:

`--serial` especifica o dispositivo (neste caso, o cartão de memória);
`--dist` especifica a versão do Ubuntu (neste caso, a 10.04 LTS);
`--fqdn` especifica o *hostname*;
`--seed` refere-se aos pacotes que serão instalados junto com o sistema operacional;
`--login` e `--password` permitem a definição prévia de usuário e senha.

A saída desta operação deverá ser um arquivo com nome similar a `armel-rootfs-201107170150.tgz`.

Preparação do cartão de memória

O cartão de memória deve ser formatado e particionado seguindo as instruções disponíveis na documentação da Gumstix (Gumstix, 2011a).

Estabelecendo comunicação serial e efetuando o processo de boot do sistema

Antes de ligar o dispositivo é necessário estabelecer uma comunicação serial para acompanhar o andamento do *boot* e poder manipular o sistema. Pode-se utilizar os *softwares* Minicom ou Kermit (Gumstix, 2011d). A figura A.1 mostra a conexão estabelecida com o dispositivo por meio do *software* Minicom.

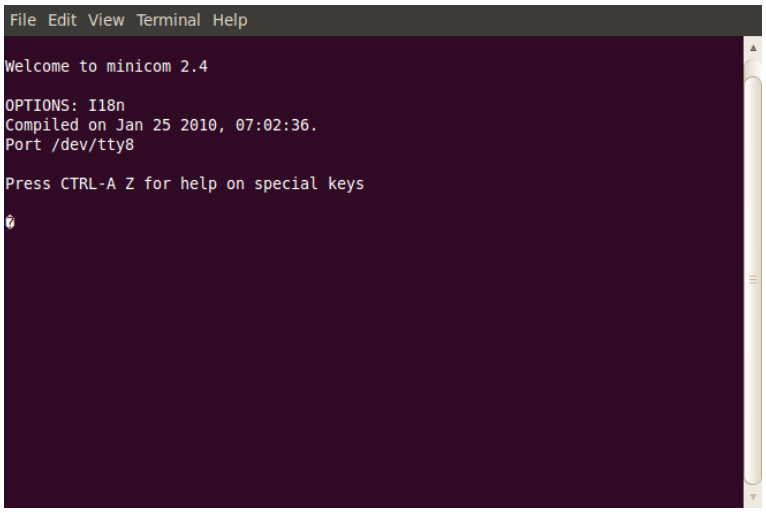


Figura A.1: Conexão estabelecida entre o *software* Minicom e o dispositivo Gumstix

Pode-se então, conectar a entrada de energia e efetuar a operação de *boot*. Ao ligar o dispositivo é possível interromper a operação de *boot* e modificar algumas variáveis do

u-boot bootloader (imagem fornecida pelo fabricante que efetua a primeira inicialização do *hardware* e, em seguida, carrega e alterna para o *kernel* do Linux).

Algumas configurações de rede poderão ser necessárias após o primeiro *boot* para que seja possível conectar-se a uma rede sem fio ou para adicionar fontes de pacotes do repositório oficial do Ubuntu. Isso vai depender da versão do sistema operacional que tiver sido adotada.

Apêndice

B

Detalhamento dos Algoritmos ECC baseados nas Bibliotecas Relic e Miracl

Algoritmo com a biblioteca Miracl

A Miracl – *Multiprecision Integer and Rational Arithmetic C/C++ Library* – é uma biblioteca proprietária, mas livre para uso educacional, produzida pela Shamus Software (Shamus, 2011). É uma ferramenta para os desenvolvedores de sistemas de criptografia que oferece as operações necessárias para lidar com números grandes, além de suporte completo a curvas elípticas.

A biblioteca permite a criação de curvas elípticas por meio dos parâmetros de definição e oferece as operações aritméticas. Além disso, é possível utilizar números grandes para a representação da mensagem e da chave privada.

No caso da Miracl a documentação não fornece a informação de como as estruturas são formadas, apenas a sua descrição. As estruturas da biblioteca utilizadas na implementação do algoritmo foram:

- `big`: um número grande.
- `epoint`: um ponto da curva elíptica.
- `miracl`: um tipo de dados que guarda informações a respeito do sistema da biblioteca.

As funções da Miracl utilizadas na implementação podem ser separadas em duas categorias. Na primeira delas estão as operações de inicialização do sistema e de manipulação de números grandes, que são:

- `mirsys`: inicializa o sistema da biblioteca para o processo atual e deve ser chamada antes da utilização de qualquer rotina da biblioteca. Retorna uma variável com informações do sistema, do tipo `miracl`.
- `irand`: inicializa o sistema interno que gera números aleatórios por meio de uma semente.
- `mirvar`: aloca espaço na memória para uma variável do tipo `big`.
- `cinstr`: converte um vetor de caracteres em uma variável do tipo `big`.
- `bigbits`: gera um número aleatório e o grava em uma variável do tipo `big` de acordo com um tamanho específico passado como parâmetro.
- `convert`: converte um inteiro para um `big`.
- `add`: adição de dois `big`.
- `subtract`: subtração de dois `big`.
- `bytes_to_big`: converte um vetor de *bytes* para um `big`.
- `big_to_bytes`: converte um número `big` em um vetor de *bytes*.
- `cotnum`: operação de saída para a escrita de uma variável do tipo `big` na tela ou em um arquivo.
- `cinnum`: operação de entrada para a leitura de uma variável do tipo `big` do teclado ou de um arquivo.
- `mirkill`: desaloca a memória usada para variáveis do tipo `big`.
- `mirexit`: desaloca a memória usada para variáveis internas usadas pelo sistema da biblioteca.

Na segunda categoria de funções da Miracl estão as operações de inicialização da curva elíptica e da manipulação de pontos da curva, que são:

- `ecurve_init`: inicializa uma curva elíptica por meio dos parâmetros `a`, `b` e `p` da equação.
- `epoint_init`: aloca espaço na memória para uma variável `epoint`.

- `epoint_set`: seta um ponto da curva elíptica em uma variável `epoint`, se possível, por meio das variáveis `x` e `y` passadas como coordenadas do ponto.
- `ecurve_mult`: multiplica uma variável `epoint` por um número grande.
- `epoint_get`: normaliza uma variável `epoint` e retorna as coordenadas `x` e `y`.
- `epoint_free`: libera a memória alocada para uma variável `epoint`.
- `epoint_x`: testa se a coordenada `x`, passada como parâmetro, é uma coordenada válida para um ponto da curva elíptica.
- `ecurve_add`: adição de duas variáveis `epoint`.
- `ecurve_sub`: subtração de duas variáveis `epoint`.

A implementação utiliza três bibliotecas, duas inerentes à linguagem C, a `stdio.h`, que fornece as funções de entrada e saída, e a `time.h`, que fornece a semente que irá gerar os números aleatórios. Utiliza-se também a biblioteca que fornece as definições da Miracl, a `miracl.h`.

O código é composto de três estruturas que definem a chave pública, a chave privada e a mensagem criptografada. A chave privada é composta apenas por uma variável do tipo `big`, já a chave pública é composta por um ponto da curva elíptica comprimido. A versão comprimida do ponto fornece uma variável `big` que representa a coordenada `x` e um inteiro que representa a variável `y`. A estrutura da mensagem criptografada também é representada pelo ponto comprimido e um inteiro que define um intervalo da coordenada `x` para se obter um ponto pertencente à curva.

As principais funções do código são a de criação das chaves, a de criptografia de um bloco de texto e a de decriptografia de um bloco de texto cifrado. Porém, antes da execução de qualquer função algumas providências são tomadas para inicializar o sistema: o sistema da biblioteca é inicializado por meio da função `mirsys`; o sistema que gera números aleatórios é inicializado por meio da função `irand`, que utiliza como semente o tempo atual; a curva elíptica é inicializada pelos parâmetros que a definem; e um ponto pertencente a ela também pré-definido é guardado em uma variável do tipo `epoint`.

Antes da criação das chaves cria-se um arquivo para a escrita da chave pública e um arquivo para a escrita da chave privada. Posteriormente são criadas as variáveis dos tipos de dados das chaves e a função para a criação das mesmas é chamada. O processo de criação das chaves consiste na geração do número grande aleatório por meio da função `bigbits` e da multiplicação desse número pelo ponto já conhecido, gerando também a chave pública. Esses valores são escritos nos arquivos correspondentes a cada chave e o programa é finalizado liberando a memória alocada.

Para a criptografia são passados, no comando de execução: o arquivo de entrada, que corresponde ao texto simples, e o arquivo de saída, que irá conter o texto cifrado. Devido à função da biblioteca não conseguir converter um vetor de *bytes* que contenha o valor `null` para um número `big`, é necessário tratar a leitura do arquivo de entrada a cada `byte` lido. As posições dos *bytes* nulos são gravadas no arquivo de saída e com os *bytes* não nulos são montados os blocos a serem criptografados.

A função de criptografia mapeia o bloco com 18 *bytes* em um número grande e verifica se existe um ponto da curva com a coordenada `x` igual ao valor desse número. Caso não exista, soma-se um ao valor da coordenada até que se encontre um ponto válido. O número de vezes que o processo de soma é executado fica gravado na estrutura da mensagem cifrada. Com o ponto obtido, a criptografia deste é feita somando-o com a multiplicação da própria chave privada pela chave pública recebida e recuperando o ponto em sua forma comprimida, que também é gravado na estrutura da mensagem cifrada. Quando a função retorna, a variável que contém a mensagem cifrada é escrita no arquivo de saída e o processo se inicia novamente até o fim da leitura do arquivo de entrada. Por fim, os arquivos são fechados e a memória alocada pelo sistema é liberada.

Para a decriptografia, feita no receptor, também são passados dois arquivos no comando de execução, um de entrada e outro de saída, mas dessa vez o arquivo de entrada corresponde ao texto cifrado e o de saída corresponde ao texto simples. O arquivo de entrada é lido e submetido a verificação para descobrir se existem posições de valores nulos ou pontos da curva elíptica. Os pontos são passados para a função de decriptografia e são subtraídos da multiplicação da própria chave privada pela chave pública do usuário que criptografou o texto. Então, se extrai o valor da coordenada `x` e se subtrai o valor gravado na mensagem para obter o número grande que representa a mensagem. O número é mapeado para o seu correspondente binário, recuperando-se o bloco original, que é gravado no arquivo de saída. Toda a operação é repetida até o fim da leitura do arquivo cifrado e, então, a memória alocada pelo sistema é liberada.

Algoritmo com a biblioteca Relic

A Relic (*Efficient LIbrary for Cryptography*) é uma biblioteca desenvolvida por pesquisadores da Universidade de Campinas com o objetivo de oferecer ferramentas criptográficas baseadas em flexibilidade e eficiência. Ela oferece implementações de aritmética de números inteiros grandes, aritmética de campos binários e primos, curvas elípticas sobre campos primos, entre outras (Aranha e Gouvêa, 2011).

Para o caso dessa biblioteca, a escolha da curva elíptica depende de uma macro cujo valor é dado antes da compilação do seu código. Com essa macro possuindo um valor válido e a biblioteca recompilada, é possível iniciar a curva efetuando a chamada de uma fun-

ção, portanto os parâmetros que a definem são determinados internamente pelo próprio sistema da biblioteca.

As estruturas da biblioteca usadas no algoritmo são:

- `bn_t`: representa um inteiro de múltipla precisão, ou seja, um inteiro muito grande. Contém um vetor de dígitos que representam o inteiro, uma variável inteira para determinar a memória alocada, uma variável inteira para determinar o número de dígitos usados e uma variável inteira que determina o sinal.
- `ep_t`: representa um ponto da curva elíptica. Contém uma variável, composta de vários dígitos, para cada coordenada, x, y e z e uma variável inteira para determinar se o ponto está normalizado.

As funções da biblioteca utilizadas no algoritmo são:

- `core_init`: inicializa o sistema da biblioteca.
- `core_clean`: finaliza o sistema da biblioteca.
- `bn_new`: aloca a memória necessária e inicializa uma variável do tipo `bn_t`.
- `bn_rand`: gera um número aleatório em uma variável do tipo `bn_t`.
- `bn_read_bin`: converte um vetor de bytes em uma variável do tipo `bn_t`.
- `bn_clean`: desaloca a memória de uma variável do tipo `bn_t`.
- `ep_param_set`: configura uma nova curva elíptica e a inicializa.
- `ep_curve_get_ord`: retorna a ordem da curva elíptica.
- `ep_null`: inicializa um ponto na curva elíptica com valor null.
- `ep_new`: aloca a memória e inicializa um ponto na curva elíptica.
- `ep_mul_gen`: multiplica o ponto gerador da curva por uma variável do tipo `bn_t`.
- `ep_curve_get_gen`: retorna o ponto gerar da curva elíptica inicializada.
- `ep_mul_basic`: multiplica um inteiro grande por um ponto da curva.
- `ep_add_basic`: adição de dois pontos da curva.
- `ep_sub_basic`: subtração de dois pontos da curva.
- `ep_free`: desaloca a memória de uma variável do tipo `ep_t`.

A implementação utiliza a biblioteca do C `stdio.h` para o uso das funções de entrada e saída e a biblioteca `relic.h` para o uso das funções da biblioteca Relic. Uma das funções da biblioteca, a `bn_write_bin`, foi reescrita para realizar a conversão de um número inteiro grande, do tipo `bn_t`, para um vetor de *bytes*.

A chave privada é definida como uma variável do tipo `bn_t`, enquanto a chave pública e a mensagem são definidas como pontos da curva elíptica, variáveis do tipo `ep_t`. As funções são divididas da mesma forma que o algoritmo da biblioteca Miracl, com uma função para a criação das chaves, uma para a criptografia de um bloco e uma para a decriptografia do bloco. Porém, para o algoritmo da biblioteca Relic também foi necessário desenvolver as funções que realizam a escrita e a leitura das variáveis `bn_t` e `ep_t` e para isso suas estruturas foram analisadas com mais rigor.

Da mesma forma que o outro algoritmo, são criados arquivos para guardar a chave pública separadamente da chave privada. Os arquivos são abertos, a função da criação das chaves gera um número aleatório por meio da função `bn_rand` para a chave privada e a multiplica pelo gerador da curva elíptica, gerando a chave pública. Depois da escrita das chaves nos respectivos arquivos, o programa libera a memória alocada pela biblioteca e pelas variáveis e finaliza.

Para a criptografia também são passados um arquivo de entrada, que representa o texto simples, e um arquivo de saída, que será o texto cifrado. O arquivo de entrada é lido e um bloco de 40 *bytes* é passado para a função de criptografia. Na função a mensagem é convertida em número grande por meio da função `bn_read_bin` e armazenada em uma variável do tipo `bn_t`. Esse valor é somado à coordenada `x` do ponto gerador da curva, que depois é somado com a multiplicação da própria chave privada pela chave pública recebida. O texto cifrado é escrito no arquivo de saída pela função criada para a escrita de variáveis do tipo `ep_t`, e o processo se repete até o fim da leitura do arquivo de entrada.

Para a decriptografia são passados, pelo comando de execução, um arquivo de entrada, que representa o texto cifrado, e um arquivo de saída, que será o texto simples recuperado. O arquivo de entrada é lido pela função criada para a leitura de variáveis do tipo `ep_t` e o ponto é passado para a função de decriptografia. Na função o ponto é subtraído pela multiplicação da própria chave privada pela chave pública de quem criptografou o arquivo e então a coordenada `x` do resultado é extraída. O valor da coordenada `x` é subtraído pela coordenada `x` do gerador e, assim, obtém-se a mensagem mapeada na variável `bn_t`. O vetor de *bytes* é recuperado pela função reescrita `bn_write_bin` da variável `bn_t` e é escrito no arquivo de saída. O processo é feito dessa maneira até o fim da leitura do arquivo de entrada, recuperando no final o arquivo que representa o texto simples original.