UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE MESTRADO EM ENGENHARIA DE PRODUÇÃO

# UM SISTEMA DISTRIBUÍDO DE CRIPTOANÁLISE COMPUTACIONAL PARA USO FORENSE

DISSERTAÇÃO DE MESTRADO

Antonio Marcos de Oliveira Candia

Santa Maria, RS, Brasil

2008

# UM SISTEMA DISTRIBUÍDO DE CRIPTOANÁLISE COMPUTACIONAL PARA USO FORENSE

**por**

**Antonio Marcos de Oliveira Candia**

Dissertação apresentada ao Curso de Mestrado em Engenharia de Produção da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

**Mestre em Engenharia de Produção, área de concentração em Tecnologia da Informação**

**Orientador: Prof. Dr. Benhur de Oliveira Stein**

**Santa Maria, RS, Brasil**

**2008**

**Universidade Federal de Santa Maria**
**Centro de Tecnologia**
**Curso de Mestrado em Engenharia de Produção**

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

# UM SISTEMA DISTRIBUÍDO DE CRIPTOANÁLISE COMPUTACIONAL PARA USO FORENSE

elaborada por
**Antonio Marcos de Oliveira Candia**

como requisito parcial para obtenção do grau de
**Mestre em Engenharia de Produção, área de concentração em Tecnologia da Informação**

## COMISSÃO EXAMINADORA:

**Prof. Dr. Benhur de Oliveira Stein**
(Presidente/Orientador)

**Prof. Dr. Raul Ceretta Nunes**

**Dr. Paulo Quintiliano da Silva**

Santa Maria, 19 de março de 2008.

# RESUMO


Dissertação de Mestrado
Curso de Mestrado em Engenharia de Produção
Universidade Federal de Santa Maria

**UM SISTEMA DISTRIBUÍDO DE CRIPTOANÁLISE COMPUTACIONAL PARA USO FORENSE**

Autor: Antonio Marcos de Oliveira Candia
Orientador: Prof. Dr. Benhur de Oliveira Stein
Local e data da defesa: Santa Maria, 19 de março de 2008.

A criptoanálise apresenta um grande desafio quando algoritmos criptográficos computacionais são enfrentados. Nestes casos é comum a utilização de algoritmos especialmente projetados para inviabilizá-la. A criação de sistemas para a criptoanálise computacional esbarra em problemas como a necessidade de uso de caros sistemas dedicados de alto desempenho. Quando esta se dá em ambiente forense computacional outros problemas surgem, como a necessidade de se utilizar métodos que sejam consenso na comunidade científica e restrições quanto ao tempo disponível para a tarefa.

Este trabalho mostra o projeto e implementação de um sistema distribuído para criptoanálise computacional visando a sua aplicação em ambiente forense. Para poder ser facilmente adaptado a novos algoritmos criptográficos e ser utilizado no maior número possível de processadores, o sistema foi implementado em linguagem Java. Para um maior controle sobre a qualidade e para simplificar a implementação e manutenção do sistema, o mesmo foi dividido em duas grandes partes: um arcabouço de criptoanálise computacional, denominado Quebra-pedra, e uma aplicação distribuída e paralela utilizando metacomputação, ou computação em grade. O uso de metacomputação mostrou-se uma forma relativamente barata de obter alto desempenho na execução da tarefa de criptoanálise. Para a implementação deste paralelismo de execução foi utilizado o *middleware* ProActive, parte do projeto ObjectWeb, que oferece as características necessárias a esta aplicação e é desenvolvido no modelo de *software* livre.

Testes realizados tanto com o arcabouço quanto com o sistema de criptoanálise distribuída demonstraram que os mesmos alcançaram bons níveis de flexibilidade, eficiência e escalabilidade. Além disso, devido às escolhas de projeto, o sistema implementado possui um baixo custo, pois visa a utilização de um ambiente formado por redes de computadores pré-existentes e ferramentas de *software* livre.



**Palavras-chave:** Criptoanálise; computação forense; paralelismo.

# ABSTRACT

Dissertação de Mestrado
Production Engineering Post-graduation Program
Universidade Federal de Santa Maria

**A DISTRIBUTED CRYPTANALYTIC SYSTEM WITH FORENSIC APPLICATIONS**

Author: Antonio Marcos de Oliveira Candia
Advisor: Prof. Dr. Benhur de Oliveira Stein

The computational cryptographic algorithms pose a challenge to cryptanalysis. The use of algorithms designed to make cryptanalysis very hard are commonplace nowadays. The development of applications for cryptanalysis has to deal with problems such as the need for expensive high-performance computing systems. When applied to forensic computing, other problems arise, such as the need to use well-known non-refutable methods and some restrictions on the timeframe available for the task.

This work shows the design and implementation of a distributed cryptanalytic system to be used in a forensic environment. It has been implemented in Java language for easier adaptation to any cryptographic algorithm and portability reasons. For better quality control and design/maintenance simplification, it has been split in two main parts: a cryptanalytic framework, called Quebra-pedra, and a parallel and distributed application using metacomputing, or grid computing. The use of metacomputing proved to be a relatively inexpensive way to obtain high-performance. The metacomputing environment chosen is the ProActive middleware, part of the ObjectWeb project. It has all the needed characteristics and is actively developed using a free software model.

Tests done with the framework and the distributed cryptanalytic system demonstrated that adequate levels of flexibility, efficiency and scalability were achieved. Beyond that, the system has a low cost of deployment achieved by the use of free software tools in its development and the harvesting of idle pre-existing computing power via metacomputing.

**Keywords:** cryptanalysis, computer forensics, parallel computing.

# LISTA DE FIGURAS

# SUMÁRIO

# 1 INTRODUÇÃO

A criação de sistemas computacionais para criptoanálise tem se mostrado, ao longo da história da criptologia, uma tarefa complexa, insatisfatória e, por vezes, frustrante. Desde a implementação dos primeiros algoritmos de criptografia computacional sabe-se que existe um método infalível de quebrá-los: a busca exaustiva. A questão sempre girou em torno dos enormes recursos computacionais em geral necessários para realizar esta tarefa. Uma vantagem dos criptoanalistas é que também, historicamente, alguns dos algoritmos criptográficos mais utilizados na prática normalmente sofrem restrições, sejam de velocidade ou de poder computacional, o que faz com que acabem sendo mais fracos do que o projetado.

Do ponto de vista da criptoanálise para computação forense, porém, o analista não tem poder de escolha. Se encontrar informações criptografadas relevantes a um caso, é sua função obter o texto original, independentemente do algoritmo utilizado, ou provar por quê isto não é possível. Por isso, ele deve ser capaz de atacar o maior número possível de algoritmos. Questões como a quantidade de algoritmos existentes e a falta de documentação sobre muitos deles tornam-se relevantes. Além disso, o tempo disponível para a realização de tal tarefa geralmente é exígüo em relação à sua magnitude, deixando o criptoanalista com pouca margem para experimentações ou explorações mais extensivas. Estas exigências fazem com que o criptoanalista precise utilizar sistemas computacionais flexíveis e de alto desempenho.

Em termos de *hardware* isto implica, em geral, no uso de algum tipo de computação paralela. Atualmente, sistemas paralelos especialmente construídos, tais como o BlueGene/L (DAVIS et al., 2004), possuem enorme capacidade de processamento se comparados aos sistemas monoprocessados comumente utilizados. Porém, por serem especialmente construídos, possuem também, em geral, alto custo de implementação

e manutenção. Uma alternativa a estes sistemas é o uso de recursos computacionais ociosos para o processamento paralelo. Com a utilização de uma infra-estrutura de *software* adequada, um conjunto de computadores interligados por uma rede de comunicação pode ser transformado em um sistema de execução distribuída de tarefas. A computação em grade oferece esta infra-estrutura. Utilizando esta técnica em uma rede de computadores pré-existente tem-se uma alternativa de baixo custo aos sistemas especiais citados.

Já em termos de *software* para criptoanálise surgem algumas outras questões. Há poucas opções disponíveis (MUFFET, 2007; SOLAR DESIGNER, 2007; MONTORO, 2007) e estas não atendem de forma satisfatória aos requisitos impostos pela informática forense, principalmente em termos de flexibilidade. Além disso, devido à restrições impostas por governos de vários países, muitas destas aplicações tornam-se ilegais ou passam a ser tratadas como armas estratégicas e, portanto, têm seu desenvolvimento proibido ou tornado secreto.

Este trabalho mostra o projeto e implementação de um sistema para criptoanálise computacional adaptável a qualquer algoritmo criptográfico do qual se tenha uma implementação em linguagem Java/C/C++. Além disso, este sistema utiliza o processamento distribuído baseado em metacomputação para obter redução de custos de implementação e, principalmente, execução. Para simplificar e viabilizar este projeto, a implementação foi dividida em duas grandes partes: um arcabouço para criptoanálise computacional, que recebeu o nome de Quebra-pedra e encapsula a parte de criptoanálise computacional, e um sistema distribuído que utiliza este arcabouço em uma infra-estrutura de metacomputação para a execução dos ataques. Desta forma, pôde-se tratar isoladamente os problemas associados a cada uma destas áreas.

Os capítulos a seguir estão estruturados da seguinte forma. No capítulo 2 analisam-se as características fundamentais das técnicas de criptoanálise e as alternativas em termos de sistemas de alto desempenho para a implementação de um sistema de criptoanálise de baixo custo. No capítulo 3 encontra-se a visão geral e o projeto do sistema baseado nas características buscadas. No capítulo 4 é mostrado o projeto e implementação do arcabouço de criptoanálise computacional Quebra-pedra. No capítulo 5 é mostrado o projeto e implementação de um sistema

para criptoanálise distribuída que utiliza o arcabouço Quebra-pedra. No capítulo 6 encontram-se o detalhamento dos testes realizados com o arcabouço e com o sistema de criptoanálise e uma avaliação de seu desempenho. Finalmente, o capítulo 7 apresenta a conclusão do trabalho.

# 2 CONTEXTO E PROBLEMÁTICA

O desenvolvimento dos computadores eletrônicos durante e após a Segunda Guerra Mundial tornou possível o avanço de várias áreas do conhecimento humano. Uma das áreas que obtiveram contribuições imensas foi a criptologia. Podendo ser livremente traduzida como "o estudo dos segredos", a criptologia divide-se em duas grandes áreas: a criptografia, que se refere ao uso de técnicas que permitem a troca segura de mensagens mesmo quando terceiros podem interceptá-las, e a criptoanálise, que se refere ao uso de técnicas que permitam a terceiros obter acesso ao conteúdo de uma mensagem criptografada (SCHNEIER, 1995).

Dentre as várias técnicas de criptoanálise, conhecidas como ataques, algumas são específicas enquanto outras possuem cunho mais generalista, podendo ser empregadas em um maior número de situações. Os ataques considerados mais genéricos são aqueles possíveis de serem usados quando somente se tem acesso ao texto cifrado e que se baseiam no método de tentativa-e-erro. Nesta categoria temos os ataques por varredura exaustiva, também conhecido como ataque de força-bruta, e os ataques de dicionário, dentre outros (LENSTRA; VERHEUL, 2001).

O ataque por varredura exaustiva tem como característica principal a garantia de encontrar a chave criptográfica procurada, pois baseia-se em uma varredura total do espaço de chaves possíveis. Porém, como na maioria dos casos este espaço de busca possui um tamanho bastante grande, o ataque, na maioria das vezes, torna-se impraticável em um sistema convencional.

Outro problema é que, mesmo com o uso de computadores, os algoritmos utilizados para a encriptação dos dados, em geral, são propositadamente lentos justamente para inviabilizar estes ataques (SCHNEIER, 1995). Portanto, a implementação e execução dos ataques requer um sistema computacional de alto desempenho ou o

resultado das buscas, mesmo as mais simples, não poderá ser conhecido em um intervalo de tempo compatível com o tempo disponível para isto. Este problema, portanto, torna-se um candidato natural à implementação utilizando processamento paralelo. O uso de mecanismos de processamento paralelo tem se mostrado continuamente, ao longo da história da computação, como uma alternativa viável sempre que é necessário um poder computacional maior do que o obtido em uma máquina monoprocessada.

Ao abordar o problema de desenvolvimento de um sistema de criptoanálise para ambiente forense, além da questão do desempenho, também devem ser levadas em consideração outras características desta área. A informática forense pode ser definida como a inspeção sistemática dos componentes de um computador e de seu conteúdo buscando evidências de atos, possivelmente criminosos, praticados com o auxílio do mesmo. Na maior parte dos casos, os peritos em informática forense investigam dados registrados em dispositivos de armazenamento.

A busca por evidências de atos criminosos na informática forense freqüentemente envolve a manipulação de informações que foram criptografadas. Quando não há cooperação no sentido de fornecimento da chave utilizada para cifragem ou a mesma foi perdida de algum modo, faz-se necessário o uso de técnicas de criptoanálise para a extração das informações originais a partir de sua versão cifrada. O problema é que, atualmente, existe uma enorme gama de algoritmos de criptografia, alguns de conhecimento público e amplamente documentados, outros de caráter proprietário e protegidos por patentes ou licenças restritivas de uso. Devido a esta variedade, a maioria dos sistemas de criptoanálise existentes engloba apenas uma parte destes algoritmos. Uma solução mais ampla é necessária, pois a aplicação de métodos como a engenharia reversa ou a quebra de patentes pode ser inviável devido ao espaço de tempo necessário para isto. Este trabalho mostra o desenvolvimento de um sistema que busca solucionar estes problemas pelo uso de técnicas de criptoanálise computacional, detalhadas na seção 2.1, em conjunto com um ambiente de metacomputação para o processamento paralelo das tarefas, detalhado na seção 2.2.

## 2.1 Criptoanálise Computacional

A criptoanálise é o estudo de métodos para obtenção de acesso à informação criptografada sem o conhecimento do segredo normalmente requerido para tanto — a chave. Tipicamente, isto envolve encontrar a chave secreta utilizada no processo de cifragem, o que, em linguagem não-técnica, recebe comumente o nome de quebra de código ou quebra de senha. A criptoanálise evolui conjuntamente com a criptografia, pois, a cada avanço em uma, a outra necessariamente precisa evoluir ou adaptar-se para tentar contorná-lo.

A partir do surgimento dos computadores, a criptografia experimentou uma grande evolução, pois estes tornaram rapidamente obsoletos os métodos vigentes de cifragem. Analisaremos, agora, o impacto desta evolução sobre a criptoanálise.

Historicamente, podemos dizer que uma das primeiras aplicações computacionais intensivas foi a criptoanálise, pois um dos primeiros computadores eletrônicos — o Colossus — foi construído com o fim, dentre outros, de ajudar os ingleses a desvendar os códigos alemães durante a segunda guerra mundial (COPELAND, 2004). Desde então, a criptoanálise computacional tem sido uma área estratégica tanto nas esferas militar quanto civil. Na informática forense, a criptoanálise é uma ferramenta freqüentemente necessária para possibilitar a busca de evidências de crimes por computador.

O projeto de um sistema criptoanalista recai, primeiramente, sobre como realizar esta tarefa. Em criptoanálise, a busca pelo texto original (*plaintext*) a partir de sua versão cifrada (*ciphertext*) é denominada ataque. Há vários tipos de ataques, cada qual com características que determinam quando o mesmo pode ser utilizado (SCHNEIER, 1995). Os tipos de ataques efetivos em um determinado caso dependem de quanta informação pode ser obtida previamente sobre o sistema a atacar. As restrições impostas pelas informações disponíveis permitem uma categorização geral dos tipos de ataques de criptoanálise existentes atualmente (KNUDSEN, 1999):

**Somente texto cifrado** (*ciphertext-only*) - neste caso, o criptoanalista somente tem acesso a uma coleção de textos cifrados com o mesmo algoritmo criptográfico. Ou seja, deve-se tentar determinar a chave a partir dos textos cifrados sem haver nenhuma informação a mais conhecida com certeza. Em vários casos é possível inferir probabilisticamente o método de criptografia utilizado, o

idioma da mensagem, o assunto ou algumas palavras, o que simplifica muito os ataques, porém nem sempre isto é possível.

**Mensagem conhecida** (*known-plaintext*) - aqui o criptoanalista possui um conjunto de mensagens cifradas e as correspondentes mensagens decifradas. Nestes casos, procura-se por padrões entre pares de cifras que podem levar a informações importantes sobre o algoritmo utilizado ou até mesmo à chave.

**Mensagem escolhida** (*chosen-plaintext*) - quando o criptoanalista pode obter os textos cifrados correspondentes a um conjunto arbitrário de textos não-cifrados a sua escolha. Podendo realizar esta escolha, mensagens especialmente escolhidas podem revelar o funcionamento do algoritmo criptográfico.

**Mensagem escolhida adaptativa** (*adaptive chosen-plaintext*) - similares aos ataques de mensagem escolhida, porém o criptoanalista pode escolher as mensagens subseqüentes baseado nas informações aprendidas nas cifragens anteriores.

**Chaves relacionadas** (*related-key*) - similar à mensagem escolhida, mas o criptoanalista pode obter textos cifrados a partir de uma mesma mensagem e de chaves de criptografia diferentes. Estas chaves, embora desconhecidas ao atacante, possuem alguma relação conhecida como, por exemplo, diferem em apenas um bit.

A partir desta categorização, podemos avaliar melhor os tipos de ataques criptoanalíticos existentes. Os métodos considerados clássicos e que remontam aos primórdios da criptoanálise, em geral, são variações da técnica de análise de freqüência (GANESAN; SHERMAN, 1993). Nesta técnica buscam-se padrões comuns à linguagem original de um texto cifrado através de análise estatística. Esta informação é, então, utilizada para inferirem-se novas informações do texto cifrado. Obviamente, tal técnica somente funciona a contento com os mecanismos clássicos de criptografia para os quais foi inventada originalmente, como, por exemplo, as substituições monoalfabéticas. A utilização de computadores fez com que a pesquisa em criptografia avançasse muito, gerando algoritmos adaptados à velocidade desta ferramenta. Tais algoritmos, em grande parte dos casos, foram projetados para impedir os ataques

por análise de freqüência. Novos métodos criptoanalíticos tiveram, então, que ser desenvolvidos.

Dentre os métodos criptoanalíticos modernos podemos citar como exemplos a criptoanálise linear e a criptoanálise diferencial. Em linhas gerais, a criptoanálise linear envolve encontrar um conjunto de operações cujo resultado se aproxime ao do algoritmo de cifragem sendo atacado e, a partir daí, inferir a chave utilizada em um determinado caso (MATSUI, 1994). Já a criptoanálise diferencial é do tipo mensagem escolhida (BIHAM; SHAMIR, 1994). Neste método, o atacante escolhe textos com uma diferença conhecida - a operação ou-exclusivo é uma escolha comum para computar esta diferença. Tais textos são enviados para criptografia e é realizada uma análise da diferença nos textos cifrados. O método busca descobrir a chave utilizada na cifragem baseado nestas análises.

De acordo com WIENER (1996), a técnica de criptoanálise linear pode encontrar uma chave do algoritmo de criptografia `DES` com o uso de $2^{47}$ textos quaisquer ou $2^{43}$ textos conhecidos. Já a técnica de criptoanálise diferencial pode ser usada para encontrar uma chave `DES` de 56 bits dados $2^{47}$ textos escolhidos. Além disso, estes ataques são originalmente dirigidos às cifras de bloco como o DES, FEAL, Skipjack e similares. Ou seja, a menos que estes tipos de ataques sejam drasticamente melhorados para diminuir o número e tipo de textos necessários e aumentar a sua abrangência em termos de algoritmos passíveis de ataque, eles possuem somente interesse acadêmico.

Vejamos, então, alguns dos métodos considerados de cunho mais genérico: os ataques que dependem somente de um texto cifrado (*ciphertext-only*). A varredura exaustiva (ou força-bruta) e o ataque dirigido por dicionário fazem parte desta categoria.

O ataque por força-bruta ou busca exaustiva tem como característica principal a garantia de encontrar a chave criptográfica procurada, pois baseia-se em uma varredura total do espaço de chaves possíveis - a única exceção, atualmente, é a criptografia por chave única (*one-time pad*) (SCHNEIER, 1995). Neste ataque, cada possível combinação de caracteres é testada como chave de decriptação de um texto cifrado. Porém, como na maioria dos casos este espaço de busca possui um tamanho bastante grande, este ataque torna-se impraticável em um sistema convencional. Por

exemplo, consideremos que o espaço de busca seja composto apenas pelo conjunto de caracteres formado pelos dez algarismos e pelas vinte e seis letras do alfabeto latino. Neste caso, se fôssemos testar todas as possíveis combinações de um a oito caracteres, chegaríamos ao número total de mais de 2,9 trilhões de possibilidades. Assumindo-se que um sistema hipotético gaste um milésimo de segundo para testar cada possibilidade, teríamos um total de aproximadamente 92 anos para cobrir o espaço de buscas total. Note-se que não foram incluídos neste cálculo os caracteres de pontuação, que corriqueiramente são utilizados para formação de chaves e que elevariam exponencialmente o número de possibilidades a serem testadas. Como o ataque por busca exaustiva normalmente necessita de quantias extremamente grandes de recursos computacionais, na forma de memória ou tempo, o mesmo costuma ser impraticável ou mesmo impossível de ser utilizado em sistemas computacionais convencionais.

O ataque dirigido por dicionário é uma variação do ataque por força-bruta que busca diminuir os recursos computacionais necessários. Nesta modalidade de ataque o criptoanalista utiliza uma lista de palavras a serem testadas como chaves. Desta maneira, o sucesso deste método depende exclusivamente da abrangência desta lista. Há duas maneiras de aumentar a eficiência deste ataque. A primeira consiste em aumentar o tamanho do dicionário (dicionários de várias línguas e dicionários técnicos são boas escolhas neste aspecto, além de informações particulares à cada caso). A segunda forma é a utilização de um conjunto de regras de modificação que atuam sobre o dicionário básico, estendendo-o para englobar uma porção maior do espaço de busca total. As modificações podem incluir, por exemplo, a inversão de palavras, adição de numerais, trocas simples de caracteres, entre outras. Este ataque baseia-se na premissa de que uma grande porcentagem dos usuários escolhe suas chaves criptográficas dentre palavras presentes em seu vocabulário, ou modificações simples destas. Como um dicionário é na verdade um subconjunto do espaço de buscas total, este direcionamento resulta em um número reduzido de possibilidades quando comparado à busca por varredura completa, o que se traduz em uma diminuição proporcional nos recursos necessários ao ataque. Por exemplo, o dicionário da língua inglesa atualmente fornecido com as distribuições do sistema operacional Linux conta com aproximadamente 235.000 palavras. Utilizando os mesmos parâ-

metros do exemplo anterior, chegamos à conclusão que são necessários 4 minutos para testar todo o dicionário. O teste de cada modificação imaginada para este dicionário irá tomar os mesmos 4 minutos. Cabe ao analista, de acordo com cada caso, escolher um dicionário e um conjunto de regras de modificação que melhorem suas possibilidades de sucesso. Mesmo com estas características, o ataque por dicionário também pode tornar-se computacionalmente dispendioso ao se utilizar dicionários e/ou conjuntos de regras muito grandes ou ainda algoritmos de criptografia mais lentos do que o utilizado neste exemplo.

Há várias abordagens para a implementação computacional destes ataques. Vejamos as características, vantagens e desvantagens de algumas dessas abordagens.

### 2.1.1 Criptoanálise através de *hardware* dedicado

É possível projetar e construir circuitos dedicados à tarefa de atacar um determinado algoritmo criptográfico. Vários destes sistemas já foram construídos ou teorizados (LENSTRA et al., 2002; SHAMIR, 2000; SHAMIR; TROMER, 2003; WIENER, 1996). Sua maior vantagem é que, como são implementações em *hardware*, tendem a ser extremamente velozes.

Em 1991 foi proposta uma variação interessante para a criptoanálise de força-bruta do DES ou outros algoritmos similares. Este ataque foi denominado de Loteria Chinesa e é, simplificadamente, o uso de um sistema massivamente paralelo baseado em uso de hardware embutido em produtos eletrônicos comuns (televisores, por exemplo) para a busca de chaves (QUISQUATER; DESMEDT, 1991).

Desde então, vários avanços na computação fazem com que vejamos a idéia da Loteria Chinesa com outros olhos. Por exemplo, hoje o número de computadores ligados à Internet é várias ordens de magnitude maior, os processadores estão mais rápidos, e o uso de *intranets* tornou-se comum. Isto criou um potencial enorme para a realização de tarefas com grandes custos computacionais utilizando esta infraestrutura.

Outro exemplo que comprova a possibilidade de utilização deste tipo de abordagem é o projeto Copacobana (KUMAR et al., 2006), que utiliza-se de computação reconfigurável para a construção de uma máquina paralela de baixo custo capaz de atacar algoritmos simétricos com chaves de até 64 bits.

A utilização de circuitos dedicados para uso prático, porém, esbarra na necessidade de haver uma implementação física de cada algoritmo a ser atacado. Esta implementação requer, no mínimo, a existência de um laboratório ou planta industrial capaz de executar o projeto dos circuitos requeridos. Este é um processo lento e oneroso e, por isso, raramente utilizado. A popularização do uso de `VHDL` e `FPGA`'s pode fazer com que este método torne-se mais freqüente em ambientes de computação forense, mas ainda assim, seu custo torna-o proibitivo para uso em larga escala ou quando a chave criptográfica excede os 64 bits.

### 2.1.2 Método de computação dinâmica

Vários programas estão disponíveis hoje em dia para a realização de criptoanálise computacional através de computação dinâmica, tornando este o método mais comum atualmente. Este método é praticamente idêntico ao uso de circuitos dedicados, exceto pela utilização de programas de computador ao invés desses circuitos. Freqüentemente tais programas são otimizados à exaustão, utilizando-se inclusive de código em linguagem de máquina na porção considerada crítica dos mesmos. Também freqüentemente estes sistemas encontram-se associados à indivíduos ou grupos com propósitos excusos - os famosos *hackers* e/ou *crackers* - o que faz com que os mesmos não sejam vistos com bons olhos pela comunidade em geral.

Controvérsias à parte, dentre os programas mais utilizados atualmente podemos citar: Crack (MUFFET, 2007), John The Ripper (SOLAR DESIGNER, 2007), Cain & Abel (MONTORO, 2007). Em geral estas ferramentas são capazes de realizar buscas por dicionário e/ou força-bruta, normalmente sobre alguns tipos específicos de algoritmos criptográficos e, em sua maioria, são voltadas para a busca por senhas de acesso a sistemas. É possível extender alguns destes programas para qualquer algoritmo de criptografia, isto, entretanto, exige a sua modificação implicando em que haja conhecimento de sua arquitetura interna. Isto torna-os inadequados ao uso imediato com algoritmos criptográficos desconhecidos. Alguns deles podem ser executados de forma distribuída ou possuem versões alternativas com esta característica, porém, exige-se a configuração manual do sistema inteiro nestes casos. Desta forma, surge a necessidade de uma ferramenta mais simples e flexível para o uso em informática forense.

### 2.1.3 Método de computação prévia

Outra abordagem possível é a computação prévia de todas as possíveis chaves e textos cifrados com as mesmas. Neste método a busca torna-se uma consulta ao banco de dados formado por estas informações (BORST; PRENEEL; VANDEWALLE, 1998; OECHSLIN, 2003). Sua principal vantagem é o fato de que o esforço computacional de geração dos textos cifrados é feito uma única vez e seus resultados são reutilizados quando preciso, não sendo necessária a geração dos mesmos dados repetidamente como na computação dinâmica. A existência de mídias de armazenamento secundário de grande capacidade, baixo custo e latência de acesso cada vez mais baixa favorecem esta técnica. Na prática, está se trocando o uso de um sistema limitado por processamento (*CPU-bounded*) por um limitado pelo sistema de entrada/saída (*I/O-bounded*). A desvantagem está na capacidade de armazenamento necessária à este tipo de sistema. Para armazenar somente as 361 trilhões de chaves do exemplo anterior cifradas com o algoritmo `DES` seriam ocupados cerca de 4.7 Petabytes. Considerando que esta é a capacidade de armazenamento utilizada para um único algoritmo e para um número relativamente reduzido de chaves, podemos notar a pouca praticidade deste método para uso geral.

Entretanto, um exemplo de aplicação prática da computação prévia foi implementado e encontra-se em franca utilização: o RainbowCrack (SHUANGLEI, 2007). Utilizando as idéias de Oechslin (OECHSLIN, 2003), este sistema foi otimizado para possibilitar ataques a *hashs*, tais como o `LM` (do sistema MS-Windows), `MD5` e `SHA-1`. Seu uso para outros tipos de algoritmos é possível, porém, esbarramos novamente nas limitações de recursos citadas acima, que tornam seu uso não-prático nestes casos.

## 2.2 Sistemas de alto desempenho e baixo custo

Ao analisarmos a história da computação, notamos que esta mostra uma clara linha evolutiva, sempre baseada na mesma premissa: poder computacional. Ano após ano, cada novo computador lançado reafirma esta tendência. Novas tecnologias de construção são inventadas, materiais são trocados, dispositivos são incorporados, tudo isto com o mesmo objetivo, fazer com que os computadores sejam cada vez mais velozes e tenham maior capacidade de armazenamento e manipulação de dados para

poderem resolver problemas cada vez mais complexos.

Alguns marcos podem ser facilmente encontrados nestes mais de 60 anos de evolução, tanto nas máquinas quanto em seus programas. O surgimento do transistor e dos circuitos integrados trouxe velocidade, confiabilidade e redução no tamanho das máquinas. A introdução das Linguagens de Programação e do conceito de Sistema Operacional trouxe flexibilidade e facilidade de uso, promovendo a popularização no uso de computadores. Estes foram, então, retirados de centros de pesquisa e aplicados às mais diversas tarefas como comércio, educação e entretenimento, somente para citar alguns exemplos. Hoje em dia, virtualmente toda tarefa que necessite de manipulação de qualquer tipo de dados pode se utilizar de computadores. Porém o mesmo objetivo ainda norteia o seu desenvolvimento: maior poder computacional.

Nos anos 60 o mundo da informática foi revolucionado com o aparecimento de computadores que, comparados aos seus concorrentes contemporâneos, eram muito mais poderosos. O segredo era uma nova técnica, descrita como sendo o uso de várias unidades internas replicadas, atuando conjuntamente para resolver um mesmo problema. Hoje em dia conhecemos esta técnica pela denominação de processamento paralelo. Desde aquela época, o processamento paralelo vem sendo utilizado regularmente para se conseguir um aumento maciço de poder computacional, porém, devido aos problemas encontrados (complexidade de criação, manutenção, uso e desenvolvimento de aplicativos), normalmente ele era relegado a permanecer em uso somente em centros de pesquisa, na forma de protótipos ou modelos para pesquisa da própria tecnologia de sua construção e uso.

Mas, surge uma questão: por quê preocupar-se desenvolvendo processamento paralelo se a cada nova geração os processadores seqüenciais mostram-se mais poderosos? A resposta é simples: por melhores que sejam os processadores, sempre há problemas que eles ainda não conseguem resolver em um tempo razoável (STERLING et al., 1995). Um exemplo deste tipo de problema, como visto na seção anterior, é a criptoanálise computacional.

Atualmente, para utilizar processamento paralelo não é necessário pesquisar/desenvolver novas tecnologias de construção, apenas ajustar as existentes, e fazer com que o seu uso se torne simplificado, permitindo assim acesso ao grande público interessado em maior poder computacional. Com a explosão no crescimento do

mercado de redes locais, praticamente qualquer grupo/instituição com sério interesse em realizar criptoanálise computacional tem muitos processadores a sua disposição. Porém, vários destes grupos - principalmente em nosso país - não possuem os recursos necessários para ter um sistema de alto desempenho dedicado a esta tarefa.

O apêndice A mostra as alternativas de processamento paralelo com baixo custo pesquisadas e a seção abaixo detalha aquela escolhida para a implementação realizada: a metacomputação.

### 2.2.1 Metacomputação

Observações simples mostram que um computador pessoal típico em uma organização é sub-utilizado pois fica a maior parte do tempo em estado ocioso. Estes processadores ociosos podem ser utilizados para a execução concorrente de tarefas durante tais períodos. Exemplos de sistemas que utilizam esta abordagem de compartilhamento de ciclos ociosos de UCP são o CADEO (CERA; PASIN, 2004), o SETI@Home (KORPELA et al., 2001) e suas variantes.

O uso de recursos computacionais distribuídos depende da habilidade de gerenciar estes recursos. Porém, a complexidade dos ambientes de rede pode facilmente tornar este gerenciamento impraticável. Por exemplo, como compartilhar recursos e, ao mesmo tempo, manter a autonomia de múltiplos domínios administrativos, esconder as diferenças entre arquiteturas computacionais incompatíveis, comunicar sistemas consistentemente enquanto conexões são perdidas ou modificadas e respeitando múltiplas políticas de segurança? A solução comumente usada tem sido a criação de sistemas que atacam estes problemas caso-a-caso. Isto acaba por criar soluções que, embora resolvam o problema, tornam-se intrincadas colchas de retalhos. Se tudo correr bem, hábeis programadores podem criar e manter um sistema nestes moldes, mas estas implementações serão, via de regra, limitadas, custosas e de difícil operação e expansão. Escalabilidade, segurança e tolerância a falhas são algumas características importantes em qualquer solução para este problema.

Uma melhor solução envolve a utilização de uma infra-estrutura que possa criar uma abstração sobre um conjunto complexo de recursos distribuídos e disponibilize uma forma, preferencialmente de alto-nível, de compartilhamento e gerenciamento destes recursos. Soluções têm sido propostas nestes moldes, como, por exemplo, o

Legion (GRIMSHAW; WULF, 1997) que utiliza a abordagem de sistema operacional distribuído em grande escala. Outra possibilidade é a utilização de mecanismos de metacomputação ou computação em grade. Assim como os sistemas distribuídos em larga escala, a grade computacional também possibilita a utilização de recursos computacionais distribuídos de forma simplificada. Porém, usualmente o foco deste tipo de infraestrutura é a utilização de recursos ociosos como, por exemplo, computadores pessoais, durante os intervalos em que estes estão ligados mas não estão sendo utilizados.

Destes, um exemplo é o BOINC (*Berkeley Open Infrastructure for Network Computing*) (ANDERSON, 2004), que é uma estrutura que busca simplificar o trabalho de criar e operar projetos de computação maciça com recursos públicos. Proprietários de computadores ligados a mesma rede de um servidor BOINC podem participar em múltiplos projetos, voluntariamente, e podem especificar como seus recursos serão alocados entre estes projetos. Outra opção de estrutura de metacomputação é o *middleware* ProActive (BADUEL et al., 2006), que, diferentemente do sistema BOINC que utiliza internamente a linguagem C++, é feito completamente em linguagem Java.

Sistemas deste tipo têm sido usados para problemas complexos como análise de modelos climáticos, renderização gráfica de cenas complexas, dentre outros. Esta arquitetura permite que muitos processadores, possivelmente heterogêneos e geograficamente dispersos, trabalhem juntos para alcançar a solução de um problema. Entretanto, características como a heterogeneidade de arquiteturas e sistemas operacionais, variação de carga nas máquinas, variação na disponibilidade das máquinas e a suscetibilidade à falhas nas comunicações e nos processadores complicam a situação para o programador. Nesse contexto, novos paradigmas de programação que reduzem a complexidade tradicionalmente envolvida na programação distribuída, que envolve tarefas como o balanceamento de carga, heterogeneidade e tolerância a falhas, ganham importância (JOSHI; RAM, 1999).

A grade computacional ou metacomputação possui como objetivo de projeto simplificar o processamento de problemas muito grandes para serem resolvidos por um único sistema de computação de alto-desempenho (FOSTER; KESSELMAN, 1998). Isto é possível pelo uso de recursos ociosos (ciclos de UCP e/ou espaço de

armazenamento) de grandes números de computadores dispersos que são tratados como um aglomerado virtual com uma infraestrutura de comunicação distribuída. O foco da metacomputação na habilidade de suportar computação transparentemente através de múltiplos domínios administrativos é o que a faz diferente dos aglomerados ou da computação distribuída tradicional.

Devido a algumas destas características, atualmente esta tecnologia é ideal para computações do tipo trivialmente paralelizável (*embarrassingly parallel*) (FOX; WILLIAMS; MESSINA, 1994). Porém, como não se tem controle sobre os seus elementos, vários cuidados devem ser tomados. Por exemplo, atualmente, não há como especificar recursos de rede, portanto não se deve utilizar este modelo para problemas que necessitem comunicar grandes volumes de dados. Similarmente, como não se sabe de antemão a capacidade de cada nó, a mesma deve ser medida dinamicamente para se poder realizar balanceamento de carga entre os nós. Em alguns casos podem haver, inclusive, diferenças tão grandes quanto à capacidade de processamento que nós simplesmente devem ser descartados por serem muito lentos em relação aos outros. Outro problema, talvez mais relevante, é o fato de que os nós não são dedicados ao processamento paralelo, mas sim apenas um empréstimo temporário. A qualquer momento um destes nós pode, portanto, simplesmente ser retirado do conjunto de nós sem aviso prévio. Embora esta situação também possa acontecer em sistemas concorrentes tradicionais, ela se torna muito mais comum neste tipo de arquitetura.

A computação em grade é freqüentemente confundida com a computação em aglomerados. A principal diferença entre estas tecnologias é que um aglomerado é formado por um conjunto de nodos (processadores) em um local — e normalmente em uma rede local — enquanto a grade é composta por qualquer tipo de recurso computacional ligado à mesma.

O termo computação em grade (*grid computing*) originou-se nos anos 1990 como uma metáfora do acesso e uso de recursos computacionais tornar-se tão simples quanto o acesso e uso do sistema de distribuição de energia elétrica (*power grid*) (FOSTER; KESSELMAN, 1998).

A definição de grade foi feita por Ian Foster (FOSTER, 2002) e engloba três pontos fundamentais:

- recursos computacionais não são administrados de forma centralizada;

- padrões abertos são utilizados; e
- qualidade de serviço não-trivial é alcançada.

Uma característica que distingue a computação em grade da computação distribuída tradicional é a abstração de um recurso distribuído em um recurso da grade. O resultado desta abstração e que ela simplifica o processo de substituição de recursos. Como efeito colateral, a sobrecarga associada a esta flexibilidade faz parte da camada de infra-estrutura e aumenta a latência associada ao acesso de um recurso da grade. Esta sobrecarga, portanto, deve ser levada sempre em consideração em termos do impacto causado pela utilização de cada recurso da grade por uma aplicação.

Este trabalho descreve algumas das peculiaridades envolvidas no desenvolvimento de uma aplicação distribuída de alto desempenho que é executada em um ambiente de metacomputação. Analisaremos a seguir o ProActive, a alternativa de infra-estrutura de processamento em grade escolhida devido à utilização da linguagem Java para a implementação do arcabouço Quebra-pedra.

### 2.2.2 Metacomputação com ProActive

Atualmente, na busca por alternativas de processamento paralelo, visando principalmente baixo custo, a metacomputação mostra-se uma das melhores. Sua capacidade de utilização de sistemas pré-existentes, desde que interligados, com a simples instalação de um *middleware* é a principal responsável por esta escolha.

Esta escolha também é reforçada pela existência de alternativas de *middleware* no modelo de *software* livre, tais como o ProActive. O ProActive é um *middleware* Java de código aberto (licença LGPL) para programação paralela, distribuída e multi-*threaded* em ambiente de grade computacional (BADUEL et al., 2006).

Como o ProActive é construído utilizando as APIs Java padrão, ele não requer qualquer modificação no ambiente de execução padrão da linguagem, nem utiliza compiladores, pré-processadores ou máquinas virtuais especiais. O modelo de distribuição e atividade do ProActive é parte de um esforço para melhorar a simplicidade e reuso na programação de sistemas orientados a objetos distribuídos e concorrentes (CAROMEL; KLAUSER; VAYSSIÈRE, 1998; CAROMEL; BELLONCLE; ROUDIER, 1996), incluindo uma semântica precisa (ATTALI; CAROMEL; GUIDER,

2000).

Uma aplicação distribuída que utilize o ProActive é composta de entidades chamadas objetos ativos (*active objects*). Cada objeto ativo possui um elemento distinto, a raiz, que é o único ponto de entrada para o objeto ativo. Cada objeto ativo tem seu próprio fluxo de controle e a habilidade de decidir em que ordem irá servir os pedidos de chamada de método que são automaticamente colocados em uma fila de requisições não-atendidas. Chamadas a métodos de objetos ativos sempre são de natureza assíncrona e a sincronização é gerenciada por um mecanismo conhecido como espera-por-necessidade (*wait-by-necessity*) (CAROMEL; KLAUSER; VAYSSIÈRE, 1998). Há um curto *rendez-vous* no início de cada chamada remota, que bloqueia o chamador até que a chamada tenha alcançado o contexto do chamado.

A biblioteca ProActive possui mecanismos para migrar qualquer objeto ativo de uma máquina virtual Java (JVM) para qualquer outra. Outro serviço extra disponibilizado pelo ProActive é a capacidade de criar remotamente objetos que podem ser acessados remotamente. Por esta razão, é necessário identificar JVMs e adicionar alguns serviços. Estas capacidades extras são dadas pelos nodos. Um nodo é um objeto definido pelo ProActive cujo objetivo é juntar um ou mais objetos ativos em uma entidade lógica. Desse modo, um nodo é uma abstração para a localização física de um conjunto de objetos ativos. Uma JVM pode conter um ou mais nodos. Para simplificação, os nodos são nomeados e manipulados através de nomes simbólicos, tipicamente uma URL que identifica sua localização, como, por exemplo, `rmi://paple.inf.ufsm.br/Node1`.

Um objeto ativo pode ser criado diretamente em um nodo ou ligado dinamicamente a um como resultado de uma migração. Para auxiliar na fase de utilização de componentes do ProActive, foi introduzido o conceito de nodos virtuais para mapeamento de objetos ativos (BAUDE et al., 2002). Estes nodos virtuais são descritos externamente através de descritores baseados em XML que são lidos em tempo de execução pelo ProActive quando necessário.

Por ser uma característica importante em computação em grade, a biblioteca ProActive oferece mecanismos de comunicação em grupo, similar àquela apresentada pelo modelo MPI. Estão disponíveis no ProActive operações de grupo tais como: *broadcast, scatter, gather*. Operações como *reduce* podem ser feitas pelo cha-

mador de um método *gather* assim que todos os resultados chegarem ao mesmo, portanto, não é explorada pelo ProActive a oportunidade de redução paralela. As operações mais sofisticadas do MPI - e mais raramente usadas - como *all-gather, all-to-all, all-reduce*, etc, não são disponibilizadas pelo ProActive. Todas estas operações podem, eventualmente, ser implementadas a partir das operações suportadas citadas acima. O motivo de a biblioteca não suportar diretamente estas operações recai sobre o tipo de paralelismo do modelo ProActive, que utiliza grãos médios ou grossos. As operações como redução fazem sentido, do ponto de vista de desempenho, em situações de grão fino.

Para abstrair do código fonte qualquer referência à configurações de *software* ou *hardware*, o ProActive utiliza os descritores de distribuição (*deployment descriptors*). Eles também disponibilizam um mecanismo integrado de especificação de processos externos que devem ser lançados e a forma de fazê-lo (HUET; CAROMEL; BAL, 2004). Seu objetivo é habilitar o lançamento de uma aplicação em qualquer conjunto de sistemas sem ter que modificar o código fonte. Para isto, toda a informação necessária encontra-se armazenada em um arquivo de descrição, em formato XML. Um arquivo de distribuição é composto de três partes. A primeira parte é usada para declarar os nomes de nodos que serão utilizados no código fonte da aplicação — os nodos virtuais. A segunda parte, mapeamento, descreve como os nodos virtuais serão mapeados às máquinas virtuais. Finalmente, na seção de infraestrutura, o arquivo descreve como as máquinas virtuais devem ser criadas. Nesta parte é possível especificar vários detalhes a serem usados na criação das máquinas virtuais, tais como variáveis de ambiente por exemplo. Essencialmente este arquivo descreve, portanto, como será o ambiente de execução distribuído a ser utilizado na execução de uma aplicação.

Outras partes importantes do ProActive, principalmente para a implementação em questão, são os seus arcabouços de segurança e de tolerância a falhas. O ProActive disponibiliza um grande conjunto de recursos de segurança. Ele oferece desde autenticação, integridade e confidencialidade de comunicações até recursos de alto nível, tais como a migração segura de objetos, políticas de segurança dinamicamente negociadas e hierárquicas. Todos estes recursos podem ser utilizados de forma transparente pelas aplicações. A segurança é baseada em infra-estrutura de chave pública.

Cada entidade possui um certificado, sendo estes gerados automaticamente pelo mecanismo de segurança, o qual também oferece meios de verificação de sua validade (BADUEL et al., 2006). A segurança pode ser expressa em diferentes níveis para as aplicações que utilizam o ProActive, variando desde o nível de domínio até o nível de objeto, permitindo grande flexibilidade em sua utilização. No nível de aplicação, por exemplo, basta incluir algumas configurações simples no arquivo descritor de distribuição XML. Ressalta-se que este arcabouço de segurança utiliza a Extensão Criptográfica nativa da linguagem Java (JCE REFERENCE GUIDE, 2007) para tudo isto.

ProActive também disponibiliza capacidade de tolerância à falhas. Isto pode ser feito através de dois protocolos distintos: um baseado em pontos de verificação induzidos pela comunicação (*Communication-Induced Checkpointing* — CIC) e outro baseado em monitoramento pessimista de mensagens (*Pessimistic Message Logging* — PML). Cada um destes protocolos permite a recuperação de um estado seguro da aplicação, quando necessário. Isto possibilita o relançamento da aplicação a partir deste ponto, eliminando qualquer possível problema causado pela falha em algum nó da infra-estrutura de metacomputação. A utilização de mecanismos de tolerância à falhas em uma aplicação ProActive, a exemplo dos mecanismos de segurança, é completamente transparente (BADUEL et al., 2006).

Resumidamente, pelas características apresentadas, o *middleware* ProActive pode ser considerado eficaz e eficiente para utilização de programação paralela em ambientes de metacomputação de natureza hierárquica, altamente distribuída e heterogênea. E, não menos importante, possui um relativo baixo custo de implantação, visto que pode ser utilizado em uma rede de computadores pré-existente.

# 3 PROJETO DE UMA SOLUÇÃO

*"It is important to realize that any lock can be picked with a big enough hammer."*
(*Sun System & Network Admin manual*)

Tendo como objetivo o desenvolvimento de um sistema criptoanalista para informática forense, podemos definir quais os aspectos principais que o mesmo deve ser capaz de suprir. Para isto, é necessário que se tenha uma idéia mais aprofundada do problema sendo enfrentado.

Durante a busca por evidências de atos criminosos realizados com o auxílio de um computador, um perito em informática forense pode deparar-se com informações criptografadas. É sua função, a partir de então, obter as informações originais para avaliar se contêm alguma daquelas evidências. Como visto no capítulo 2, se não houver outra maneira de se obter as informações originais, o perito deve utilizar ataques criptoanalíticos sobre os dados criptografados. Para realizar um ataque, em geral são necessárias algumas informações. Normalmente uma destas informações é o algoritmo criptográfico utilizado. Como há um grande número de tais algoritmos, um sistema flexível de criptoanálise deve ser capaz de se adaptar a qualquer um deles. Na maioria dos casos um ataque possui um alto custo computacional, sendo comum a medição em anos do tempo necessário para um computador monoprocessado executá-lo. Assim, deve haver uma busca pelos métodos que apresentem uma maior redução no tempo total do ataque. Por ser um sistema voltado à aplicação forense, algumas restrições são impostas ao mesmo. Por exemplo, deve-se ter bastante cuidado quanto à segurança das informações manipuladas pelo mesmo. Além disso, os métodos utilizados pelo sistema devem ser consenso na comunidade científica especializada para evitar a refutação dos resultados obtidos. Também, em geral, há restrições de tempo impostas pelo sistema legal que fazem com que seja necessário

se ter uma previsão do tempo total que um ataque irá dispender.

Este capítulo apresenta os requisitos considerados importantes para a implementação do sistema de criptoanálise proposto e uma descrição da funcionalidade esperada do mesmo.

## 3.1 Avaliação dos requisitos

Em qualquer projeto de aplicativo a tarefa mais importante a ser realizada é a análise dos requisitos do mesmo. Partindo deste princípio, analisaremos os principais requisitos do sistema de criptoanálise para computação forense proposto:

**Capacidade de realização de ataques -** A pesquisa na área de criptoanálise ao longo dos anos tem gerado uma grande variedade de métodos para tal fim, como visto na seção 2.1. Um sistema de criptoanálise deve poder ser extendido para abranger novos tipos de ataques que sejam mais eficientes em uma determinada situação. Além disso, deve ser capaz de atacar, ou ser extendido para atacar, qualquer algoritmo criptográfico.

**Desempenho -** Como visto anteriormente, a criptoanálise tipicamente demanda enorme poder computacional. Na informática forense, rotineiramente, há restrições de tempo impostas pelo decurso das investigações e processos legais. Estas restrições impõe que um sistema de criptoanálise forense computacional utilize métodos que possam ser dimensionados para execução em um dado espaço de tempo.

**Previsibilidade de tempo de execução -** Devido à existência de restrições temporais, deve-se poder estimar de antemão o tempo total de realização de um ataque. Isto serve tanto para verificação de viabilidade de ataques quanto para dimensionamento temporal dos mesmos.

**Segurança da informação -** Por se tratar de área que trata de informações sujeitas a sigilo legal, esta característica deve se extender a todo os sistemas que sejam utilizados. Ou seja, os dados e resultados devem ser mantidos privados e o sistema deve ser imune à ataques por terceiros.

Além disso, em termos de métodos adequados à informática forense algumas considerações extras devem ser feitas.

Um perito forense deve ser capaz de demonstrar que seus métodos investigativos, além de eficazes, são o mais transparentes possíveis. Isto se deve ao fato de que o resultado do trabalho do perito é uma evidência criminal, ou seja, uma prova que se destina a firmar a convicção do juiz sobre a verdade dos fatos alegados pelas partes envolvidas em um processo penal. Devido a isto, o perito deve utilizar-se de métodos que sejam corretos e que esta corretude seja consenso na comunidade científica especializada.

## 3.2 Projeto geral

A premissa básica de um sistema de criptoanálise é simples. O analista possui o texto cifrado e precisa da chave que decifra o mesmo. Como visto na seção 2.1, para se obter esta chave é necessário realizar um ataque sobre o texto cifrado. Portanto, esta deve ser a funcionalidade principal do sistema. O diagrama da figura 3.1 mostra, sob o modelo de *use case* do padrão UML de modelagem de sistemas, esta idéia principal. Nota-se que este diagrama já inclui também a idéia de uma base de dados, no formato de cache, que armazena informação relevante sobre os ataques já realizados.

Um sistema mais robusto possibilita mais de um tipo de ataque, neste caso, o analista escolhe o mais apropriado, passa o texto cifrado como entrada e espera a chave como saída. O diagrama da figura 3.2 mostra estes refinamentos da idéia principal do sistema. Aqui pode-se ver a incorporação de dois conceitos novos ao sistema: ataque e tarefa. Para realizar um ataque o criptoanalista deve definir uma tarefa no sistema. Esta tarefa possui alguns atributos, tais como, por exemplo, o algoritmo a ser utilizado. Depois de definir a tarefa, o analista deve definir o tipo de ataque a ser utilizado. A partir destas informações, o sistema irá executar este ataque.

Ainda buscando robustez, o sistema deve ser capaz de realizar ataques a algoritmos do tipo irreversível (*one-way*), como no caso de senhas de sistemas como o Unix ou Windows, armazenadas neste formato, mas também deve ser capaz de atacar arquivos de texto criptografados. Para ser capaz deste último ataque, o sistema deve contar com algum método de detecção de texto para saber quando obteve sucesso.

Além das características pertinentes à tarefa de criptoanálise é necessário o uso

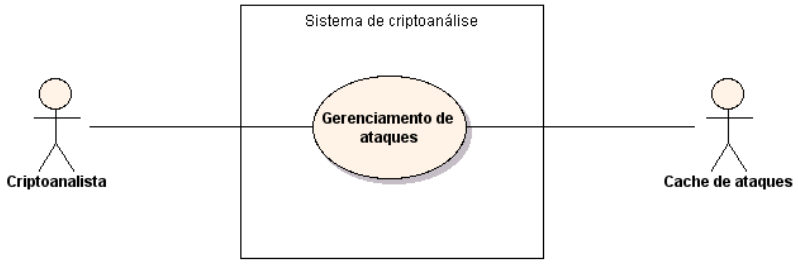



Figura 3.1: Diagrama de caso de uso do sistema de criptoanálise

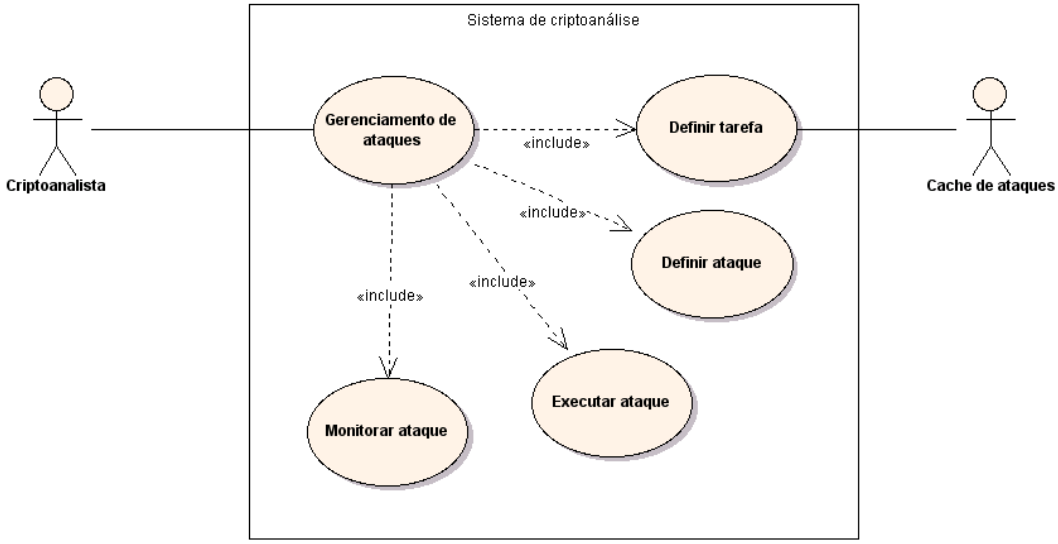



Figura 3.2: Diagrama refinado de caso de uso do sistema de criptoanálise

de processamento paralelo e distribuído, conforme visto na seção 2.2, para obter um melhor desempenho em um problema como a criptoanálise. Vejamos, portanto, algumas características importantes neste tipo de sistema.

Internamente a um sistema distribuído algumas características são esperadas, como a capacidade de utilização de recursos disponíveis e a recuperação de falhas durante a sua execução. Além disso, a facilidade de disparo e término de uma tarefa em um ambiente distribuído também é um fator importante quando se leva em consideração o uso de um grande número de processadores para a sua execução. Por se tratar de uma tarefa que demanda, potencialmente, muito tempo para sua resolução, a possibilidade de reinício a partir de um ponto de verificação (*checkpoint*) torna-se bastante útil.

Para simplificar e melhorar o projeto e implementação do sistema pretendido, optou-se por separar a parte de criptoanálise em um arcabouço (o Quebra-pedra).

Tal separação permite que os métodos criptoanalíticos sejam tratados isoladamente dos mecanismos de execução distribuída de código. Isto permite que novos métodos de ataque sejam implementados sem que seja necessário modificar os métodos de execução, e vice-versa. Também permite que o mecanismo de ataques seja utilizado para a implementação de qualquer sistema voltado à criptoanálise, com ou sem o uso de processamento paralelo para sua execução. O sistema final conta, portanto, com duas partes distintas: o núcleo de criptoanálise e o núcleo de execução distribuída. Os capítulos seguintes tratam, respectivamente, destes núcleos.

# 4 UM ARCABOUÇO PARA CRIPTOANÁLISE

"Se você remover pedra por pedra, até mesmo
uma montanha será demolida."
(Provérbio hindu)

Durante a revisão da literatura pertinente a este trabalho não foi encontrada nenhuma biblioteca ou arcabouço que oferecesse as características desejadas a um sistema criptoanalista de uso geral como o pretendido. Devido a isto, optou-se por projetar e implementar um arcabouço conforme as especificações necessárias. Teve origem, assim, o arcabouço Quebra-pedra. Ele recebeu este nome em referência ao trabalho maciço necessário para se atacar um texto cifrado, comparável ao trabalho historicamente realizado por apenados em regime de trabalhos forçados.

Deve ser ressaltado que este arcabouço, ao menos inicialmente, pretende somente dar suporte ao desenvolvimento de um sistema básico de criptoanálise distribuída. Eventualmente, ficando comprovada a eficácia deste sistema ou havendo necessidade ou demanda, o arcabouço deverá evoluir para englobar técnicas mais sofisticadas de criptoanálise, sendo, porém, este foco deixado para possíveis trabalhos futuros. Também deve ficar claro que as técnicas básicas de criptoanálise são suficientes para uso em muitos casos, o que faz com que mesmo um sistema simples baseado nestas seja útil em situações reais.

## 4.1 O Quebra-pedra

O objetivo inicial do Quebra-pedra é ser um arcabouço Java para criptoanálise baseada em varredura de espaço de soluções voltado ao uso em ambiente de computação forense. Inicialmente busca-se obter a mínima funcionalidade necessária a esta tarefa. Para tanto, devemos então especificar o problema que gera a demanda por sua criação: o ataque a um texto cifrado por um criptoanalista em ambiente de

computação forense.

Neste problema, tipicamente, o criptoanalista tem acesso a um texto cifrado do qual pode ou não ter informações adicionais, tais como o algoritmo criptográfico utilizado, a linguagem do texto original, etc. Ou seja, o arcabouço deve ser capaz de auxiliar tanto um atacante que possui tais informações quanto aquele que não as tem. Além disso, deve ser flexível e simples de alterar pois o criptoanalista freqüentemente depara-se com novas situações em que sistemas sem estas capacidades tornam-se inúteis.

## 4.2 Projeto e Implementação

Para permitir um sistema simplificado de criptoanálise, tornou-se necessário o projeto e implementação de uma ferramenta extensível, portável, de alto desempenho e baixo custo. Estas são as premissas básicas do arcabouço Quebra-pedra. Seguindo estas premissas, foi escolhido o método de computação dinâmica para sua implementação, embora o arcabouço também possa ser utilizado para a geração dos dados necessários ao método de computação prévia (ver seção 2.1).

Mais especificamente, este arcabouço oferece, atualmente, as técnicas de ataque por força-bruta e busca dirigida por dicionário e dá ao criptoanalista a capacidade de extender o seu uso a qualquer problema de criptoanálise que possa ser atacado por essas técnicas.

### 4.2.1 Princípios básicos

Este arcabouço foi projetado em torno de dois princípios básicos para obter as características necessárias:

- independência de plataforma de execução;
- independência de algoritmos de criptografia (extensibilidade);

A independência de plataforma de execução é importante pois permite ao criptoanalista escolher qual processador (ou processadores) melhor se adapta à tarefa a ser realizada. Para se conseguir esta independência, optou-se pelo uso da linguagem de programação Java para a implementação do arcabouço. Esta linguagem permite, atualmente, a utilização do mesmo código em uma grande gama de processadores,

mostrando-se uma alternativa adequada aos requisitos de interoperabilidade do sistema. Esta característica também é importante pois houve, desde o início, a intenção de utilizar o arcabouço para uma implementação de criptoanálise paralela em ambientes heterogêneos, tais como redes locais ou de grande distância e, principalmente, em ambiente de grade computacional. Esta intenção vem da necessidade intensiva de processamento típica dos sistemas de criptoanálise.

Quanto à independência de algoritmos de criptografia, ainda no estágio inicial de projeto tornou-se claro que esta seria uma característica chave do sistema. Como exposto anteriormente, os sistemas criptoanalistas atuais não são facilmente adaptáveis a algoritmos diferentes daqueles para os quais foram projetados. Isto cria uma situação em que sua praticidade de uso torna-se baixa, pois, em computação forense, constantemente são necessárias buscas que envolvem novos algoritmos criptográficos. Ou seja, se a plataforma criptoanalítica não for adaptável, tornaria-se obsoleta rapidamente. Este problema foi resolvido, no Quebra-pedra, com o uso de conceitos disponibilizados pela linguagem Java e que são discutidos a seguir.

### 4.2.2 Implementação das características necessárias

Para o projeto inicial do arcabouço decidiu-se por centralizar a criação deste em torno da sua tarefa básica, ou seja, priorizar os ataques. Decidiu-se, então, iniciar o seu projeto em torno de algumas classes básicas que podem ser vistas no diagrama 4.1.

Abaixo tem-se a descrição destas classes conforme implementadas atualmente no arcabouço.

**Search -** classe que realiza os ataques por busca exaustiva e dirigida por dicionário. No caso da busca exaustiva, foram implementados métodos capazes de testar todas as possibilidades a partir de um dado conjunto de caracteres. Ou seja, o analista tem controle total sobre o alfabeto a ser utilizado no ataque. Já para a busca dirigida é necessário fornecer a localização dos arquivos contendo o dicionário a ser utilizado e de regras de modificação a serem aplicadas ao mesmo.

**Task -** classe que serve para a definição de uma tarefa de criptoanálise. Um objeto pertencente a esta classe possui todos os atributos necessários à realização de

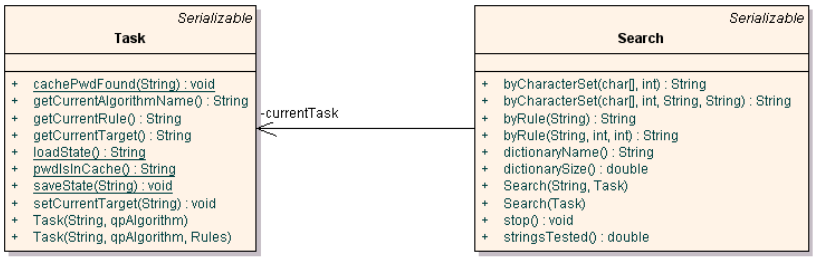



Figura 4.1: Diagrama de classes inicial do Quebra-pedra

um ataque, tais como qual algoritmo a ser usado e o texto cifrado. Esta classe também possui métodos para a criação e manipulação de uma base de dados de ataques realizados que é verificada antes de qualquer ataque. Isto evita que um ataque seja repetido e, portanto, economiza recursos nestes casos. Além disso, há também métodos para possibilitar a criação de pontos de verificação (*checkpoints*) que permitem que uma tarefa seja suspendida e depois reiniciada do mesmo ponto. Esta funcionalidade também busca economizar recursos evitando retrabalho.

Durante a implementação destas classes básicas, chegou-se à conclusão que outras seriam necessárias. No modelo final implementado três classes implementam o núcleo completo de ataques do arcabouço: *Task*, *Search* e *Rules*. A classe *Task* define uma tarefa de criptoanálise em termos de um alvo (texto cifrado) e do adaptador do algoritmo criptográfico que deve ser usado. A classe *Search* implementa os ataques

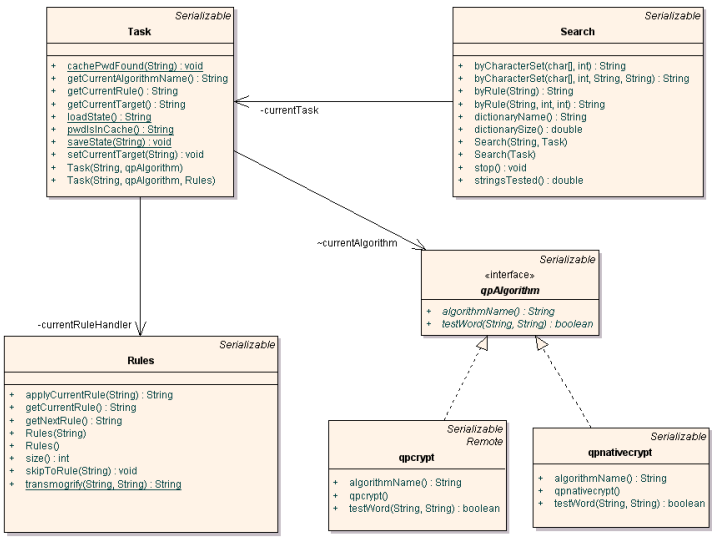



Figura 4.2: Diagrama de classes do núcleo de ataques do Quebra-pedra

disponíveis. Há métodos específicos para cada tipo de ataque e implementação (seqüencial/distribuída). Por fim, a classe *Rules* encapsula os métodos responsáveis pela manipulação e aplicação de regras de modificação de palavras para ataques de dicionário, quando estes são utilizados. No diagrama de classes da figura 4.2 podem ser vistos os métodos públicos correntemente implementados no arcabouço para cada uma destas classes do núcleo de ataques do arcabouço. Além destas classes também decidiu-se pela implementação de duas outras, *TextRecognition* e *Benchmark*, para a complementação das funcionalidades projetadas. A interface *qpAlgorithm* completa a lista inicial de classes do arcabouço. Abaixo encontra-se uma descrição mais detalhada de cada uma destas classes e da interface.

**Rules -** classe utilizada para os ataques por busca dirigida. Maiores detalhes sobre as regras que o arcabouço atualmente implementa são encontrados no apêndice B.

**qpAlgorithm -** *interface* para a criação de adaptadores que permitem a extensão dos ataques a qualquer algoritmo do qual se tenha uma implementação em C/C++/Java.

**TextRecognition -** classe que oferece alguns mecanismos para o reconhecimento de texto. Atualmente esta classe possui implementações de métodos estatísticos baseados em freqüência de dígrafos, tais como o logaritmo da verossimilhança de Sinkov. Maiores detalhes sobre estes métodos são discutidos em 4.2.2.1.

**Benchmark -** classe que serve para verificar o desempenho de um ataque e estimar o tempo necessário ao mesmo. Também serve para possibilitar um mecanismo de balanceamento de carga distribuído baseado na capacidade de processamento dos nós.

Um adaptador é um objeto de uma classe que deve implementar a interface *qpAlgorithm*. Um criptoanalista que pretende utilizar o Quebra-pedra com um algoritmo ainda não disponível no mesmo precisa somente programar este adaptador. No diagrama 4.2 podem ser vistos dois exemplos destes adaptadores, ambos para o algoritmo DES padrão do Unix, sendo que um deles utiliza uma versão Java deste

algoritmo e o outro a versão com implementação nativa em linguagem C. A figura 4.3 mostra o código Java de um adaptador simples para a versão Java. Já as figuras 4.4 e 4.5 mostram, respectivamente, o código Java de um adaptador simples para a versão nativa e o código C que a complementa.

Como vantagem adicional, este método isola o criptoanalista de detalhes de implementação da plataforma. Além disso, outra vantagem importante é o fato de que o método de verificação de sucesso na busca é desacoplado do arcabouço, pois fica no adaptador. Isto significa que o analista pode utilizar o sistema para tarefas simples como a quebra de senhas ou tarefas mais complexas, como a busca por chaves utilizadas para a criptografia de textos, arquivos compactados, fragmentos de arquivos e assim por diante. Nestes casos mais elaborados, mecanismos para reconhecimento de texto deverão ser utilizados, daí sua inclusão no arcabouço.

O arcabouço conta com um mecanismo simples, porém flexível para a implementação destes adaptadores. Este mecanismo baseia-se em dois princípios introduzidos na plataforma Java a partir da versão 1.3: o código reflexivo e a interface nativa Java (GOSLING et al., 2005). A reflexão é um conceito que permite, dentre outras coisas, a carga dinâmica de código em tempo de execução. Isto permite a modificação de um sistema para a adição de funcionalidade sem a necessidade de uma nova compilação do mesmo. Já a interface nativa Java (*Java Native Interface* - JNI) permite o uso de código nativo escrito em linguagem C/C++ que esteja na forma de DLL (em ambiente Microsoft Windows) ou de biblioteca de carga dinâmica .so (em ambiente Unix).

O uso destes conceitos em conjunto permite, portanto, a carga dinâmica de código nativo sem a necessidade de recompilação do sistema de criptoanálise. Para entendermos como este mecanismo funciona dentro da arquitetura Quebra-pedra, vejamos um exemplo: uma situação em que um criptoanalista depara-se com um conjunto de dados criptografados e, após uma análise inicial, chega à conclusão que foi utilizado um programa proprietário para esta criptografia. Analisando este programa, ele descobre que o mesmo utiliza um algoritmo criptográfico desconhecido. Porém, em sua análise do sistema, ele descobre uma biblioteca de carga dinâmica que contém uma implementação do algoritmo. Nesta situação o analista pode se utilizar das bibliotecas usadas pelo programa original para a busca da chave no Quebra-

```
import java.rmi.*;
import java.io.*;
import qp.qpAlgorithm;

public class qpcrypt implements qpAlgorithm, Serializable, Remote{
        public qpcrypt(){
                // construtor vazio, requisito do ProActive
        }

        public boolean testWord(String word, String target){
                String tstStr = new String(Crypt.crypt(target, word));
                if(tstStr.equals(target))
                        return true;
                else
                        return false;

        }

        public String algorithmName(){
                return "crypt";
        }
}
```

Figura 4.3: Um adaptador simples para utilizar uma implementação Java do algoritmo DES (crypt) no Quebra-pedra.

pedra. Para isto, somente é necessário realizar a programação de um adaptador para o Quebra-pedra e o sistema estará apto a auxiliá-lo em sua tarefa.

Este modelo de reuso de código nativo elimina, portanto, a tarefa de reprogramação ou mesmo recompilação do Quebra-pedra como um todo para seu uso com novos algoritmos de criptografia tornando-o totalmente adaptável de forma simples e rápida.

#### *4.2.2.1 Reconhecimento automático de texto*

Para possibilitar o uso do arcabouço em tarefas de criptoanálise mais complexas, o mesmo teve que ser extendido para poder realizar reconhecimento de texto. Sem este mecanismo seria impossível automatizar os ataques nos casos em que o analista não possui informações sobre o texto cifrado além da língua utilizada no mesmo. Por exemplo, supondo-se que o criptoanalista deseja atacar um arquivo criptografado, ele terá que realizar uma verificação visual do resultado de cada tentativa do sistema para verificar se o ataque obteve sucesso. Com o reconhecimento automático de

```
import java.io.*;

public class qpnativecrypt implements qpAlgorithm, Serializable{

        public qpnativecrypt(){
                // construtor vazio, requisito do ProActive.
        }

        public native boolean testWord(String word, String target);

        public String algorithmName(){
                return "nativecrypt";
        }

        static { System.loadLibrary("qpnativecryptImp"); }
}
```

Figura 4.4: Um adaptador simples para utilizar uma implementação nativa Unix do algoritmo DES (crypt) no Quebra-pedra.

texto ele pode programar o sistema para atacar o arquivo até que o resultado seja um texto em português ou em qualquer outra linguagem conhecida, ou até mesmo em algum formato específico como um arquivo de imagem por exemplo. A classe que implementa esta funcionalidade pode ser vista no diagrama da figura 4.6.

Para o reconhecimento de texto foi implementada a técnica de freqüência de dígrafos através do logaritmo da verossimilhança de Sinkov, demonstradamente o mecanismo com maior poder estatístico nestes casos (GANESAN; SHERMAN, 1993). Nesta técnica computa-se uma matriz com a freqüencia absoluta de ocorrência de cada par de caracteres. Esta matriz é chamada de matriz de co-ocorrência e através dela é calculado o logaritmo da verossimilhança. Se o método for aplicado a um texto padrão em uma linguagem, português por exemplo, e depois aplicado a um outro texto nesta mesma linguagem, o logaritmo da verossimilhança irá obter valores próximos para ambos. Ou seja, este logaritmo revela a quantidade de semelhança entre dois textos baseado na freqüência dos pares de caracteres de ambos. Embora esta técnica seja a de maior poder estatístico, ela exige uma interpretação do resultado que deve ser refeita para cada nova matriz de co-ocorrência. Além disso, ela exige a utilização de um texto padrão de comparação que deve forçosamente ser similar ao texto original criptografado, nem sempre algo trivial de se obter. Trata-se, portanto, de uma ferramenta que deverá receber atenção especial do criptoanalista e

```
#include <jni.h>
#include <unistd.h>
#include <string.h>
#include "qpnativecrypt.h"

#define TRUE 1
#define FALSE 0

JNIEXPORT jboolean JNICALL Java_qpnativecrypt_testWord (JNIEnv *env,
                                                        jobject obj,
                                                        jstring word,
                                                        jstring target) {
  /* JNI envia os parametros da chamada Java via variaveis de ambiente */
  const char *strword = (*env)->GetStringUTFChars(env,word,0);
  const char *strtarget = (*env)->GetStringUTFChars(env,target,0);

  /* cifrar string de teste usando chamada crypt e comparar com string alvo */
  if (strcmp(crypt(strword,strtarget), strtarget) == 0)
        return TRUE;
  else
        return FALSE;

}
```

Figura 4.5: Código C que complementa o adaptador da implementação nativa do algoritmo DES no Quebra-pedra.

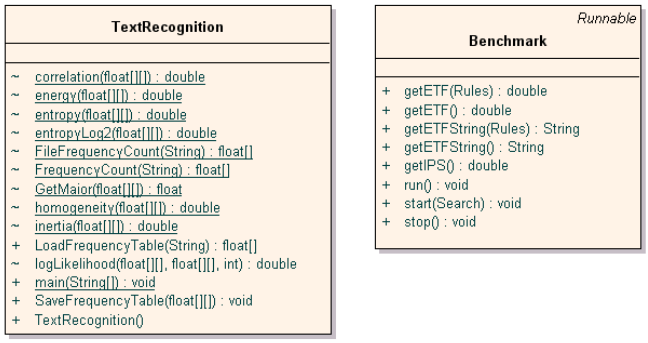



Figura 4.6: Diagrama de classes utilitárias do Quebra-pedra

conhecimento por sua parte de seu funcionamento para a obtenção de um resultado satisfatório.

Com o objetivo de simplificar o trabalho do criptoanalista e automatizar o processo de reconhecimento de texto, outras alternativas foram pesquisadas. Além do logaritmo da verossimilhança, também foram implementados no arcabouço vários outros coeficientes normalmente utilizados na área de processamento de imagens

para o reconhecimento de padrões. Para esta escolha, considerou-se que o reconhecimento de texto também é um problema de reconhecimento de padrões. Estes coeficientes são: entropia, inércia, energia, homogeneidade e correlação (HARALICK; SHANMUGAM; DINSTEIN, 1973; WALKER; JACKWAY; LONGSTAFF, 1995). Um estudo mais aprofundado do uso destes coeficientes para verificar seu grau de precisão e automatização no reconhecimento de texto deve ser feito posteriormente pelo autor.

### 4.2.3 Outras características

Outras características disponibilizadas pelo arcabouço são: salvamento automático do estado da execução, cache de sucessos e um sistema interno de análise de desempenho. A primeira permite ao sistema salvar automaticamente um ponto de referência (*checkpoint*) do estado de execução. Se por alguma razão o sistema for parado antes de obter sucesso ou chegar ao fim da busca, na próxima execução a busca será reiniciada deste ponto de referência.

Quanto à cache de sucessos, o arcabouço salva automaticamente todas as chaves encontradas. Esta cache sempre é verificada antes de se iniciar qualquer busca, evitando assim o desperdício de tempo e recursos com a repetição de trabalho já feito. Esta cache é externa ao arcabouço e pode, portanto, ser implementada como um simples arquivo de texto — atualmente em uso — ou utilizar um sistema gerenciador de banco de dados, à escolha do programador.

Finalmente, o sistema de avaliação de desempenho pode ser utilizado para vários fins, como a avaliação do tempo máximo que será despendido em uma busca ou como auxílio a um mecanismo de balanceamento de carga em uma implementação paralela através do cálculo da capacidade de processamento de um nó. A classe que implementa esta funcionalidade pode ser vista no diagrama da figura 4.6.

# 5 UM SISTEMA PARA CRIPTOANÁLISE COMPUTACIONAL DISTRIBUÍDA

*"If brute force isn't working...*
*you aren't using enough of it."*
(anônimo)

Tendo o arcabouço Quebra-pedra como ponto de partida e o *middleware* ProActive como infraestrutura de programação, podemos agora definir o projeto de um sistema criptoanalista distribuído utilizando processadores ociosos para criptoanálise forense através de uma estrutura de grade computacional. Esta seção descreve o projeto e a implementação de tal sistema.

## 5.1 Criptoanálise computacional de baixo custo

Em qualquer sistema distribuído, e em particular em um sistema criptoanalista, algumas características são desejáveis/requeridas:

**Utilizar ao máximo o poder computacional agregado disponível:** sendo um problema com enorme demanda de poder computacional, faz sentido que todo recurso disponível seja utilizado. Isto quer dizer que deve haver, no mínimo, um mecanismo de escalonamento de tarefas combinado a um bom sistema de balanceamento de carga para evitar desperdício de recursos.

**Capacidade de comunicação:** para avisar quando um resultado positivo for alcançado ou requisitar novas tarefas caso contrário. Em sistemas com muitos nós disponíveis, algum método deve ser utilizado para detectar/evitar o congestionamento da rede de comunicação.

**Segurança:** para evitar que uma chave descoberta seja visível a alguém sem permissão para tal, e também para evitar sabotagens ao sistema (informando uma

chave falsa como se fosse a correta, por exemplo).

Some-se a cada um destes requisitos a necessidade de baixo custo e o projeto de tal sistema fica ainda mais complexo. Vejamos, então, como tentar obter estas características.

### 5.1.1 Escalonamento e balanceamento de carga

A computação paralela tem se tornado mais comum com a popularização dos aglomerados de computadores produzidos em larga escala e/ou a exploração de ciclos ociosos em computadores interconectados. Tais sistemas oferecem um poder de processamento teórico igual à soma das capacidades de seus recursos individuais. Entretanto, para obter um desempenho próximo a este limite teórico, é necessária a distribuição efetiva da carga de trabalho (*workload*) sobre os recursos disponíveis, obtendo o máximo de lucro desta multiplicidade. Esta otimização da distribuição da carga é conhecida como o problema do escalonamento (CALZAROSSA; MASSARI; TESSERA, 2000).

O problema do escalonamento se torna mais complexo ao passo que o conjunto de recursos computacionais é compartilhado dinamicamente entre diversos usuários e aplicações. Em um ambiente formado por um conjunto heterogêneo de recursos, que podem variar no tempo e espaço, para satisfazer requerimentos de desempenho o escalonador deve orquestrar corretamente a distribuição dinâmica da carga de trabalho sobre estes recursos. Nestes casos o problema do escalonamento não pode ser resolvido otimamente devido à dinâmica do próprio ambiente e mecanismos limitados de medição. Modelos estocásticos podem ser mais adequados que os determinísticos (SANTOS; PROENÇA, 2001).

O escalonador pode ser colocado no nível de sistema ou no nível de aplicação. Escalonamento em nível de aplicação é definido como aquele que é realizado pela própria aplicação, procurando atingir objetivos internos de desempenho, enquanto compete com outras aplicações que podem compartilhar o mesmo sistema distribuído. Já o escalonamento em nível de sistema segue a abordagem oposta: um único escalonador controla todos os recursos e aplicações, e faz todas as decisões de alocação. Colocando o escalonador no nível de aplicação, o programador pode incorporar ao mesmo o seu conhecimento sobre o comportamento e requerimentos

da aplicação. O desenvolvimento da aplicação, entretanto, torna-se mais complexo (SCHOPF; BERMAN, 1999; BERMAN et al., 1996).

Escalonar recursos em um sistema paralelo ou distribuído é um problema de escalonamento bi-dimensional (BAUMGARTNER; WAH, 1991). O escalonamento local, ou intranodo, preocupa-se com o escalonamento dentro de um único nodo. Ele ocorre em vários sub-níveis: hierarquia de memória, dispositivos, e entre processadores quando o nodo é multiprocessado. Este tipo de escalonamento é realizado pelo sistema operacional de cada nodo. O escalonamento global, ou internodo, atua em um nível superior e está preocupado com o escalonamento entre vários nodos.

Várias características de um sistema causam impacto no escalonamento de um sistema distribuído, tais como a capacidade absoluta dos recursos disponíveis e seu estado atual. Estas características, portanto, definem o ambiente sobre o qual o escalonador atuará. O ambiente gera eventos, normalmente informados ao escalonador na forma de mensagens relacionadas à carga de trabalho. Exemplos de eventos são a chegada de novas tarefas, remoção de tarefas antigas ou informação sobre o estado atual de uma tarefa.

Para melhorar sua efetividade, a maioria dos escalonadores mantém um modelo interno de execução do ambiente e seu estado corrente, que é atualizado sempre que há uma nova informação sobre o sistema distribuído ou sobre a carga de trabalho. Esta informação normalmente está na forma de alguma medida. Estas medidas, feitas por agentes ou sensores executados nos computadores que compõem o sistema, determinam, então, os aspectos que podem influenciar as decisões do escalonador. A variedade das medidas e sua precisão estão fortemente relacionadas à eficácia do escalonador.

A partir do modelo interno de execução, dos eventos gerados pelo sistema e das medidas feitas, o escalonador tem uma visão do que está acontecendo no sistema como um todo. A partir deste momento, sua tarefa passa a ser atender aos requisitos de desempenho. Tais requisitos são o motivo da existência do escalonador e podem incluir um ou mais objetivos - maximização da vazão, minimização do tempo de execução, confiabilidade do sistema, etc.

Portanto, o escalonamento pode ser definido como o mapeamento de tarefas nos recursos disponíveis realizado pelo escalonador. Este mapeamento pode ser gerado

totalmente antes da execução - escalonamento estático - se todas as tarefas e características dos sistemas são conhecidas, ou pode ser gerado em tempo de execução por um conjunto de regras - escalonamento dinâmico. Estas regras especificam ações de correção que redistribuem tarefas sobre o sistema para que haja uma maior aproximação de algum objetivo de desempenho. Esta capacidade é denominada balanceamento de carga.

Para poder fazer decisões e verificar sua eficiência e/ou efetividade, um escalonador dinâmico baseia-se, em geral, em um conjunto de medidas. Estas medições podem ser feitas pelo sistema operacional ou pela própria aplicação e são, usualmente, baseadas em instrumentação estática, ou seja, são instruções inseridas no código.

No caso de serem feitas pelo sistema operacional, utilizam-se medidas genéricas na tentativa de torná-las o mais abrangentes possíveis. Exemplos de tais medidas são o nível de utilização de memória ou processador, tamanho de filas de espera por dispositivos, largura de banda de comunicação, etc. Já quando as medidas são feitas por um escalonador em nível de aplicação, podem ser utilizadas métricas voltadas à realização de tarefas específicas desta. A vantagem, neste caso, é a possibilidade de se obterem resultados que sejam mais relevantes à aplicação. Esta aproximação, no entanto, pode impor uma sobrecarga adicional se a métrica a ser computada não contribui diretamente com a tarefa a ser realizada ou não pode ser facilmente gerada em sua execução.

Qualquer que seja o método escolhido, portanto, o código que realiza as medições deve ser cuidadosamente projetado para que não interfira na funcionalidade da aplicação ou gere sobrecarga demasiada pelo seu uso. A quantia de interferência gerada por este código é conhecida como nível de intrusão. Deve haver um balanceamento cuidadoso entre este e a acurácia e relevância das medidas (SANTOS, 2001).

### 5.1.2 A busca por uma boa abstração de comunicação

De acordo com (MAINWARING; CULLER, 1996), uma abstração crítica para o processamento paralelo e distribuído permanece sem solução: uma *interface* de comunicações simples e de propósito geral. Esta *interface* deve disponibilizar primitivas que formam um conjunto de instruções portável para comunicacões. As

primitivas devem ser mapeadas eficientemente na camada de rede de baixo nível (*interface* de rede) e compor eficientemente protocolos de alto nível e aplicações. O aspecto crucial destas *interfaces* é sua integração com o sistema operacional. Para promover a construção de sistemas de propósito geral em larga escala, elas devem suportar multiprogramação protegida em ambiente com grande número de usuários e recursos finitos de rede.

### 5.1.3 Privacidade da informação

Como geralmente as mensagens trocadas pelos elementos de um sistema distribuído devem trafegar pela mesma rede de comunicação que interliga os nós, surgem algumas questões: como manter estas mensagens privadas e, principalmente, como evitar ataques realizados por terceiros com o intuito de influir no resultado do processamento? Tais questões ganham maior importância em um sistema que será utilizado em ambiente forense, pela própria natureza deste.

Felizmente, o *middleware* ProActive disponibiliza um grande conjunto de recursos de segurança. Ele oferece desde autenticação, integridade e confidencialidade de comunicações até recursos de alto nível, tais como a migração segura de objetos, políticas de segurança dinamicamente negociadas e hierárquicas. Todos estes recursos podem ser utilizados de forma transparente pelas aplicações (BADUEL et al., 2006).

## 5.2 Primeiros testes

Para prova de conceito e demonstração do potencial do arcabouço Quebra-pedra, foi feita uma implementação de um sistema criptoanalista seqüencial não-distribuído (*stand-alone*) que o utiliza. A figura 5.1 mostra a arquitetura desta implementação.

Basicamente esta implementação explora diretamente as funcionalidades mínimas do arcabouço necessárias à realização de ataques. A classe principal - `qpseq` - é responsável por obter os dados necessários, criar os objetos que possibilitem a busca e monitorar o resultado da mesma. A figura exemplifica um caso em que há dois adaptadores disponíveis para buscas: um para o algoritmo DES padrão dos sistemas Unix (`crypt`) e outro para o algoritmo de hash MD5.

O qpseq, como citado anteriormente, é capaz de realizar ataques criptoanalíticos

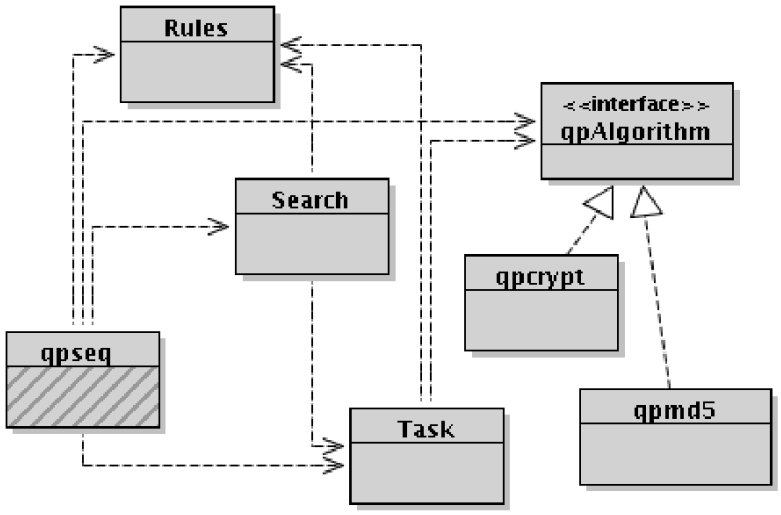



Figura 5.1: Diagrama de classes do núcleo de ataques do qpseq

de forma seqüencial. O usuário fornece um texto-alvo criptografado e algumas informações de configuração dependentes do tipo de ataque a ser realizado. Isto é feito definindo-se uma tarefa. Se estiver sendo feita uma busca exaustiva, é necessário fornecer uma tabela de caracteres a serem utilizados para geração das palavras de teste e a largura das mesmas. Se for uma busca por dicionário, deve ser informada a localização da lista de palavras e das regras de modificação. A partir de então, entra em execução o mecanismo de ataque do arcabouço. Após esta preparação, inicia-se o laço principal que consiste em: gerar uma combinação a ser testada (palavra-teste); utilizar o módulo de criptografia para efetuar a comparação entre a palavra-teste e o texto-alvo no adaptador. Os algoritmos DES e MD5 deste teste são aqueles utilizados pelos sistemas do tipo Unix para criptografia de senhas de usuários, portanto, nestes casos, o adaptador simplesmente criptografa a palavra-teste com o algoritmo adequado e compara o resultado com o texto-alvo. Se forem iguais, a palavra-teste é a chave, se não, passa-se à próxima combinação.

Os testes realizados com o `qpseq`, que encontram-se detalhados na seção 6, demonstraram a viabilidade do arcabouço para uso em situações reais de criptoanálise.

## 5.3 Projeto e implementação de um sistema criptoanalista distribuído

Primeiramente, analisando o problema em questão, podemos visualizar algumas importantes características do mesmo:

**Facilmente divisível -** tanto a busca exaustiva quanto a busca dirigida por dicionário oferecem várias opções de divisão de tarefas. Pode-se obter granularidade de tarefas praticamente em qualquer tamanho desejado. Os únicos limites impostos são aqueles dados pelo algoritmo de criptografia utilizado para cifrar os textos.

**Grãos independentes -** os grãos de processamento neste problema são naturalmente independentes, pois trata-se simplesmente de porções disjuntas do espaço de soluções. Dentre outras coisas, isto elimina os problemas de bordas ou condições de contorno. Um dos efeitos disto é a simplificação de migração de código durante a execução.

**Baixa necessidade de comunicação -** como os grãos de processamento são independentes e eventuais nodos de processamento distribuído não necessitam de nenhuma informação adicional para realizar a sua parte da tarefa de criptoanálise, as únicas comunicações necessárias são para envio de tarefas e, eventualmente, retorno de chave em caso de sucesso.

**Alta necessidade de processamento -** a criptoanálise através de busca exaustiva ou dirigida demanda forçosamente uma enorme quantidade de processamento para ser realizada. Isto se deve ao fato de que os algoritmos de criptografia são, via de regra, projetados especificamente para combater estes tipos de ataques.

**Baixa necessidade de armazenamento -** devido à escolha de projeto do sistema ter recaído na utilização de criptoanálise dinâmica, não é necessário armazenar dados nos nodos de processamento.

Para o projeto do sistema optou-se por uma abordagem modular. O sistema pode ser decomposto em alguns módulos principais: geração de tarefas, balanceamento de carga ou escalonamento, monitoramento e controle, interface de usuário e execução de tarefas, como pode ser visto na figura 5.2.

Por se tratar de uma implementação distribuída, estes módulos foram divididos de tal forma a simplificar a implementação. Como optou-se pela implementação em modelo mestre-escravo, separou-se o sistema em duas grandes partes: Gerenciador

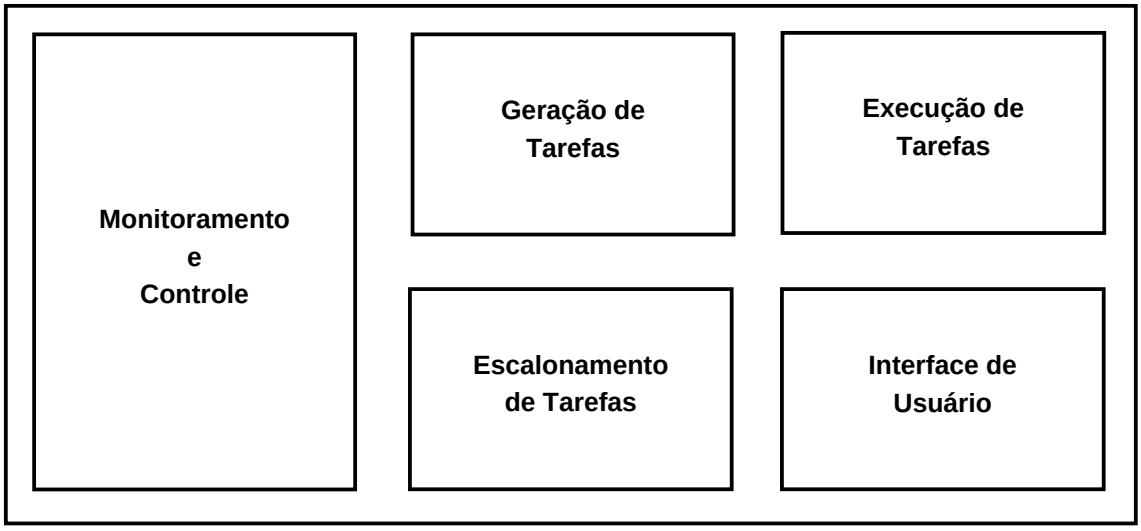


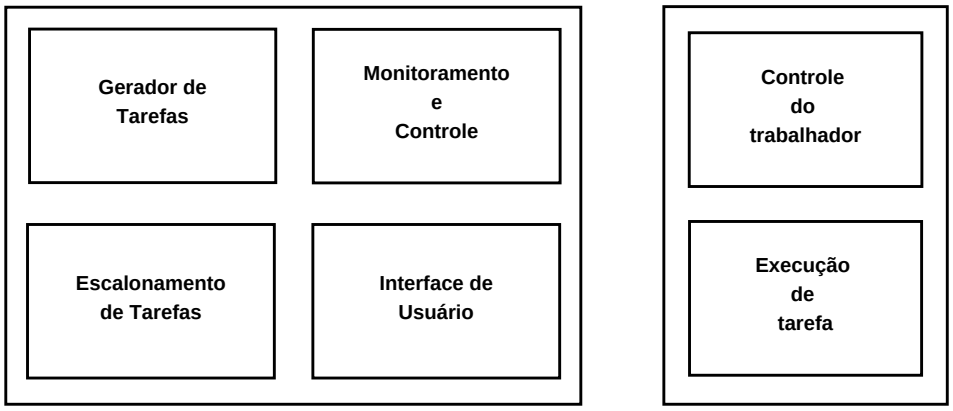

Figura 5.2: Um sistema distribuído de criptoanálise computacional para uso forense

e Trabalhador. O Gerenciador, como pode ser visto na figura 5.3(a), engloba as funções de controle e monitoramento da execução distribuída e o Trabalhador, conforme a figura 5.3(b), engloba as funções relativas à execução do processamento do ataque. Tipicamente o Gerenciador é executado localmente a um nó de controle e o módulo Trabalhador é replicado em cada nó disponível à execução distribuída. A seguir são vistos detalhes de cada um dos módulos internos destas entidades.

### 5.3.1 Geração de Tarefas

O módulo gerador de tarefas é responsável pela criação da carga de trabalho que cada processador recebe, portanto, o mesmo visa maximizar a utilização de cada processador. Para que as tarefas criadas pelo gerador alcancem este objetivo, é necessário que o mesmo trabalhe em conjunto com o sistema de escalonamento. Na verdade, o gerador de tarefas é subordinado ao escalonador, pois atende às requisições diretas deste. A existência do gerador é justificada pela necessidade de geração dinâmica de tarefas de acordo com os pedidos realizados pelo módulo de ataques em um determinado período de tempo.

O objetivo principal do gerador de tarefas é criar os grãos de processamento que serão depois distribuidos pelo escalonador para os nodos que irão executá-los. O arcabouço Quebra-Pedra possui o suporte necessário a esta operação oferecendo, por exemplo, a capacidade de processar faixas arbitrárias do espaço de buscas que são enviadas pelo gerenciador a cada nodo para execução de um ataque.




(a) Gerenciador (b) Trabalhador

Figura 5.3: Componentes principais do sistema

### 5.3.2 Escalonamento

A função principal de um escalonador em um sistema distribuído é gerenciar eficientemente a distribuição de carga de trabalho entre os nós que o compõe. Em um sistema que busca utilizar o poder de processamento de processadores ociosos algumas considerações especiais devem ser feitas. Neste caso, por exemplo, não se pode precisar quantos nós o sistema terá a sua disposição previamente. Como os recursos computacionais sofrem mudanças dinamicamente - processadores podem sofrer falhas ou serem necessários à aplicações de maior prioridade, por exemplo - as aplicações devem ser tolerantes a falhas e maleáveis (KALÉ; KUMAR; DESOUZA, 2002), ou seja, capazes de lidar com um número flutuante de processadores. Esta característica fundamental de dinamicidade faz com que o escalonador deva ser capaz de reagir à situação atual do sistema. Esta reação, entretanto, não pode ser instantânea, senão o sistema poderia entrar em um ciclo infinito de tentar continuamente se adaptar a novas situações ao invés de realizar alguma tarefa.

O modelo de escalonamento dividir-para-conquistar (*divide-and-conquer*) é uma generalização do modelo mestre-escravo recomendada pelo *Global Grid Forum* como um método eficiente de escrita de aplicações em grade (LEE et al., 2001). O paradigma mestre-escravo tipicamente utiliza somente um processo (mestre) para repartir o trabalho, o que restringe o tipo de aplicação que pode ser implementado e cria um gargalo de desempenho. Já o paradigma dividir-para-conquistar divide as tarefas de forma recursiva.

A adição de processadores ao sistema não traz maiores preocupações, devido ao

modelo de escalonamento escolhido, no qual os processadores que entram simplesmente irão receber novas tarefas. Entretanto, a saída de processadores durante a execução de alguma tarefa cria o problema de como terminá-la. Além disso, em sistemas do tipo dividir-para-conquistar isto pode criar as chamadas tarefas órfãs, que são tarefas que dependem de outras realizadas por processadores que se tornaram indisponíveis durante a computação. Uma solução simples é reiniciar a tarefa pendente em outro processador. Isto acaba reduzindo o desempenho do sistema, pois resulta em redundância de computação. Esta redundância é agravada quanto maior for a dinamicidade de alteração da grade em uso. Uma forma de minimizar este problema é a utilização da técnica de migração, na qual tarefas podem ser realocadas dinamicamente. Obviamente tal mecanismo somente pode ser utilizado quando um processador avisa o sistema de sua saída iminente. Quando um processador sai do sistema por falha seu trabalho terá que ser recomputado, mas somente nestes casos haverá redundância de trabalho. Conforme visto na seção 2.2.2, o ProActive conta com um mecanismo avançado de migração que pode ser utilizado nestes casos. Porém, é necessário avaliar se o custo da migração de tarefas é vantajoso em relação a simplesmente refazê-las.

Com base nisto, o escalonador do sistema foi projetado para tentar manter os nós de um sistema distribuído dinâmico processando alguma tarefa útil no máximo de seu tempo livre possível, sem, porém, tentar adaptar-se instantaneamente à dinamicidade dos nós. Uma forma simples de fazer isto é fazer com que cada nó receba tarefas relativamente pequenas e sempre tente cumprí-las até o fim, evitando a migração de tarefas inacabadas que é um possível foco de problemas neste tipo de sistema.

Para que este mecanismo trivial de escalonamento funcione adequadamente, o escalonador precisa obter, de cada nó, dados sobre o volume de carga adequado aos mesmos. Como esta é uma característica importante ao próprio processo de escalonamento de, provavelmente, qualquer sistema que utilize o arcabouço Quebra-pedra, foi introduzido no arcabouço um subsistema para a coleta de medidas de desempenho (*benchmarking*) de um nó de processamento. Como visto na seção 5.1.1, o escalonamento em nível de aplicação permite justamente esta utilização de medidas voltadas à tarefa sendo atacada pelo sistema distribuído, resultando,

portanto, em um escalonamento de tarefas de melhor qualidade.

### 5.3.3 Monitoramento e Controle

O controle da aplicação baseia-se em troca de mensagens entre o gerenciador e os trabalhadores. Por exemplo, quando algum nó encontra a chave correta, o gerenciador automaticamente envia uma mensagem de controle para cada nó terminando sua execução.

As funções de monitoramento são bastante simples e não-intrusivas para evitar ao máximo o aumento de sobrecarga na execução de um ataque. Somente ao final do processamento de um bloco por um nó são atualizadas as informações deste nó junto ao gerenciador.

### 5.3.4 Interface de Usuário

Responsável pela interação como o usuário final do sistema, esta interface oferece uma forma única de execução, controle e monitoramento do sistema. A interface em uso é bastante simples, refletindo o estágio atual do sistema. A figura 5.4 mostra uma das interfaces iniciais do módulo de controle.

Com a eventual evolução do sistema, deve-se procurar a criação de uma identidade visual para simplificar a memorização, pelos usuários, das principais capacidades oferecidas pelo sistema.

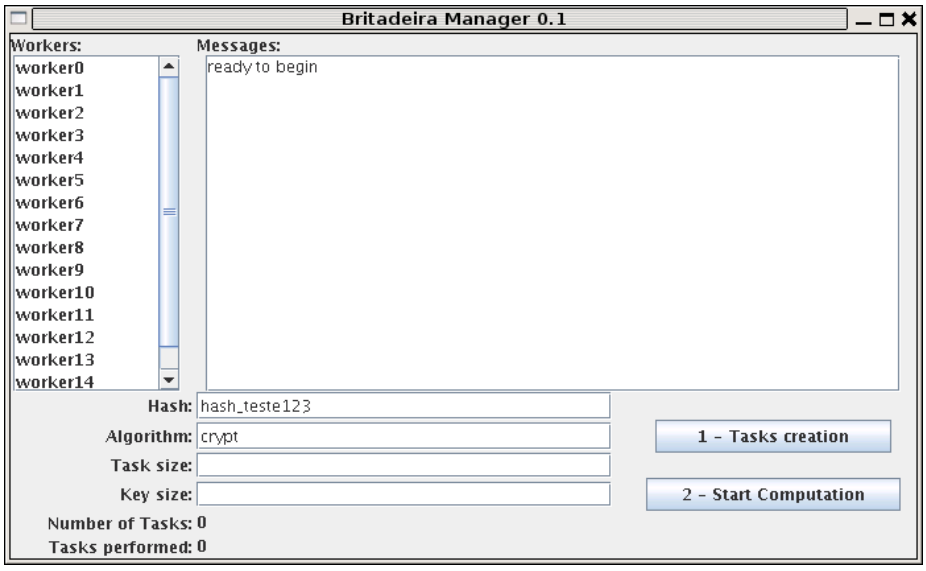



Figura 5.4: Interface simples de gerenciamento do sistema

### 5.3.5 Execução de Tarefas

Este módulo é responsável pela execução das tarefas que compõe um ataque. Portanto, é este módulo que utiliza o arcabouço Quebra-Pedra. Para a sua implementação buscou-se evitar ao máximo qualquer sobrecarga que influencie esta tarefa principal. Por exemplo, após o início de uma busca pelo módulo nenhuma tarefa de monitoramento e/ou controle é disparada em paralelo para evitar a postergação dos resultados.

### 5.3.6 O Sistema `qppro`

Conforme visto anteriomente, o sistema implementado, denominado `qppro`, possui dois tipos de processos principais: Gerenciador (*qpproManager*) e Trabalhador (*qpproWorker*). Atualmente o Gerenciador é controlado por uma interface gráfica simples e, a partir dos parâmetros dados pelo usuário, cria e distribui tarefas ao conjunto de trabalhadores. O diagrama 5.5 mostra a implementação deste gerenciador.


Figura 5.5: Diagrama de classes do gerenciador

Os trabalhadores são executados, em geral, remotamente, embora nada impeça a utilização de trabalhadores locais. Cada trabalhador recebe uma tarefa e a processa utilizando o arcabouço Quebra-pedra. Como visto anteriormente, as tarefas são relativamente pequenas para minimizar os efeitos de uma eventual perda de processador e também para evitar a necessidade de utilização de mecanismos de migração de tarefas. Ou seja, atualmente, o sistema é puramente do tipo mestre-

escravo. O escalonamento é do tipo FIFO ou FCFS (*first come first served*) e são usados grãos pequenos de processamento. O diagrama 5.6 mostra a implementação deste trabalhador.

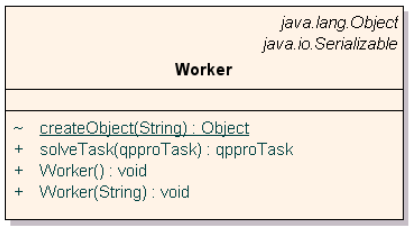



Figura 5.6: Diagrama de classes do trabalhador

O diagrama 5.7 ilustra o fluxo de mensagens entre o processo gerenciador e um processo trabalhador. Como pode ser constatado neste diagrama, o gerenciador é responsável pelo lançamento remoto do trabalhador. Assim que o trabalhador encontra-se inicializado, o mesmo recebe uma tarefa. Quando a tarefa está pronta, o trabalhador envia o resultado ao gerenciador. Este ciclo de envio de tarefas e recebimento de resultados é repetido enquanto houverem tarefas disponíveis para o trabalhador ou até que haja sucesso na tarefa de busca da chave. É importante ressaltar que as mensagens entre o gerenciador e o trabalhador são assíncronas. Essa característica garante que o gerenciador poderá trabalhar sem interrupções em suas tarefas, o que se torna importante em sistemas com muitos trabalhadores a serem gerenciados. Características como segurança e tolerância à falhas são deixadas a cargo do *middleware* ProActive.

De forma resumida, tipicamente uma execução do sistema seguirá os seguintes passos. Durante a inicialização, os trabalhadores locais e remotos são disparados. A seguir, o usuário fornecerá os parâmetros do ataque a ser realizado. O sistema irá verificar os mesmos e, se tudo estiver correto, começar a preparação para o ataque. Durante a preparação o gerador de tarefas é ativado, criando tarefas iniciais de mesmo tamanho para cada trabalhador ativo. Na primeira fase do ataque cada trabalhador recebe sua tarefa e, depois de processá-la, envia ao gerenciador o pedido de mais tarefas que inclui o seu desempenho durante a tarefa inicial em termos de iterações por segundo. O escalonador, baseado nas informações de desempenho dos trabalhadores, irá acionar o gerador de tarefas para dar início à segunda fase do ataque em que são criadas tarefas sob demanda específica para cada trabalhador.

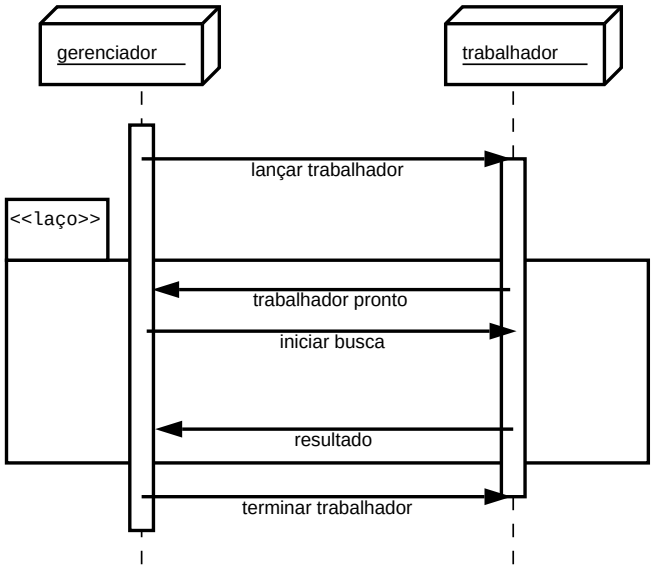



Figura 5.7: Fluxo típico de mensagens entre o gerenciador e um trabalhador no `qppro`

O módulo de monitoramento e controle, também com base nas informações de desempenho dos nós, irá, a partir desta segunda fase, gerar uma estimativa de tempo total para o ataque. Este ciclo se repete até a obtenção de sucesso no ataque ou o fim do espaço de buscas.

# 6 RESULTADOS

Nesta seção são mostrados os resultados gerais, em termos de desempenho, obtidos com o arcabouço e com a implementação do sistema de criptoanálise distribuída.

## 6.1 Avaliação de desempenho do Quebra-pedra

A linguagem Java é freqüentemente citada como tendo baixo desempenho devido ao seu caráter interpretado. Foram encontrados alguns estudos que a avaliaram neste aspecto (BULL et al., 2001; MOREIRA et al., 2000; BESSET, 2000) e chegaram à conclusão oposta, ou seja, que a linguagem evoluiu para um modelo onde o desempenho é tão satisfatório quanto o de outras linguagens consideradas rápidas. Porém, nenhum estudo de desempenho na área de criptoanálise foi encontrado. Para verificar a viabilidade do arcabouço foram feitas análises de seu desempenho na realização de ataques. Estas análises basearam-se na execução de ataques utilizando a implementação qpseq e dois algoritmos criptográficos: DES e MD5.

Para haver uma quantificação dos resultados definiu-se que seria realizado um teste comparativo entre a implementação Java e uma implementação em linguagem C ou C++. Foi realizada, portanto, a programação desta implementação de referência. Para evitar influência da latência de entrada e saída (I/O), optou-se pela implementação de um programa específico para ataques por busca exaustiva em linguagem C. Este programa replica o funcionamento básico do arcabouço para ataques de força-bruta. Duas versões do mesmo foram geradas, uma para o algoritmo DES e outra para o MD5.

O passo seguinte foi a realização de medições de tempos de execução destas implementações. Foram escolhidas cinco arquiteturas para a realização destas medições:

- Arquitetura A: processador Pentium 4, 2.4GHz, S.O. Windows XP, Sun J2SE versão 1.4.2;
- Arquitetura B: processador Pentium 4, 2.4GHz, S.O. Linux *kernel* 2.6.20, Sun J2SE versão 1.5.0.11;
- Arquitetura C: processador Pentium 3, 1GHz, S.O. Linux *kernel* 2.6.5, Sun J2SE versão 1.4.2;
- Arquitetura D: processador Athlon, 1.4GHz, S.O. Linux *kernel* 2.6.5, Sun J2SE versão 1.4.2;
- Arquitetura E: processador Celeron D 310+, 2.13GHz, S.O. Linux *kernel* 2.6.19, Sun J2SE versão 1.5.0.11.

Analisaremos, agora, alguns dos resultados obtidos nestas medições. Todas os resultados aqui mostrados são valores médios de várias execuções (no mínimo 10). A tabela 6.1 mostra os valores médios de iterações por segundo das implementações de referência, denominada qpseqC.

Tabela 6.1: Número médio de iterações por segundo (i/s) das implementações de referência

| **qpseqC** | **A** | **B** | **C** | **D** | **E** |
|---|---|---|---|---|---|
| DES | 82312 i/s | 111261 i/s | 57340 i/s | 52599 i/s | 26880 i/s |
| MD5 | - | 1338 i/s | 887 i/s | 1384 i/s | 1704 i/s |

Após esta definição do padrão de controle, prosseguiu-se à medições da implementação qpseq (Java). Nesta etapa foram coletados resultados utilizando-se diversas implementações de adaptadores e versões dos algoritmos de criptografia. Durante esta coleta, uma clara desvantagem das implementações Java dos algoritmos de criptografia foi revelada. Buscando uma melhora no desempenho da aplicação, foram realizadas análises da execução do código Java com o uso de ferramentas de análise de perfil (*profiling*). Como esperado, estas ferramentas demonstraram que o maior tempo de execução é gasto com a execução do módulo de criptografia.

Novas medições foram realizadas, agora com a utilização de chamadas a código nativo através da interface Java JNI. Nestes casos, o adaptador, ao invés de utilizar-se de um método Java para a criptografia da palavra-teste faz uma chamada à código

nativo, presente em uma biblioteca de carga dinâmica, para esta criptografia. Nestas novas medições os resultados mostraram-se bastante superiores, em média, às melhores implementações Java para os algoritmos analisados. A tabela 6.2 mostra os valores relativos entre o número médio de iterações por segundo obtido na implementação de referência e o obtido na implementação em Java. Pode-se observar que, para o algoritmo DES, a implementação que utiliza a linguagem Java de forma puramente interpretada (JRE) foi, em média, 2,81 vezes mais lenta que a implementação de referência e a versão que utiliza chamadas à código nativo através da JNI foi 1,13 vezes mais lenta que a implementação nativa. Para efeito de comparação realizou-se novo teste, desta vez com o algoritmo MD5 nas máquinas B, C, D e E (por serem as únicas que possuem as implementações em Java e nativa deste algoritmo atualmente). A partir dos dados na mesma tabela pode-se observar que, para este algoritmo, a versão interpretada é extremamente lenta quando comparada à implementação de referência e a versão que utiliza chamadas ao código nativo C, ao contrário, obteve desempenho bastante aproximado ao da linguagem C pura.

Tabela 6.2: Tempos médios relativos de execução entre qpseq a implementação de referência para os algoritmos DES e MD5

| **Implementação** | **A** | **B** | **C** | **D** | **E** | **Média** |
|---|---|---|---|---|---|---|
| DES JRE | 1,63 | 5,31 | 3,30 | 1,83 | 1,98 | 2,81 |
| DES JRE+JNI | 1,15 | 1,01 | 1,21 | 1,03 | 0,88 | 1,13 |
| MD5 JRE | - | 11,55 | 17,05 | 14,55 | 19,20 | 15,59 |
| MD5 JRE+JNI | - | 1,00 | 1,00 | 1,00 | 1.01 | 1,00 |

Como os valores mostrados na tabela 6.2 foram bastante divergentes em relação à velocidade relativa entre a implementação em C e a implementação Java, chega-se à conclusão que detalhes de arquitetura tanto de processadores quanto de sistema operacional possuem grande influência sobre o seu desempenho. Por exemplo, os processadores A e B são idênticos, a única diferença é o sistema operacional, já as máquinas C e D executam o mesmo sistema operacional. Em ambos os casos, o número de iterações por segundo medido variou consideravelmente entre estas arquiteturas, exceto no caso da implementação nativa do algoritmo MD5.

Destes testes conclui-se que o algoritmo sendo utilizado possui grande influência sobre o desempenho final da aplicação e que a linguagem Java interpretada é, em média, mais lenta do que a linguagem C para o fim específico de criptoanálise.

Porém, quando é utilizado o mecanismo de execução de código C nativo através da JNI, a implementação do sistema em Java torna-se uma alternativa bastante atrativa. Se levarmos em conta outras características inerentes à linguagem, tais como simplicidade de programação e portabilidade, o problema representado por esta sobrecarga torna-se praticamente irrelevante.

## 6.2 Análise de desempenho do sistema de criptoanálise distribuída

Vários testes foram realizados com a implementação distribuída do sistema de criptoanálise para a verificação de seu desempenho. Durante estes testes buscou-se minimizar qualquer influência externa ao sistema isolando-se a rede onde o mesmo foi executado, sempre que possível. Para estes testes, a versão distribuída do sistema de criptoanálise foi denominada `qppro`. Este nome foi escolhido para denotar o uso do arcabouço Quebra-pedra e do *middleware* ProActive. Todos os resultados aqui mostrados são valores médios de várias execuções.

Inicialmente, vários ajustes de configuração nos ambientes de execução foram realizados buscando otimizar a execução dos ataques criptoanalíticos remotamente. Um exemplo é a utilização do parâmetro *server* na máquina virtual java. Os resultados desta simples modificação de configuração podem ser vistos na tabela 6.3.

Tabela 6.3: Comparação entre qppro executando em JVM com parâmetros *client* e *server*. Bloco de 390.500 iterações, host orelhano.

| | JVM *-client* | JVM *-server* |
|---|---|---|
| qppro | 19,75 s | 12,68 s |

O passo seguinte foi a comparação do `qppro` com o `qpseq`. A tabela 6.4 mostra os primeiros resultados obtidos em tal comparação. Como pode-se notar nesta tabela, há uma discrepância no sentido de que os valores medidos são, em média, menores para a execução remota via ProActive do que para a execução local. Após verificação das medições, que resultou na mesma discrepância, foi modificado o protocolo de testes para a realização do teste seqüencial em um grande bloco de 16.564.000 iterações. Foi medido o tempo de execução deste bloco pelo qpseq e após, normalizou-se os resultados para blocos de 390.500 iterações para comparação com os valores anteriormente obtidos. Tais resultados encontram-se na tabela 6.5.

Tabela 6.4: Comparação entre qppro com um único trabalhador remoto e qpseq para o algoritmo DES. Blocos de 390.500 iterações.

| | Host tatuira | Host paple |
|---|---|---|
| qpseq | 13,42 s | 13,51 s |
| qppro | 12,68 s | 13,41 s |

Tabela 6.5: Comparação entre qppro com um único trabalhador remoto e qpseq para o algoritmo DES normalizado para blocos de 390500 iterações.

| | Host tatuira | Host paple |
|---|---|---|
| qpseq normalizado | 12,46 s | 12,83 s |
| qppro | 12,68 s | 13,41 s |

Os tempos mostrados após a utilização da normalização levam a crer que a explicação para esta discrepância deve-se à uma melhor utilização dos mecanismos internos da máquina virtual Java - em especial o sistema de caches do HotSpot - quando de seu uso em tarefas mais longas. Ou seja, na implementação seqüencial, a linguagem Java consegue melhor desempenho por iteração quando a máquina virtual executa um maior número destas iterações.

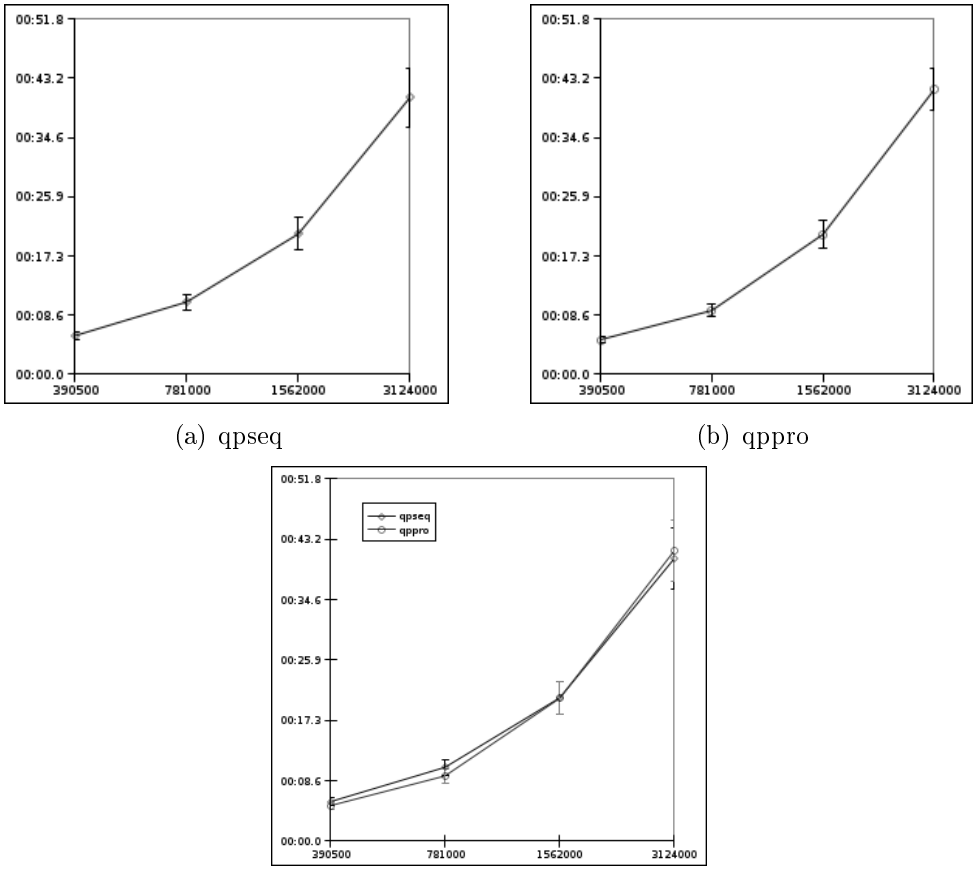
Veremos agora uma análise da eficiência da implementação paralela. Os testes a seguir foram todos realizados sobre um conjunto de 6 máquinas SGI, cada qual com 2 processadores de 4 núcleos Intel Xeon 2.00 GHz. Cada sistema conta com 4 GB de memória principal e todos encontram-se ligados através de uma rede Gigabit Ethernet. O sistema operacional é o Linux, kernel 2.6.20-xen-r6, e a máquina virtual Java é o Sun JDK 1.6.0_03. A tabela 6.6 mostra uma comparação entre os tempos de execução do `qpseq` e `qppro` para grãos variando de 390500 a 3124000 iterações em um destes processadores.

Tabela 6.6: Comparação de tempos de execução entre `qpseq` e `qppro`.

| Tamanho do grão | qpseq | | qppro | |
|---|---|---|---|---|
| | Tempo médio | $\sigma$ | Tempo médio | $\sigma$ |
| 390500 | 00:05.6 | 00:00.5 | 00:05.1 | 00:00.5 |
| 781000 | 00:10.6 | 00:01.1 | 00:09.3 | 00:00.9 |
| 1562000 | 00:20.5 | 00:02.4 | 00:20.4 | 00:02.1 |
| 3124000 | 00:40.4 | 00:04.3 | 00:41.5 | 00:03.1 |

Aqui podemos notar que os tempos de execução para estas duas implementações são muito próximos. As figuras 6.1(a) e 6.1(b) mostram estes tempos e seus respectivos desvios padrão na forma de barras verticais. A figura 6.1(c) deixa claro, ao

levarmos em conta o desvio padrão dos tempos de cada implementação, que não há diferença entre as duas para o caso em análise. Ou seja, a implementação distribuída obteve o mesmo — senão melhor em alguns casos — desempenho da versão seqüencial para a execução dos blocos de iterações escolhidos.




(a) qpseq

(b) qppro

(c) qpseq e qppro

Figura 6.1: Tempos de execução médios e desvio padrão para `qpseq` e `qppro`

As tabelas 6.7, 6.8, 6.9 e 6.10 mostram uma avaliação da escalabilidade do qppro quando utilizados 1, 8, 16, 24, 32, 40 e 48 processadores idênticos, respectivamente, para execução de um ataque ao algoritmo DES com varredura do espaço de chaves de 5 caracteres. Nestas tabelas, a coluna Tempo decorrido mostra o tempo médio real de duração de um ataque e a coluna $\Sigma T_{bloco}$ mostra a soma dos tempos gastos pelos trabalhadores para executar todas as tarefas. A coluna Aceleração mostra a taxa $\Sigma T_{bloco}$ / Tempo decorrido. Os respectivos gráficos das figuras 6.2, 6.3, 6.4 e 6.5 mostram esta informação em um formato de melhor visualização. Aqui pode ser notada a importância do tamanho do grão de processamento no desempenho final do sistema, melhor visualizada pela sobreposição destes gráficos na figura 6.6.

Tabela 6.7: Escalabilidade do qppro para o algoritmo DES. 2346 blocos de 390500 iterações. Tempo médio de processamento do grão: 5,07s.

| Trabalhadores | $\Sigma T_{bloco}$ | Tempo decorrido | Aceleração |
|---|---|---|---|
| 1 | 03:40:54.01 | 03:40:55.45 | 1 |
| 8 | 03:34:18.81 | 00:26:50.72 | 7.98 |
| 16 | 03:07:30.08 | 00:11:45.77 | 15.94 |
| 24 | 03:31:02.48 | 00:08:50.60 | 23.86 |
| 32 | 03:31:33.54 | 00:06:40.43 | 31.70 |
| 40 | 03:22:29.73 | 00:05:04.54 | 39.90 |
| 48 | 03:22:16.00 | 00:04:18.08 | 47.02 |

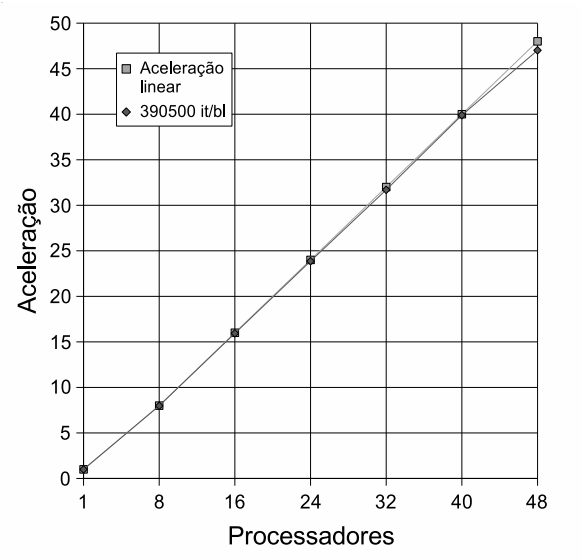



Figura 6.2: Aceleração do qppro para o algoritmo DES. Blocos de 390500 iterações.

Observando a figura 6.6 também podemos concluir que a sobrecarga causada pela porção seqüencial da implementação qppro é mínima. Esta conclusão é decorrência direta da observação anterior pois, se a sobrecarga fosse alta, esta implementação forçosamente não poderia obter uma boa escalabilidade em caso algum, de acordo com a lei de Amdahl (TANENBAUM; STEEN, 2006).

Tabela 6.8: Escalabilidade do qppro para o algoritmo DES. 1173 blocos de 781000 iterações. Tempo médio de processamento do grão: 9,32s.

| Trabalhadores | $\Sigma T_{bloco}$ | Tempo decorrido | Aceleração |
|---|---|---|---|
| 1 | 03:29:22.98 | 03:29:23.76 | 1 |
| 8 | 03:02:10.79 | 00:22:52.67 | 7.96 |
| 16 | 03:10:30.94 | 00:12:00.69 | 15.86 |
| 24 | 03:22:55.41 | 00:08:34.97 | 23.64 |
| 32 | 03:28:02.49 | 00:06:37.71 | 31.39 |
| 40 | 03:16:57.97 | 00:05:02.36 | 39.09 |
| 48 | 03:21:12.75 | 00:04:25.50 | 45.47 |

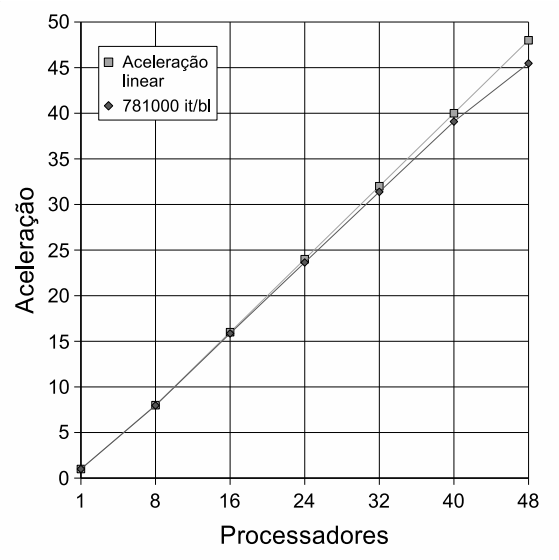



Figura 6.3: Aceleração do qppro para o algoritmo DES. Blocos de 781000 iterações.

Tabela 6.9: Escalabilidade do qppro para o algoritmo DES. 587 blocos de 1562000 iterações. Tempo médio de processamento do grão: 20,43s.

| Trabalhadores | $\Sigma T_{bloco}$ | Tempo decorrido | Aceleração |
|---|---|---|---|
| 1 | 03:21:22.51 | 03:21:23.44 | 1 |
| 8 | 03:19:49.25 | 00:25:07.29 | 7.95 |
| 16 | 03:07:52.85 | 00:11:55.75 | 15.75 |
| 24 | 03:28:22.49 | 00:08:55.49 | 23.35 |
| 32 | 03:27:44.00 | 00:06:41.49 | 31.04 |
| 40 | 03:16:43.66 | 00:05:04.30 | 38.79 |
| 48 | 03:22:23.23 | 00:04:32.12 | 44.62 |

Como demonstrado por estes testes, quanto maior o grão de processamento, maior o grau de afastamento da aceleração ideal. Também deve ser notado que este grau de afastamento, mesmo no pior dos casos aqui mostrados, não é muito acentuado, permitindo ao usuário ampla escolha do tamanho do grão. Ou seja, o usuário pode escolher o tamanho do grão baseado em alguma métrica que lhe seja

Tabela 6.10: Escalabilidade do qppro para o algoritmo DES. 294 blocos de 3124000 iterações. Tempo médio de processamento do grão: 41,55s.

| Trabalhadores | $\Sigma T_{bloco}$ | Tempo decorrido | Aceleração |
|---|---|---|---|
| 1 | 03:20:12.93 | 03:20:13.18 | 1 |
| 8 | 03:24:56.59 | 00:25:59.05 | 7.89 |
| 16 | 03:04:48.94 | 00:12:03.49 | 15.33 |
| 24 | 03:26:40.85 | 00:09:04.83 | 22.76 |
| 32 | 03:28:12.22 | 00:06:56.13 | 30.02 |
| 40 | 03:17:35.97 | 00:05:16.62 | 37.45 |
| 48 | 03:23:00.87 | 00:04:45.13 | 42.72 |

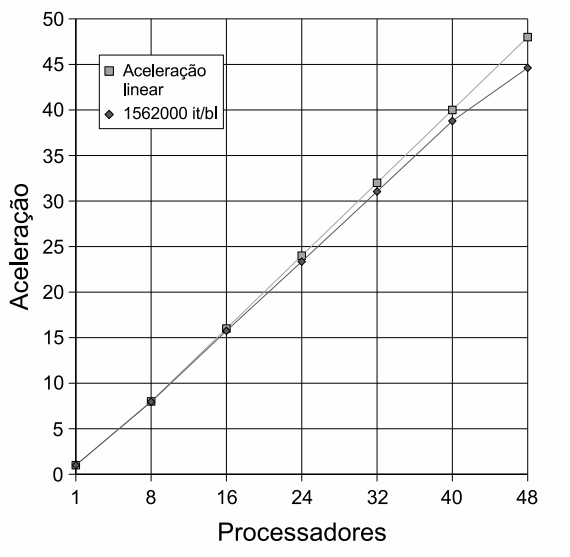



Figura 6.4: Aceleração do qppro para o algoritmo DES. Blocos de 1562000 iterações.

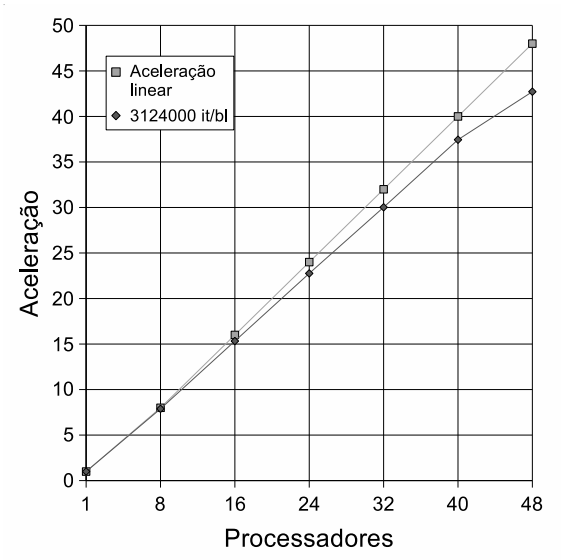



Figura 6.5: Aceleração do qppro para o algoritmo DES. Blocos de 3124000 iterações.

importante, ao invés de puramente buscar o máximo desempenho possível.

Um exemplo de tais métricas é a busca por um grão de processamento que não afete o processamento de um usuário local quando este requisitar um processador em uso pelo sistema. Os usuários podem ser desestimulados a executar o cliente local de processamento se isto se traduzir em redução significativa de seu desempenho. Um grão relativamente pequeno, de 10 segundos por exemplo, quer dizer que na média dos casos o usuário, ao necessitar realizar processamento, irá experimentar um decréscimo de desempenho por 5 segundos, que pode ser considerado aceitável. Portanto, para uma boa escolha do tamanho do grão é necessário levar

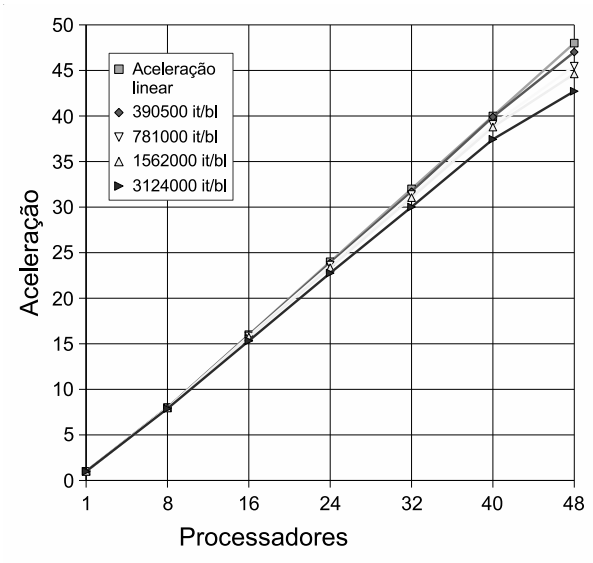



Figura 6.6: Aceleração do qppro para o algoritmo DES. Comparação geral.

em consideração a disponibilidade dos usuários que irão ceder seus processadores para o sistema quando ociosos.

Em resumo, os testes realizados demonstram que tanto o arcabouço Quebra-pedra quanto o sistema distribuído implementados alcançaram os resultados esperados para eficácia, eficiência, escalabilidade e baixa sobrecarga. Isto indica que os mesmos podem ser utilizados para a tarefa de criptoanálise em ambiente real de computação forense.

# 7 CONCLUSÃO

O uso da linguagem Java para a programação de sistemas de criptoanálise para uso em informática forense, a despeito dos problemas de desempenho da linguagem, fornece uma solução de alto-desempenho, portabilidade, extensibilidade e facilidade de uso. Conceitos introduzidos à linguagem permitem estas características, em especial o código reflexivo e a interface nativa Java. Com base nestas considerações, este trabalho mostrou a criação de um sistema distribuído de criptoanálise voltado ao uso forense. Para isto, foi criado um arcabouço, o Quebra-pedra, que oferece um conjunto de ferramentas para simplificar a criação de aplicações de criptoanálise computacional baseadas em ataques de dicionário e busca exaustiva. Além do arcabouço, foi utilizado o *middleware* ProActive como infra-estrutura de metacomputação para a implementação paralela e distribuída de forma simplificada. Testes realizados com uma implementação seqüencial que utiliza o arcabouço demonstraram que a perda de desempenho média é muito pequena em relação à uma implementação em linguagem C, quando utilizado o mecanismo de carga de código nativo da linguagem Java. Detalhes arquiteturais dos processadores e, principalmente, o algoritmo criptográfico sendo atacado tem grande influência sobre este desempenho.

Também verificou-se a eficácia do uso da computação paralela em Java para aumentar o desempenho de um sistema de criptoanálise computacional baseado no arcabouço. Testes realizados mostraram que um aumento de desempenho muito próximo ao linear pode ser conseguido para este problema, comprovando que a solução implementada para o mesmo apresenta grande escalabilidade e baixa sobrecarga.

Finalmente, os resultados dos testes realizados mostram que a metacomputação oferece uma alternativa simples, eficaz, eficiente e de baixo custo para uso em ambiente forense. E, tanto o arcabouço Quebra-pedra quanto a implementação

distribuída podem ser seguramente melhoradas, mas, em seu estado atual de desenvolvimento já apresentam condições de serem utilizadas em aplicações reais de criptoanálise forense computacional.

# REFERÊNCIAS

ANDERSON, D. P. BOINC: a system for public-resource computing and storage. In: FIFTH IEEE/ACM INTERNATIONAL WORKSHOP ON GRID COMPUTING., 2004. **Proceedings...** IEEE Computer Society, 2004. p.4–10.

ATTALI, I.; CAROMEL, D.; GUIDER, R. A step toward automatic distribution of java programs. **FMOODS 2000**, [S.l.], 2000.

BADUEL, L.; BAUDE, F.; CAROMEL, D.; CONTES, A.; HUET, F.; MOREL, M.; QUILICI, R. **Grid Computing**: software environments and tools. [S.l.]: Springer-Verlag, 2006.

BAUDE, F.; CAROMEL, D.; HUET, F.; MESTRE, L.; VAYSSIERE, J. Interactive and descriptor-based deployment of object-oriented grid applications. **The Eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC-11)**, [S.l.], p.93–102, July 2002.

BAUMGARTNER, K. M.; WAH, B. W. Computer scheduling algorithms: past, present and future. **Inf. Sci.**, [S.l.], v.57-58, p.319–345, 1991.

BERMAN, F. D.; WOLSKI, R.; FIGUEIRA, S.; SCHOPF, J.; SHAO, G. Application-level scheduling on distributed heterogeneous networks. In: SUPERCOMPUTING '96: PROCEEDINGS OF THE 1996 ACM/IEEE CONFERENCE ON SUPERCOMPUTING (CDROM), 1996. **Anais...** [S.l.: s.n.], 1996. p.39.

BESSET, D. H. **Object-Oriented Implementation of Selected Numerical Methods**: an introduction with java and smalltalk. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000.

BIHAM, E.; SHAMIR, A. **Differential Cryptanalysis of the Data Encryption Standard**. [S.l.]: Springer, 1994.

BORST, J.; PRENEEL, B.; VANDEWALLE, J. On time-memory tradeoff between exhaustive key search and table precomputation. In: SYMP. ON INFORMATION THEORY IN THE BENELUX, 19., 1998, Veldhoven (NL). **Anais...** Werkgemeenschap Informatie- en Communicatietheorie: Enschede (NL), 1998. p.111–118.

BULL, J. M.; SMITH, L. A.; POTTAGE, L.; FREEMAN, R. Benchmarking Java against C and Fortran for scientific applications. In: JGI '01: PROCEEDINGS OF THE 2001 JOINT ACM-ISCOPE CONFERENCE ON JAVA GRANDE, 2001, New York, NY, USA. **Anais...** ACM Press, 2001. p.97–105.

CALZAROSSA, M.; MASSARI, L.; TESSERA, D. Workload Characterization Issues and Methodologies. In: HARING, G.; LINDEMANN, C.; REISER, M. (Ed.). **Performance Evaluation**: origins and directions. [S.l.]: Springer-Verlag, 2000. p.459–482. Lect. Notes Comput. Sci. vol. 1769.

CAROMEL, D.; BELLONCLE, F.; ROUDIER, Y. The C++// Language. In: WILSON, G.; LU, P. (Ed.). **Parallel Programming Using C++**. Cambridge (MA), USA: MIT Press, 1996. p.257–296.

CAROMEL, D.; KLAUSER, W.; VAYSSIÈRE, J. Towards seamless computing and metacomputing in Java. **Concurrency: Practice and Experience**, [S.l.], v.10, n.11–13, p.1043–1061, 1998.

CERA, M. C.; PASIN, M. Suporte em Java para alocação dinâmica de processadores. **Anais da 4a Escola Regional de Alto Desempenho, ERAD2004**, [S.l.], p.171–172, January 2004.

CLARKE, L.; GLENDINNING, I.; HEMPEL, R. The MPI Message Passing Interface Standard. In: PROGRAMMING ENVIRONMENTS FOR MASSIVELY PARALLEL DISTRIBUTED SYSTEMS: WORKING CONFERENCE OF THE IFIP WG10.3, APRIL 25–29, 1994, ASCONA, ITALY, 1994, Boston, MA, USA. **Anais...** Birkhäuser, 1994. p.213–218.

COMER, D. E. **Internetworking with TCP/IP**. 2nd.ed. New Jersey: Prentice-Hall, 1991. v.I: Principles,Protocols and Architectures.

COPELAND, B. J. Colossus: its origins and originators. **IEEE Annals of the History of Computing**, Los Alamitos, CA, USA, v.26, n.4, p.38–45, 2004.

DAVIS, K.; HOISIE, A.; JOHNSON, G.; KERBYSON, D. J.; LANG, M.; PAKIN, S.; PETRINI, F. A Performance and Scalability Analysis of the BlueGene/L Architecture. In: SC '04: PROCEEDINGS OF THE 2004 ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2004, Washington, DC, USA. **Anais...** IEEE Computer Society, 2004. p.41.

FOSTER, I. What is the Grid? - a three point checklist. **GRIDtoday**, [S.l.], v.1, n.6, July 2002.

FOSTER, I.; KESSELMAN, C. (Ed.). **The grid**: blueprint for a new computing infrastructure. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

FOX, G. C.; WILLIAMS, R. D.; MESSINA, P. C. **Parallel Computing Works!** [S.l.]: Morgan Kaufmann Publishers Inc., 1994. xviii + 977p.

GANESAN, R.; SHERMAN, A. T. Statistical Techniques for Language Recognition: an introduction and guide for cryptanalysts. **Cryptologia**, [S.l.], v.17, n.4, p.321–366, 1993.

GENAUD, S.; RATTANAPOKA, C. P2P-MPI: a peer-to-peer framework for robust execution of message passing parallel programs on grids. **Journal of Grid Computing**, [S.l.], v.5, n.1, p.27–42, 2007. Online since Friday, December 15, 2006.

GOSLING et al., J. **The Java Language Specification, Third edition**. [S.l.]: Sun Microsystems, Inc., 2005.

GRIMSHAW, A. S.; WULF, W. A. Legion: the next logical step toward the world-wide virtual computer. **Communications of the ACM**, [S.l.], v.40, n.1, p.39–45, Jan. 1997.

HARALICK, R.; SHANMUGAM, K.; DINSTEIN, I. Textural Features for Image Classification. **IEEE International Conference on Systems, Man, and Cybernetics**, [S.l.], v.3, n.6, p.610–621, November 1973.

HUET, F.; CAROMEL, D.; BAL, H. E. A High Performance Java Middleware with a Real Application. In: SC '04: PROCEEDINGS OF THE 2004 ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2004, Washington, DC, USA. **Anais...** IEEE Computer Society, 2004. p.2.

JCE Reference Guide. Disponível em http://java.sun.com/j2se/1.5.0/docs/guide/security/jce/JCERefGuide.html.

JOSHI, R. K.; RAM, D. J. Anonymous Remote Computing: a paradigm for parallel programming on interconnected workstations. **IEEE Trans. Softw. Eng.**, [S.l.], v.25, n.1, p.75–90, 1999.

KALÉ, L. V.; KUMAR, S.; DESOUZA, J. A Malleable-Job System for Timeshared Parallel Machines. In: CCGRID '02: PROCEEDINGS OF THE 2ND IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, 2002, Washington, DC, USA. **Anais...** IEEE Computer Society, 2002. p.230.

KNUDSEN, L. R. Contemporary Block Ciphers. In: LECTURES ON DATA SECURITY, MODERN CRYPTOLOGY IN THEORY AND PRACTICE, SUMMER SCHOOL, AARHUS, DENMARK, JULY 1998, 1999, London, UK. **Anais...** Springer-Verlag, 1999. p.105–126.

KORPELA, E. et al. SETI@home: massively distributed computing for SETI. **CISE Computing in Science and Engineering**, [S.l.], p.78–83, January 2001.

KUMAR, S.; PAAR, C.; PELZL, J.; PFEIFFER, G.; SCHIMMLER, M. COPACOBANA A Cost-Optimized Special-Purpose Hardware for Code-Breaking. In: FCCM '06: PROCEEDINGS OF THE 14TH ANNUAL IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES (FCCM'06), 2006, Washington, DC, USA. **Anais...** IEEE Computer Society, 2006. p.311–312.

LEE, C.; MATSUOKA, S.; TALIA, D.; SUSSMANN, A.; MULLER, M.; ALLEN, G.; SALTZ, J. **A Grid programming primer**. 2001.

LENSTRA, A. K.; VERHEUL, E. R. Selecting Cryptographic Key Sizes. **Journal of**

**Cryptology: the journal of the International Association for Cryptologic Research**, [S.l.], v.14, n.4, p.255–293, 2001.

LENSTRA, A.; SHAMIR, A.; TOMLINSON, J.; TROMER, E. **Analysis of Bernstein 's factorization circuit**. 2002.

MAINWARING, A.; CULLER, D. **Active Message Applications Programming Interface and Communication Subsystem Organization**. [S.l.]: University of California, Berkeley, 1996. Technical Report. (CSD-96-918).

MATSUI, M. **Linear Cryptanalysis Method for DES Cipher**. [S.l.: s.n.], 1994. 386+p.

MONTORO, M. **Cain & Abel**. Disponível em http://www.oxid.it/cain.html.

MOREIRA, J. E.; MIDKIFF, S. P.; GUPTA, M.; ARTIGAS, P. V.; SNIR, M.; LAWRENCE, R. D. Java programming for high-performance numerical computing. **IBM Systems Journal**, [S.l.], v.39, n.1, p.21–, 2000.

MUFFET, A. **Crack**: a sensible password checker for Unix. Disponível em ftp://ftp.cert.dfn.de/pub/tools/password/Crack/.

NAOR, M.; WIEDER, U. Novel architectures for P2P applications: the continuous-discrete approach. In: SPAA '03: PROCEEDINGS OF THE FIFTEENTH ANNUAL ACM SYMPOSIUM ON PARALLEL ALGORITHMS AND ARCHITECTURES, 2003, New York, NY, USA. **Anais...** ACM Press, 2003. p.50–59.

OECHSLIN, P. Making a Faster Cryptanalytic Time-Memory Trade-Off. In: CRYPTO'03, 2003. **Proceedings...** [S.l.: s.n.], 2003.

QUISQUATER, J.-J.; DESMEDT, Y. G. Chinese Lotto as an Exhaustive Code-Breaking Machine. **Computer**, Los Alamitos, CA, USA, v.24, n.11, p.14–22, 1991.

SANTOS, L. P. **Application Level Runtime Load Management**: a bayesian approach. 2001. PhD Dissertation — University of Minho, Braga, Portugal.

SANTOS, L. P.; PROENÇA, A. J. A Bayesian RunTime Load Manager on a Shared Cluster. In: CCGRID, 2001. **Anais...** IEEE Computer Society, 2001. p.674–679.

SCHNEIER, B. **Applied cryptography (2nd ed.)**: protocols, algorithms, and source code in c. [S.l.]: John Wiley & Sons, Inc., 1995.

SCHOPF, J. M.; BERMAN, F. Stochastic scheduling. In: SUPERCOMPUTING '99: PROCEEDINGS OF THE 1999 ACM/IEEE CONFERENCE ON SUPERCOMPUTING (CDROM), 1999, New York, NY, USA. **Anais...** ACM Press, 1999. p.48.

SHAMIR, A. Analysis and optimization of the TWINKLE factoring Device. **Advances in Cryptology: EUROCRYPT 2000, lecture Notes in computer Science**, [S.l.], p.35–52, 2000.

SHAMIR, A.; TROMER, E. Factoring Large Numbers with the TWIRL Device. **Lecture Notes in Computer Science**, [S.l.], v.2729, p.1–26, 2003.

SHUANGLEI, Z. **RainbowCrack**. Disponível em http://www.antsight.com/zsl/rainbowcrack/.

SOLAR DESIGNER. **John The Ripper, JtR**. Disponível em http://www.openwall.com/john/.

STEEN, M. van; HOMBURG, P.; TANENBAUM, A. S. **The Architectural Design of Globe**: A wide-area distributed system. Netherlands: [s.n.], 1997. (IR-422).

STERLING, T.; BECKER, D.; SAVARESE, D.; DORBAND, J.; RANAWAKE, U.; PACKER, C. Beowulf: a parallel workstation for scientific computation. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING (ICPP), 1995., 1995. **Proceedings...** [S.l.: s.n.], 1995. v.1, p.11–14.

SUNAGA, H.; UEDA, K.; IWATA, T.; KIKUMA, K.; TAKEMOTO, M. P2P Applications Using the Semantic Information Oriented Network. **p2p**, Los Alamitos, CA, USA, v.00, p.272–273, 2004.

TANENBAUM, A. S.; STEEN, M. van. **Distributed Systems**: principles and paradigms (2nd edition). Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

Team distributed.net. **distributed.net**. 2004.

THAIN, D.; TANNENBAUM, T.; LIVNY, M. Distributed computing in practice: the condor experience. **Concurrency - Practice and Experience**, [S.l.], v.17, n.2-4, p.323–356, 2005.

WALKER, R.; JACKWAY, P.; LONGSTAFF, I. **Improving co-occurrence matrix feature discrimination**. 1995.

WIENER, M. J. Efficient DES Key Search, Technical Report TR-244, Carleton University. In: **William Stallings, Practical Cryptography for Data Internetworks**. [S.l.]: IEEE Computer Society Press, 1996.

# APÊNDICE A — ALTERNATIVAS EM PROCESSAMENTO PARALELO DE BAIXO CUSTO

Nesta seção são analisados alguns sistemas paralelos em utilização ou desenvolvimento atualmente. Devido ao escopo deste trabalho, não serão analisados sistemas especialmente construídos, normalmente de alto custo, mas apenas aqueles que funcionam sobre redes pré-existentes sem a necessidade de grande alteração das mesmas.

## A.1 Redes de Computadores, sockets, MPI, RMI, ...

Após o advento das redes de computadores mas antes do surgimento de ambientes voltados especialmente ao desenvolvimento de sistemas distribuídos, estes sistemas eram, e ainda são em muitos casos, feitos utilizando mecanismos simples de comunicação por troca de mensagens. Neste caso, algum meio de envio e recebimento de mensagens completas é utilizado para realizar a comunicação de dados entre dois ou mais computadores. Exemplos destes mecanismos são os soquetes TCP/IP (*sockets*) (COMER, 1991) e a *Message Passing Interface* (MPI) (CLARKE; GLENDINNING; HEMPEL, 1994), dentre outros.

Em geral, um ponto comum a todos estes métodos é que o programador do sistema distribuído deve criar um protocolo de comunicação e controle e implementá-lo totalmente. Ou seja, deve haver sempre o controle da troca de dados em um nível mais baixo, forçando o programador a ter controle absoluto sobre que processador irá executar cada uma das tarefas do sistema. A vantagem disto é a capacidade de ajustes finos de otimização, porém, as desvantagens trazidas pelo grau de controle necessário, em geral, contrapõem-se fortemente à sua utilização em ambientes com

muitos processadores.

## A.2 Aglomerados de Processadores Produzidos em Larga Escala

A popularização das redes de computadores levou ao surgimento de tecnologias que se utilizam da capacidade de comunicação destas para diversos fins. Dentre estas tecnologias os aglomerados (*clusters*) surgem como uma das mais promissoras, devido às suas características de flexibilidade e baixo custo.

Um aglomerado pode ser definido como sendo um conjunto de nós interligados por uma rede, preferencialmente de alta velocidade, e com o devido suporte de sistema para que funcione como se fosse uma única máquina. Os nós de um aglomerado podem ser desde computadores pessoais comuns até sistemas reduzidos, formados apenas por processadores e suas memórias locais ou mesmo sistemas especialmente construídos para este fim, mas que usam processadores produzidos em larga escala e sistemas operacionais populares como elementos básicos de processamento. Trata-se, portanto, de um sistema multicomputador que, devido às suas características, normalmente alcança níveis de custo bem abaixo de similares multiprocessados.

Atualmente os aglomerados são a mais popular e variada maneira de dar suporte ao processamento paralelo, gerando diversas pesquisas sobre seu funcionamento por inúmeros grupos. Um exemplo bastante popular são os sistemas montados a partir do projeto Beowulf (STERLING et al., 1995). Este projeto foi desenvolvido pela Agência Espacial Norte-Americana (NASA) para realizar simulações que necessitam de processamento intensivo. Várias de suas idéias e metodologias são hoje a base de quase todos os aglomerados.

O uso de aglomerados para o processamento paralelo oferece importantes vantagens:

**utilização de equipamentos existentes/de prateleira** cada uma das máquinas em um aglomerado pode ser um sistema completo, utilizável para qualquer tarefa de processamento de dados, e não apenas para a execução de aplicativos paralelos. Isto levou muitos pesquisadores a projetar sistemas que possibilitem a criação de aglomerados que utilizem os ciclos ociosos de estações ligadas à uma rede mas que em alguns momentos não estão realizando processamento

útil para seus usuários particulares. Embora esta não seja uma tarefa simples, pode ser feita. Exemplos deste tipo de sistema incluem o projeto CADEO, desenvolvido por pesquisadores da UFSM (CERA; PASIN, 2004), o SETI@Home (KORPELA et al., 2001) e o distributed.net (Team, 2004).

**baixo custo** com a atual ubiqüidade de sistemas interconectados por redes de computadores a maioria do *hardware* para contruir aglomerados pode ser obtida a um relativo baixo custo. Em comparação, sistemas multiprocessadores, por exemplo, possuem mercados menores e, portanto, tendem a ter preços maiores.

**escalabilidade** aglomerados possuem boa escalabilidade, podendo chegar a tamanhos bastante grandes (sistemas com milhares de nós de processamento não são incomuns). Sistemas multiprocessados normalmente não possuem tal característica. Existem até mesmo pesquisas que visam disponibilizar o poder de processamento da Internet para uso como um enorme aglomerado planetário. Exemplos destes sistemas são Legion, Condor e Globe (GRIMSHAW; WULF, 1997; THAIN; TANNENBAUM; LIVNY, 2005; STEEN; HOMBURG; TANENBAUM, 1997).

**disponibilidade** pela natureza das estruturas de redes utilizadas para sua interconexão, os aglomerados possuem alta disponibilidade. A troca de nós falhos do sistema é trivial quando comparada à mesma tarefa em sistemas multiprocessadores. Isto se torna importante não somente para aplicativos que possuam tolerância à falhas mas, também, para que sistemas que possuem alta probabilidade de falhas simplesmente por conterem muitos elementos possam continuar funcionando a contento.

Embora os aglomerados possuam estas vantagens, eles também possuem seus problemas. Abaixo veremos algumas das desvantagens associadas aos mesmos:

- com poucas exceções, o *hardware* de rede atual não é projetado para o processamento paralelo. Sua latência típica é muito alta e sua vazão é baixa quando comparados com sistemas com SMP por exemplo - que possuem memória compartilhada. Embora existam estruturas de rede com estas características, nenhuma delas é de longo alcance e todas possuem alto custo.

- se a rede do aglomerado não é isolada de outros tráfegos, caso comum quando se usa uma rede de propósito geral para o processamento paralelo, o seu desempenho pode ser substancialmente deteriorado.

- pouco suporte dos sistemas existentes para tratar um aglomerado como um único sistema. Por exemplo, é difícil até mesmo encontrar utilitários de sistema que reportem a execução de processos sobre um dado aglomerado. Além disso, os sistemas existentes tendem a possuir grande complexidade de instalação e manutenção.

Devido a alguns dos problemas associados aos aglomerados e, principalmente, pela ascenção das tecnologias de interligação de longa distância e da popularização da Internet, novas alternativas ganharam destaque na área de processamento distribuído: a computação ponto-a-ponto (*peer-to-peer*) e a computação em grade ou metacomputação — tratada na seção 2.2.1.

## A.3 Ponto-a-ponto (*Peer-to-peer*)

Mais conhecida pelo acrônimo p2p, trata-se de uma arquitetura em que não há nodos ou máquinas especiais que disponibilizam um determinado serviço ou gerenciam os recursos disponíveis. Resumindo, não há centralização de serviços. Ao invés disso, as responsabilidades são uniformemente divididas entre todas as máquinas do sistema, conhecidas como pares ou pontos (*peers*). Estas máquinas podem ser, simultaneamente, tanto clientes quanto servidoras, não havendo, portanto, a hierarquia dos sistemas tradicionais mestre-escravo/cliente-servidor (TANENBAUM; STEEN, 2006). Esta abordagem surgiu como uma evolução natural do conceito de compartilhamento de arquivos em redes ponto-a-ponto, popularizado por aplicações como o Napster, e atualmente encontra-se em fase de pesquisa e desenvolvimento de implementações que a possibilitem.

Um exemplo de sistema em desenvolvimento que utiliza esta abordagem é o *middleware* P2P-MPI (GENAUD; RATTANAPOKA, 2007). Este sistema busca oferecer características da computação p2p — auto-configuração, gerenciamento de dados, tolerância a falhas e capacidade de computação abstrata — sobre o modelo padrão MPI de processamento paralelo. Como este padrão é um dos mais utilizados para este tipo de processamento, o P2P-MPI busca, portanto, simplificar a

transformação de sistemas paralelos convencionais para execução em ambiente de computação p2p.

Por ser uma área ainda em desenvolvimento, não há estudos que demonstrem se este tipo de processamento é melhor ou pior do que os tipos atualmente utilizados. Além disso, por utilizar um novo paradigma em que não há centralização, é necessário adaptar as soluções tradicionalmente utilizadas em sistemas centralizados e/ou hierárquicos para esta nova arquitetura, ou criar novas soluções quando tal adaptação não tiver bons resultados (NAOR; WIEDER, 2003; SUNAGA et al., 2004).

# APÊNDICE B — QUEBRA-PEDRA: REFERÊNCIA RÁPIDA

Quebra-Pedra (qp) é uma plataforma para Criptoanálise baseada em varredura de espaço de soluções. Este sistema é capaz de empregar técnicas conhecidas, como o ataque por força-bruta ou a busca dirigida por dicionário. Na verdade, mais que um sistema, esta plataforma é um arcabouço (framework) que oferece ao criptoanalista a capacidade de extender o seu uso a qualquer problema de criptoanálise que possa ser atacado pelas técnicas oferecidas.

## B.1 Características principais

Atualmente, a plataforma conta com uma implementação sequencial (*stand-alone*) e uma versão distribuída capazes de realizar os ataques. Vejamos agora os tipos de ataque que a plataforma oferece e suas principais características.

### B.1.1 Ataque por Varredura Completa ou Força-Bruta

Este tipo de ataque tem como característica principal a garantia de encontrar a chave criptográfica procurada, pois baseia-se em uma varredura total do espaço de chaves possíveis.

### B.1.2 Ataque Dirigido por Dicionário

Nesta modalidade de ataque o criptoanalista utiliza-se de um dicionário contendo as sequências básicas de caracteres a serem testados e um conjunto de regras de modificação que atuam sobre o dicionário básico, extendendo-o para englobar uma porção maior do espaço de busca total. Este tipo de ataque baseia-se na premissa de que uma grande porcentagem dos usuários escolhe suas chaves criptográficas dentre palavras presentes em seu vocabulário, ou modificações destas. Como um dicionário

é na verdade um subconjunto do espaço de buscas total, este direcionamento resulta em um número reduzido de possibilidades quando comparado à busca por varredura completa, o que se traduz em uma diminuição proporcional no tempo despendido pelo ataque.

#### *B.1.2.1 Regras de modificação*

Abaixo encontra-se uma descrição do mecanismo de modificação, bem como as regras reconhecidas pelo mesmo para a geração do espaço de busca dirigida.

O sistema conta com um arquivo de regras "rules.qp". Neste arquivo devem ser escritas todas as regras de modificação de dicionário a serem utilizadas. Tais regras encontram-se abaixo descritas.

| | |
|---|---|
| : | Não realiza mudanças na palavra. Útil como primeira regra ou para demarcar caracteres invisíveis. Ex: “s_ :” substitui o caractere “_” por um espaço em branco. |
| c | Capitalização. Coloca o primeiro caracter da palavra em caixa alta e os demais em caixa baixa. Ex: “abc” torna-se “Abc”. |
| C | Capitalização invertida. Coloca o primeiro caracter da palavra em caixa baixa e os restantes em caixa alta. Ex: “abc” torna-se “aBC”. |
| d | Duplicação. Coloca uma cópia da palavra ao fim da mesma. Ex: “abc” torna-se “abcabc”. |
| f | Reflexão. Coloca uma cópia da palavra invertida ao fim da mesma. Ex: “abc” torna-se “abccba”. |
| iNX | Inserção. Insere o caracter X na posição N da palavra. N deve ser um número de um digito em base hexadecimal. |
| l | Coloca a palavra toda em caixa baixa. |
| oNX | Substitui o caracter na posição N por X. N deve ser um número de um digito em base hexadecimal. |
| r | Reversão. Inverte a palavra. |
| sXY | Substituição. Substitui todas as instâncias do caracter X pelo caracter Y. |
| u | Coloca a palavra toda em caixa alta. |
| $X | Concatena o caracter X à palavra. |
| ^X | Coloca o caracter X antes da palavra. |
| [ | Elimina o primeiro caracter da palavra. |
| ] | Elimina o último caracter da palavra. |
| @X | Elimina todas as instâncias do caracter X da palavra. |
| ´N | Trunca a palavra na posição N. |

Cada linha do arquivo “rules.qp” será aplicada de uma só vez a cada palavra do dicionário. As regras descritas acima podem ser usadas sozinhas ou concomitantemente, sendo, neste caso, aplicadas sequencialmente uma após a outra, seguindo

a precedência da esquerda para a direita. Por exemplo, uma linha do arquivo de regras pode conter a regra ":" e outra linha pode conter uma regra tão intrincada quanto "lsa4@eo0+ ´9[$.".

## B.2 Versão *stand-alone* - qpseq

As aplicações *stand-alone* que acompanham a distribuição servem para a realização de ataques reais ou como exemplo de uso do arcabouço. Estão disponíveis exemplos de cada possível variação de ataque do arcabouço. São estas variações atualmente: ataque por varredura exaustiva do espaço de busca, varredura exaustiva por sub-espaço de busca, ataque por dicionário completo e por sub-espaço de dicionário.

## B.3 Versão distribuída - qppro

A distribuição atual do arcabouço Quebra-Pedra conta também com uma aplicação exemplo paralela, denominada qppro. Esta aplicação utiliza o ProActive, parte do projeto ObjectWeb, como infra-estrutura de metacomputação. ProActive é um *middleware* de código aberto para computação *multi-threaded*, paralela e distribuída. Maiores informações sobre este *middleware* podem ser obtidas em `http://www.objectweb.org/proactive`.

A configuração dos nós que serão utilizados pelo qppro é bastante simples, sendo feita inteiramente em um arquivo XML (`qppro.xml`). São fornecidos exemplos comentados deste arquivo para referência.

Qppro conta, também, com uma interface gráfica de configuração, monitoramento e controle da execução de ataques.