

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

**CENTRO DE CIÊNCIAS EXATAS, AMBIENTAIS E DE
TECNOLOGIAS**

FACULDADE DE ENGENHARIA DE COMPUTAÇÃO

NICHOLAS BASTOS FERREIRA FANELLI

**GERAÇÃO DE CHAVES CRIPTOGRÁFICAS
USANDO UM ALGORITMO BIO-INSPIRADO**

**CAMPINAS
2014**

NICHOLAS BASTOS FERREIRA FANELLI

**GERAÇÃO DE CHAVES CRIPTOGRÁFICAS
USANDO UM ALGORITMO BIO-INSPIRADO**

Monografia de Trabalho de Conclusão de Curso apresentada como exigência da disciplina Projeto Final II, ministrada no Curso de Engenharia de Computação, do Centro de Ciências Exatas, Ambientais e de Tecnologias da Pontifícia Universidade Católica de Campinas.

Orientador: Prof. Dr. Carlos Miguel Tobar Toledo

Coorientador: Micael Cabrera Carvalho

**PUC-CAMPINAS
2014**

**Pontifícia Universidade Católica de Campinas
Centro de Ciências Exatas, Ambientais e de Tecnologias
Faculdade de Engenharia de Computação**

**FANELLI, Nicholas Bastos Ferreira
Geração de Chaves Criptográficas Usando Um Algoritmo Bio-Inspirado**

**Monografia de Trabalho de Conclusão de Curso
Graduação em Engenharia de Computação**

BANCA EXAMINADORA

Presidente e Orientador Prof. Dr. Carlos Miguel Tobar Toledo

1º Examinador Prof.(a). Dr.(a). <Nome do Segundo Professor da Banca>

Campinas, <dd> de <mês> de 2014.

Aos meus queridos avós maternos, Luiz e Lylah (*in memoriam*), que sempre me amaram e torceram pelo meu sucesso.

AGRADECIMENTOS

À minha mãe, Jael Cristina, pelo ininterrupto esforço e dedicação para com a minha educação e formação, pelo fundamental apoio emocional em todos os momentos de dificuldade, por compreender meus momentos de ausência e por incontáveis outros fatores que sofreram influência dela e me deram a base para ser quem sou hoje.

Ao meu irmão, Gregory, que apesar de muitas vezes longe fisicamente, sempre foi e continua sendo um grande amigo e me motivou constantemente durante o desenvolvimento do Trabalho de Conclusão de Curso.

Ao meu padrasto, Carlindo, por ter influenciado a minha escolha de profissão rumo à engenharia e ter contribuído orientando intelectualmente minha formação.

Ao meu professor e orientador, Prof. Dr. Carlos Miguel Tobar Toledo, pelas valiosas orientações e inúmeros desafios propostos a mim durante o desenvolvimento do Trabalho de Conclusão de Curso, que me fizeram crescer, evoluir e, acima de tudo, me permitiram enxergar que estou realmente pronto para exercer a profissão de Engenheiro de Computação.

Ao meu amigo e coorientador, Micael Cabrera Carvalho, pelos conselhos, propostas, dicas e impagáveis horas dedicadas a mim no auxílio do desenvolvimento do Trabalho de Conclusão de Curso (TCC).

À minha querida amiga Isabella Allgauer, pelas valiosas contribuições na geração de diagramas usados no TCC e pelas incomparáveis demonstrações de gentileza e paciência para tratar de alterações em versões dos mesmos.

Ao meu amigo Paulo Vitor Merlin, que mesmo não tendo obrigação alguma, se dispôs a me ajudar em tudo que pudesse, oferecendo seus conhecimentos e experiência. A todos os meus colegas de turma nas disciplinas de Projeto Final I e Projeto Final II, que foram essenciais na troca de experiências e soluções para problemas comuns.

Aos meus mentores no CPqD, Alexandre Braga e Leonardo Mariote, que foram extremamente compreensivos e apoiadores durante o período de desenvolvimento do TCC.

"There are no shortcuts to any place worth going."

Beverly Sills
(1929-2007)

RESUMO

FANELLI, Nicholas Bastos Ferreira. *Geração de Chaves Criptográficas Usando Um Algoritmo Bio-Inspirado*. 2014. 67p. Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) – Pontifícia Universidade Católica de Campinas, Centro de Ciências Exatas, Ambientais e de Tecnologias, Faculdade de Engenharia de Computação, Campinas, 2014.

Nesta monografia relata-se um Trabalho de Conclusão de Curso (TCC), em que foi desenvolvido um artefato de *software* com o objetivo de se gerar chaves criptográficas robustas, através do uso de um Algoritmo Genético (AG), para aplicação em um modelo de criptografia de chave pública. Para o desenvolvimento do TCC adotou-se o método Scrum. Foram utilizadas as ferramentas rsync e Crontab para a aplicação de uma política de *backup*, enquanto que para o versionamento de código foi usado a ferramenta Git. O artefato de *software* foi desenvolvido na linguagem de programação Java e foi batizado GenCryptoKey. As chaves criptográficas por ele geradas foram avaliadas usando-se os testes *Runs*, *Serial* e *Linear Complexity*, providos pela suíte de testes do *National Instruments and Standards Institute* (NIST), dos Estados Unidos. Os resultados dos testes de avaliação comprovaram que a solução desenvolvida permitiu alcançar o objetivo proposto no TCC.

Palavras chave: Chaves criptográficas. Algoritmo Genético. Criptografia de chave pública.

ABSTRACT

FANELLI, Nicholas Bastos Ferreira. *Cryptographic Keys Generation Using A Bio-Inspired Algorithm*. 2014. 67p. *Capstone Project (Computer Engineering Undergraduate)* – Pontifical Catholic University of Campinas, Center for Exact, Environmental and Technology Sciences, School of Computer Engineering, Campinas, 2014.

In this monograph a Course Completion Work (CCW) is reported, in which a software artifact was developed with the goal of generating robust cryptographic keys through the use of a Genetic Algorithm (GA), for application in a public key cryptography model. The Scrum method was adopted for the development of this CCW. The rsync and Crontab tools were used for the application of a backup policy, while the Git tool was used for the code versioning. The software artifact was developed in the Java programming language and was called GenCryptoKey. The cryptographic keys it generated were evaluated using the Runs, Serial and Linear Complexity tests, provided by test suite of The United States' National Instruments and Standards Institute (NIST). The results of the evaluation tests showed that the developed solution allowed the achievement of the goal proposed in the CCW.

Descriptors: *Cryptographic keys. Genetic Algorithm. Public key cryptography.*

LISTA DE FIGURAS

Figura 1. Funcionamento do método Scrum	19
Figura 2. Modelo genérico do padrão arquitetural MVC	26
Figura 3. Diagrama de arquitetura do artefato de software	27
Figura 4. Diagrama de sequência do Algoritmo Genético	29
Figura 5. Diagrama de atividade do Algoritmo Genético	31
Figura 6. Tela inicial da aplicação GenCryptoKey	35
Figura 7. Tela de cadastro do usuário	36
Figura 8. Cadastro do usuário com exemplos do filtro validador	37
Figura 9. Tela de configuração dos parâmetros para o Algoritmo Genético	38
Figura 10. Tela de visualização de chaves armazenadas no banco de dados	39
Figura 11. Exemplo de chave exportada para arquivo	40
Figura 12. Pseudocódigo da rotina <i>crossover</i>	51
Figura 13. Tempo de execução do AG em função do tamanho da chave	51

LISTA DE TABELAS

Tabela 1. Parâmetros do AG para testes de avaliação	47
Tabela 2. Resultados consolidados da avaliação do TCC	48

LISTA DE QUADROS

Quadro 1. Comparação entre aspectos do TCC e outros trabalhos publicados	16
Quadro 2. Ferramentas para elaboração de diagramas	23
Quadro 3. Possíveis erros na conclusão de um teste estatístico	43

LISTA DE ABREVIATURAS E SIGLAS

AE	= Algoritmos Evolutivos
AG	= Algoritmo Genético
API	= <i>Application Programming Interface</i>
ASN.1	= <i>Abstract Syntax Notation One</i>
IDE	= <i>Integrated Development Environment</i>
LFSR	= <i>Linear Feedback Shift-Register</i>
PEM	= <i>Privacy Enhanced Email</i>
PRNG	= <i>Pseudorandom Number Generator</i>
RFC	= <i>Request For Comment</i>
RNG	= <i>Random Number Generator</i>
RSA	= Rivest-Shamir-Adleman
SDK	= <i>Software Development Kit</i>
TCC	= Trabalho de Conclusão de Curso

SUMÁRIO

1 INTRODUÇÃO	14
1.1 Problemas e Objetivo	15
1.2 Proposta de Artefato	16
1.3 Estado da Arte	16
1.4 Organização da Monografia	17
2 MÉTODO DE DESENVOLVIMENTO E ASPECTOS DE INOVAÇÃO E APRIMORAMENTO	18
2.1 Método de Desenvolvimento	18
2.2 <i>Backup</i>	19
2.3 Versionamento de Código	20
2.4 Algoritmo Genético	20
2.5 Criptografia Rivest-Shamir-Adleman	20
2.6 Testes Estatísticos	21
3 DESENVOLVIMENTO	22
3.1 Ferramentas Utilizadas	22
3.1.1 Linguagem de Programação e Ambiente de Desenvolvimento	22
3.1.2 Elaboração dos Diagramas	23
3.1.3 Controle de Versões e Revisões de Código	23
3.1.4 Política de <i>Backup</i>	24
3.2 Arquitetura	25
3.3 Geração de Chaves	28
3.3.1 RSA	28
3.3.2 Algoritmo Genético	28
3.3.3 Cálculo de <i>Fitness</i>	32
4 FUNCIONALIDADES DO ARTEFATO	35
4.1 Apresentação e <i>Help</i>	35
4.2 Cadastro de Usuário e <i>Login</i>	36
4.3 Configuração de Parâmetros	38
4.4 Visualização de Chaves Geradas	39
4.5 Exportação	39
5 AVALIAÇÃO E RESULTADOS	41
5.1 Avaliação	41
5.2 Testes Estatísticos	41
5.2.1 Hipótese Nula e Hipótese Alternativa	42
5.2.2 Nível de Significância	43
5.2.3 <i>P-value</i>	44
5.2.4 Valor Crítico	44
5.3 Suíte de Testes	44
5.4 Testes Escolhidos	46
5.5 Análise dos Resultados	47
6 CONCLUSÃO	49
6.1 Grau de Complexidade	50
6.2 Dificuldades Encontradas	52

6.2.1 <i>Backup</i> e Versionamento de Código	52
6.2.2 Entendimento e Implementação do Algoritmo Genético	52
6.2.3 Testes Estatísticos	53
6.2.4 Integração com a Suíte de Testes	53
6.3 Qualidade de <i>Software</i>	54
6.4 Propostas para Futuras Melhorias	56
REFERÊNCIAS	58
APÊNDICES	61
Apêndice A Diagrama de sequência descrito em texto para PlantUML	61
Apêndice B Diagrama de atividade descrito em texto para PlantUML	62
Apêndice C Ferramentas para tarefas de <i>Backup</i>	63
Apêndice D <i>Bash script</i> para <i>backup</i> com o <i>rsync</i>	64
Apêndice E Entradas no Crontab do sistema	65
Apêndice F Relatórios de resultados finais de avaliação	66
ANEXOS	67
Anexo A Tabela de valores críticos do teste Kolmogorov-Smirnov	67
Anexo B Tabela de valores críticos do teste Qui-quadrado de Pearson	68

1 INTRODUÇÃO

Atualmente, chaves criptográficas são usadas para cifrar dados sensíveis quando da necessidade de comunicação entre pares em um ambiente que envolve terceiros. Um dos modelos mais utilizados para promover segurança nessa comunicação é a Criptografia de Chave Pública (mais conhecido como *Public Key Cryptography*, em inglês; ou pela sigla: PKC), que consiste em uma chave pública para cifrar e uma chave privada para decifrar. No entanto, diferentes chaves criptográficas podem ser classificadas de acordo com sua robustez, característica que promove maior qualidade de segurança ao dado que deve ser cifrado.

1.1 Problemas e Objetivos

Sabe-se hoje que a segurança de dados por criptografia é ameaçada pelo poder de processamento das máquinas (BENNET et al., 1996). Em outras palavras, através da técnica de força bruta, basta dar tempo a uma máquina e a vasta maioria das cifras é passível de ser quebrada – a cifragem por *One-time padding*, como proposta e patenteada por Gilbert Vernam em 1919, é talvez o único método criptográfico que provê impossibilidade prática de quebra (SCHNEIER, 1996). Felizmente, no cenário atual, as soluções utilizadas comercialmente ainda são razoavelmente aceitáveis, pois preveem que o tempo necessário para quebrar suas cifras é praticamente inviável para qualquer propósito prático de um agente mal intencionado.

Pode-se dizer que uma chave criptográfica é robusta quando tem uma grande quantidade de caracteres (usualmente entre 1024 e 4096 *bits*), curto prazo de validade – é trocada com alta periodicidade – e está suficientemente distante de outras chaves.

Força bruta, no contexto de chaves criptográficas, é um tipo de ataque baseado em um algoritmo determinístico trivial, que consiste em definir todos os possíveis candidatos de uma solução – no caso, uma determinada chave criptográfica – e verificar se ao menos um satisfaz o problema sendo atacado. Este tipo de algoritmo sempre encontrará uma solução, se ela existir. Entretanto,

seu custo computacional é proporcional ao número de candidatos à solução do problema. Uma chave que tomaria um tempo da ordem de décadas, ou mais, para ser descoberta é considerada computacionalmente segura. (PAAR; PELZL, 2011)

Sabendo-se disso, uma das alternativas para tornar uma chave mais segura a esse tipo de ataque é fazê-la suficientemente grande ao ponto que o tempo necessário para encontrá-la por força bruta seja impraticável. No entanto, isso não é suficiente para uma segurança eficaz, pois um ataque amplamente distribuído poderia ser capaz de atravessar todo o espaço de candidatos até encontrar a solução em um tempo aceitável.

Pela dificuldade de se prover segurança a dados por criptografia, como explicado anteriormente, é possível perceber que produzir uma chave criptográfica robusta pode não ser uma tarefa tão simples.

Portanto, o Trabalho de Conclusão de Curso relatado nessa monografia teve como objetivo o desenvolvimento de um artefato de *software* para a geração de chaves criptográficas robustas.

1.2 Proposta de Artefato

O artefato desenvolvido, denominado GenCryptoKey, consiste de uma aplicação *desktop* capaz de gerar um par de chaves criptográficas a partir de um Algoritmo Genético (AG), um tipo de algoritmo de busca heurística adaptativa baseado no funcionamento da genética e seleção natural, pertencente à classe de Algoritmos Evolutivos (AEs) (MISHRA, 2013). Esse tipo de algoritmo é usado para encontrar soluções para problemas de otimização usando-se mecanismos baseados na evolução biológica, tais como mutação, *crossover* (cruzamento entre cromossomos), seleção e herança. (MITCHELL, 1999)

O artefato possui também um módulo de configuração, através do qual o usuário é capaz de definir parâmetros essenciais para a execução do AG, a exemplo do tamanho das chaves, da taxa de mutação, do número de gerações a serem processadas, entre outros.

Outra característica fundamental e possivelmente a mais importante do artefato é o cálculo do valor de *fitness* dos indivíduos da população que é evoluída no AG, o qual serve o propósito de selecionar, em cada geração do algoritmo, as chaves produzidas que atendem ao requisito de robustez procurado.

1.3 Estado da Arte

A geração de chaves criptográficas através de um AG é uma proposta recente na área de Tecnologia da Informação, tendo sido alguns dos principais estudos publicados apenas em 2013. Por essa razão, até onde se sabe, atualmente não existe aplicação comercial alguma que faça uso dessa técnica. Entretanto, é uma técnica que tem sido bastante estudada e aos poucos está ganhando espaço e reconhecimento.

O Quadro 1 faz uma comparação entre as aplicações descritas nos principais estudos sobre o tema e o artefato resultante do TCC, levando em consideração os aspectos julgados mais relevantes para a qualidade final das chaves geradas.

Quadro 1. Comparação entre aspectos do TCC e outros trabalhos publicados

Aspectos	App 1	App 2	App 3	GCK
Algoritmos utilizados	AG	AG + <i>Artificial Neural Network</i>	AG + <i>Particle Swarm Optimization</i>	AG
Parametrização	Fixa	Fixa	Fixa	Configurável
Tamanho de chave	192 <i>bits</i>	192 <i>bits</i>	192 <i>bits</i>	Configurável
Cálculo do <i>Fitness</i>	Coeficiente de Autocorrelação, <i>Frequency Test</i> e <i>Gap Test</i>	Coeficiente de Autocorrelação, <i>Frequency Test</i> e <i>Gap Test</i>	Coeficiente de Autocorrelação, <i>Frequency Test</i> e <i>Gap Test</i>	<i>Frequency Test</i> e <i>Gap Test</i>

As aplicações descritas nas publicações são identificadas no quadro da seguinte forma:

- A aplicação descrita por Mishra e Bali (2013) é denominada App 1;
- A aplicação descrita por Jhajharia et al. (2013a) é denominada App 2;
- A aplicação descrita por Jhajharia et al. (2013b) é denominada App 3;
- A aplicação desenvolvida no TCC é denominada GCK, sigla da própria aplicação.

1.4 Organização da Monografia

Esta monografia está organizada da seguinte forma:

- O Capítulo 2 apresenta o método de desenvolvimento adotado para o trabalho e descreve os aspectos de inovação e aprimoramento decorrentes do TCC;
- O Capítulo 3 apresenta todos os elementos relativos ao desenvolvimento do TCC, desde as ferramentas utilizadas até a arquitetura e os principais conceitos abordados no desenvolvimento do trabalho.
- O Capítulo 4 descreve as funcionalidades do GenCryptoKey;
- O Capítulo 5 descreve a avaliação realizada sobre o artefato produzido e a análise dos resultados nela obtidos;
- O Capítulo 6 finaliza o documento com as conclusões sobre o trabalho. São apresentadas a a análise do grau de complexidade do artefato, dificuldades enfrentadas, aspectos de qualidade de *software* contemplados pelo GenCryptoKey e propostas de trabalho futuro, com possíveis melhorias do produto.

2 MÉTODO DE DESENVOLVIMENTO E ASPECTOS DE INOVAÇÃO E APRIMORAMENTO

A realização de um Trabalho de Conclusão de Curso não tem por objetivo somente colocar em prática, simultaneamente, um amplo conjunto de teorias, práticas e técnicas estudadas na graduação. Espera-se também que o autor seja capaz de incorporar novos conhecimentos e, em alguns casos, até mesmo contribuir com a ciência da área, através de inovação(ões) consequente(s) do próprio TCC.

O aprimoramento pela incorporação de novos conhecimentos não é imediato, mas sim decorrente da superação de dificuldades e desafios que surgem ao longo do período de desenvolvimento. Esses são, muitas vezes, motivos de frustração e desconfiança da viabilidade de certas abordagens, mas a sua superação torna-se indicativo de um trabalho bem sucedido, gerador de resultados que, ao menos, atendem às expectativas. Como já era de se esperar, a experiência do TCC para o autor não foi diferente.

Esse capítulo apresenta o método de desenvolvimento escolhido e os aspectos de inovação e aprimoramento decorrentes do TCC.

2.1 Método de Desenvolvimento

O método utilizado para o desenvolvimento do TCC foi o Scrum solo, uma adaptação do Scrum (SCHWABBER, 2013), para projetos que são desenvolvidos individualmente.

O Scrum é um método ágil de desenvolvimento de *software* que aplica-se muito bem a projetos que demandam revisões constantes dos requisitos, como é o caso do TCC e por essa razão foi o método escolhido.

A principal atividade deste método é a *sprint*, nome atribuído a uma porção de tempo (*time-box*), que pode durar entre duas e quatro semanas, sendo que durante o desenvolvimento do TCC, cada uma teve duração de três

semanas. E ao fim de cada uma dessas *sprints*, foi entregue um incremento do produto.

Ao final de cada uma das 8 *sprints* divididas no cronograma do TCC, foi realizada uma análise da *sprint retrospective*, atividade definida pelo método, que tem como principal função avaliar os pontos de sucesso e fracasso da *sprint* que se passou de modo que se possa melhorar o desempenho nas *sprints* seguintes. O autor usou esse momento também para avaliar as atividades que não puderam ser finalizadas na *sprint* e realocá-las no *product backlog*, para que fossem replanejadas e executadas em outra *sprint*.

A Figura 1 ilustra o funcionamento de uma *sprint* segundo o método.

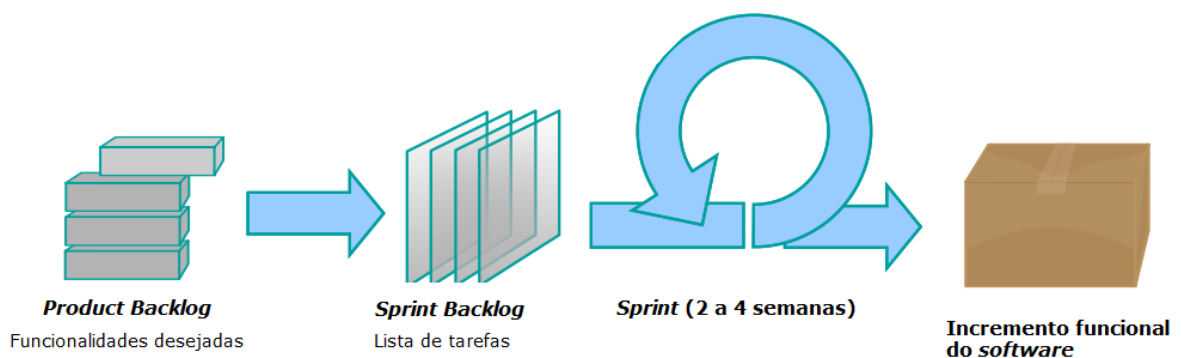


Figura 1. Funcionamento do método Scrum. **Baseada em:** Creative Commons

2.2 Backup

O primeiro aprimoramento foi decorrente da criação da política de *backup* para os arquivos do projeto. Nenhuma das ferramentas usadas para esse fim (rsync e Crontab, já mencionadas anteriormente) eram de conhecimento prévio do autor, além de ter sido o primeiro contato com *script*, o que nada mais é que uma rotina de tarefas escrita numa linguagem que um *shell* do sistema operacional seja capaz de interpretar – como o TCC foi desenvolvido primordialmente em um sistema operacional Linux, o *shell* padrão desse sistema, o *Bash*, foi o utilizado. Para o uso correto do rsync e do Crontab foi necessária leitura das respectivas documentações e realização de alguns experimentos antes das configurações desejadas terem sido alcançadas.

2.3 Versionamento de Código

Outro aspecto de aprimoramento foi quanto ao uso da ferramenta Git, atrelada ao serviço GitHub de repositório em nuvem, para controle de versão de códigos-fonte. O autor já havia experimentado versionamento de *software* previamente, porém nunca de forma tão intensa e frequente quanto durante o desenvolvimento do TCC e, muito menos, com o Git. E a razão pela qual essa ferramenta se difere das suas similares é que ela usa conceitos distintos para a submissão e a manutenção de histórico dos arquivos versionados, a começar pelo comando *commit*, que, ao contrário do que se esperava, dado o costume com outras ferramentas, não realiza escrita no repositório final, entre outras abordagens.

2.4 Algoritmo Genético

O conceito fundamental abordado no Trabalho de Conclusão de Curso descrito por essa monografia, o Algoritmo Genético, foi também um com o qual o autor tinha sequer alguma experiência prévia ao início do trabalho. Portanto, leitura e exercícios de estudo sobre o tema foram essenciais.

Embora tenha-se concluído que tanto o conceito quanto a implementação em código são, em grande parte, simples, a etapa conhecida como “Seleção” mostrou-se mais complexa do que se esperava, dado o fato de depender de um atributo chamado “*Fitness*” – explicado detalhadamente na seção 4.3.3 – o qual é particularmente complexo de se calcular e, conseqüentemente, com o qual se gastou a maior parte do tempo.

2.5 Criptografia Rivest-Shamir-Adleman

Apesar do autor já ter algum conhecimento prévio sobre o método criptográfico Rivest-Shamir-Adleman (RSA), foi um ponto que requisitou um estudo aprofundado para que se pudesse compreender plenamente os conceitos ao ponto de realizar a implementação corretamente.

Detalhes sobre o RSA e a razão de escolha do mesmo são apresentados na seção 4.3.1.

2.6 Testes Estatísticos

Os quatro aspectos descritos anteriormente foram identificados e previstos desde o momento de planejamento das atividades do TCC. Não se contava, entretanto, com este quinto: testes estatísticos.

Esses testes estatísticos foram necessários em dois momentos fundamentais do trabalho: composição do atributo *fitness* do AG e avaliação do TCC.

Além de não experimentado nada parecido previamente ao TCC, o autor foi surpreendido pela necessidade dos mesmos, no momento de estudo aprofundado sobre a aplicação de AGs para a geração de chaves criptográficas, tendo sido, portanto, um dos pontos que mais demandou tempo no desenvolvimento do trabalho.

Os testes aplicados e os conceitos que envolvem suas aplicações são detalhados na seção 4.3.3.

3 DESENVOLVIMENTO

Este capítulo dedica-se à apresentação e ao detalhamento das ferramentas utilizadas, da arquitetura do artefato de *software* e dos principais conceitos aplicados sobre o desenvolvimento do mesmo.

3.1 Ferramentas Utilizadas

O desenvolvimento do artefato de *software* do Trabalho de Conclusão de Curso foi realizado utilizando-se máquinas pessoais do autor, principalmente, um computador *desktop* com a seguinte configuração de *hardware*: um processador de 3.5GHz com quatro núcleos, memória RAM de 8GB a 2133MHz e *Solid State Drive* (SSD) de 120GB; o sistema operacional utilizado foi o Linux Ubuntu, versão 12.04. Em caráter secundário, foi usado um computador *laptop* equipado com processador de 2.5GHz com dois núcleos, memória RAM de 8GB a 1600MHz e *Solid State Hybrid Drive* (SSHD) de 1TB. Nesse *laptop*, o sistema operacional utilizado foi o Windows 8.1.

A configuração descrita provê qualidade a ambas as máquinas utilizadas, em especial à primeira, mas não é imprescindível para a reprodução do trabalho descrito nessa monografia. O benefício mais significativo provido pelos processadores e memórias de melhor desempenho é o tempo reduzido na execução dos testes mais longos do artefato, o que pode ser um fator levado em consideração.

3.1.1 Linguagem de Programação e Ambiente de Desenvolvimento

O *software* foi programado na linguagem Java, utilizando-se o *Software Development Kit* (SDK) 7. Por essa razão, optou-se por tirar proveito de um ambiente integrado de desenvolvimento (ou *Integrated Development Environment*, IDE, como é mais conhecido), pois esse tipo de ferramenta provê algumas facilidades que aceleram a produção de código, tais como análise sintática em tempo real, sugestões de correção e compilação automática. O IDE escolhido foi o Eclipse, modelo Luna, versão 4.4.1.

As escolhas se deram pelo fato do autor possuir experiência de, aproximadamente, 4 anos com ambos, tanto no contexto acadêmico quanto profissional.

3.1.2 Elaboração dos Diagramas

Os diagramas criados para modelar o *software* desenvolvido foram desenvolvidos em diferentes ferramentas. O Quadro 2 apresenta quais foram eles.

Quadro 2. Ferramentas para elaboração de diagramas.

Ferramenta	Foco	Versão	Disponibilidade
yEd	Diagrama de Arquitetura	3.12.2	Grátis, multiplataforma
PlantUML	Diagramas de Atividade e Sequência	8009	Grátis, multiplataforma
CorelDraw	Diagrama de Arquitetura	X6 (16)	Pago, somente Windows

É interessante ressaltar a ferramenta PlantUML pois essa permite que o diagrama a ser criado seja descrito textualmente, garantindo a padronização dos elementos que compõem cada tipo de diagrama e do resultado final como um todo. Os Apêndices A e B apresentam a descrição textual dos diagramas de atividade e sequência, os quais estão ilustrados em figuras nas seções 3.3.2.

3.1.3 Controle de Versões e Revisões de Código

O uso de uma ferramenta de versionamento de código é imprescindível para o desenvolvimento de um projeto de longa duração, que pode tomar diferentes rumos ao longo de sua evolução e que, acima de tudo, dependa de um histórico para possíveis *rollbacks*, isto é, a restauração de um estado passado consistente do código no evento de erros, mantendo assim sua integridade.

Para se obter um controle e bom gerenciamento do código fonte do *software* em desenvolvimento, foi utilizado o sistema de controle de versões e revisões Git, instalado no computador do autor, aliado a um repositório obtido através do serviço GitHub, podendo-se assim versionar o código e armazenar as revisões em um servidor remoto.

3.1.4 Política de *Backup*

O *backup* contínuo do projeto em desenvolvimento foi essencial para a segurança do trabalho e, às vezes, é confundido com as atividades de versionamento: a rigor, o versionamento, como o próprio nome sugere, só é realizado em versões, que devem estar inteiras (ou funcionais, no caso de código fonte), para que no repositório se mantenha sempre um estado estável e consistente do projeto; o *backup*, por sua vez, é realizado com a intenção de criar cópias de segurança do material, esteja esse em qualquer estado, acabado ou não.

De modo a não haver esquecimento das fundamentais tarefas de *backup*, escolheu-se algumas ferramentas para realizá-las automaticamente, as quais são apresentadas no Apêndice A.

No caso do TCC, foram usadas duas políticas para *backup*: parcial – esse subdividido em duas categorias: diário e semanal – e total. Em ambos os casos, as cópias de segurança (ou *snapshots*) foram armazenados no repositório Dropbox (versão 2.6.31) de propriedade do autor.

Sob esse esquema, as ferramentas foram integradas de tal maneira que uma cópia de segurança foi gerada todos os dias às 21:40, outra refletindo o estado semanal foi gerada todos os domingos às 22:00 e o *backup* total foi realizado nos dias 30 de cada mês, às 22:15.

Em muitos casos, políticas de *backup* são aplicadas e as cópias de segurança acabam não sendo utilizadas. Este é o cenário ideal, mas só se percebe o valor das mesmas quando se fazem necessárias. Foi o caso em duas ocasiões durante o desenvolvimento do TCC, a última delas justamente com

relação à escrita desse documento, quando o arquivo do mesmo foi corrompido e só pode ser recuperado graças ao *snapshot* do dia anterior.

As configurações aplicadas no *rsync* (versão 3.0.9) e no *Crontab* (versão 14.04) estão ilustradas nos Apêndices B e C, respectivamente.

3.2 Arquitetura

O *software* desenvolvido no TCC seguiu o padrão arquitetural *Model-View-Controller* (mais conhecido por sua sigla, MVC), padrão esse que visa separar o conteúdo do programa em camadas, as quais têm interações entre si, mas são independentemente responsáveis cada qual por um tipo de tratamento da informação.

A camada *Model*, ou modelo, é responsável pelas regras de negócio e persistência de dados da aplicação. É através dela, por exemplo, que a aplicação interage com um banco de dados.

A camada, *View*, ou apresentação, é responsável unicamente por apresentar os dados manipulados pela aplicação ao usuário da mesma. Essa apresentação pode se dar por meio de texto, telas – como é o caso na aplicação desenvolvida – entre outros.

A camada *Controller*, ou controle, por sua vez, é a que desempenha o papel de receber as entradas do usuário, manipulá-las e convertê-las em comandos ou ações para as outras duas camadas.

Este *design pattern* é amplamente aplicado em projetos que utilizam o paradigma de orientação a objetos, por permitir a separação dos componentes por funções bem definidas, o que resulta em organização, facilidade de reutilização de código e manutenibilidade.

O MVC possui qualidades que facilitam o entendimento de uma aplicação do ponto de vista *top-level*. O nítido desacoplamento de componentes, por exemplo, permite modificações em regras de negócio sem causar transtornos outros quaisquer. Além disso, provê facilidade de manutenção, atualização e a

vantagem de se poder ter múltiplas *views* utilizando um mesmo *controller* e um mesmo *model*.

Por outro lado, o padrão MVC também apresenta algumas desvantagens. Dentre elas, as que tiveram impacto direto no desenvolvimento foram a dificuldade de aplicá-lo corretamente sem ampla experiência prévia (logo muitas correções) e a ineficiência do acesso a dados pelos componentes (ou também chamados módulos) da camada *View*.

A Figura 2 ilustra genericamente o padrão descrito, usado como base para o diagrama arquitetura da aplicação, ilustrado na Figura 3.

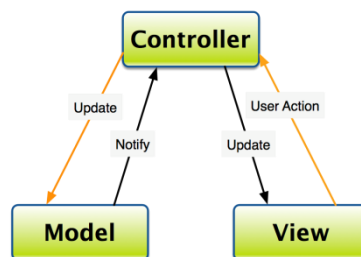


Figura 2. Modelo genérico do padrão arquitetural MVC. **Fonte:** Binpress (2011)

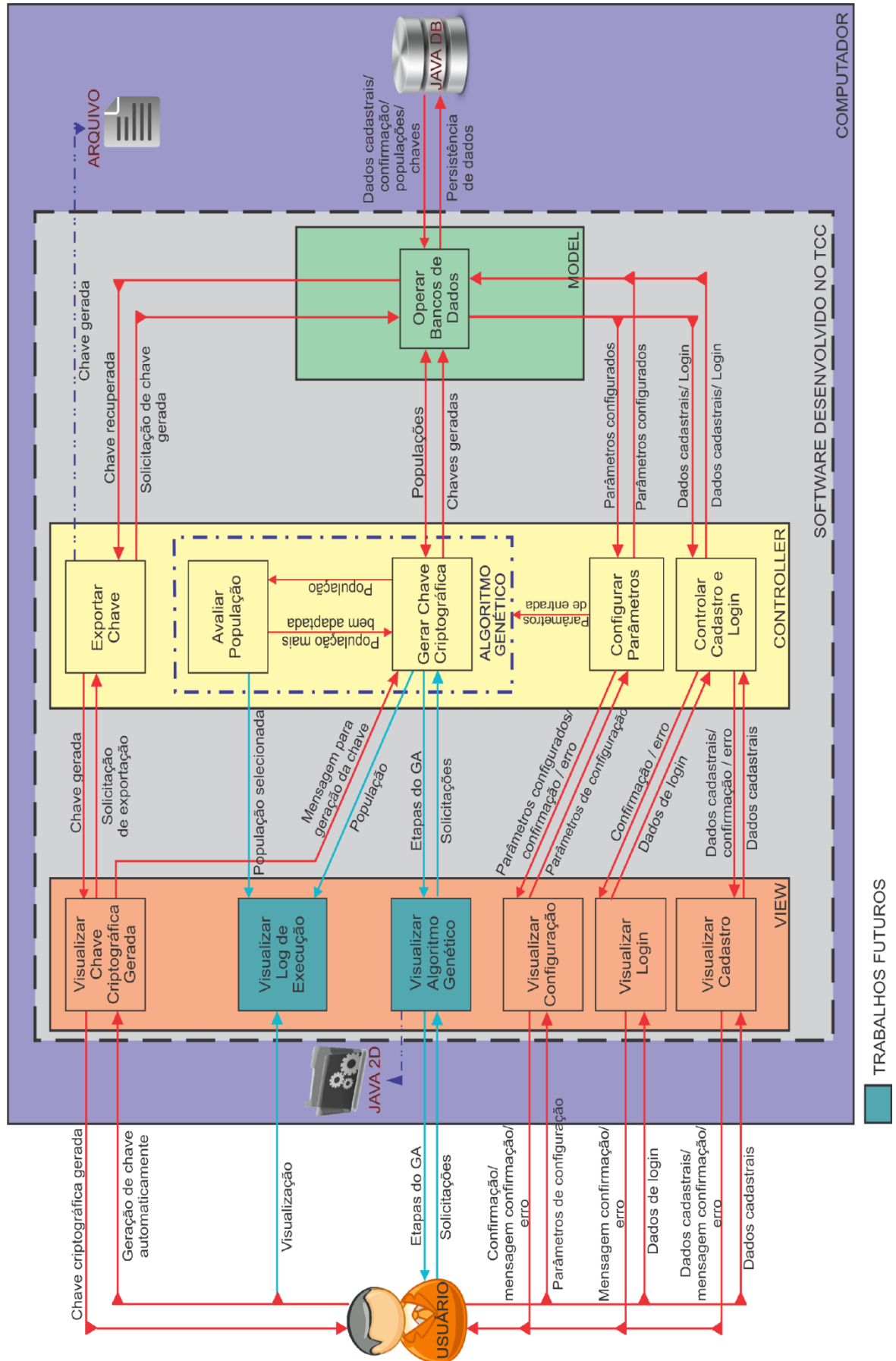


Figura 3. Diagrama de arquitetura do artefato de software

3.3 Geração de Chaves

O propósito do artefato de *software* produzido é a geração de chaves criptográficas. Quando se diz “chaves”, deve-se entender um par de chaves, pois a aplicação final dessas é no modelo assimétrico *Public Key Cryptography*.

3.3.1 RSA

Dentre diversos modelos assimétricos, escolheu-se usar o Rivest-Shamir-Adleman, ou simplesmente RSA, como é mais conhecido. Essa escolha se deu por ser um dos primeiros sistemas criptográficos e, por conta disso, um dos mais conhecidos, reconhecidos e utilizados para a transmissão segura de dados.

Atualmente existem variações do RSA – que, na indústria, não é aplicado na sua forma pura – no entanto, usou-se a definição pura do modelo por ser suficientemente boa para a geração de chaves criptográficas robustas.

Afirma-se que o RSA produz chaves criptográficas robustas pois ele baseia-se na dificuldade de se fatorar grandes números inteiros compostos em números primos, um problema matemático considerado difícil. Na prática, se uma mensagem é cifrada com uma das chaves do RSA, e esta for suficientemente grande, apenas um atacante com pleno conhecimento dos números primos que foram usados na composição da mesma é capaz de decifrar a mensagem (Rivest et al., 1977). Apesar do algoritmo concebido por Shor (1994) ser capaz de resolver o problema em tempo polinomial de complexidade $O(n^3)$, ele depende de computação quântica. Em contrapartida, acredita-se que o método clássico mais rápido conhecido requer tempo superpolinomial, classe essa que, com os conhecimentos dos dias atuais, contempla problemas cujas soluções durariam períodos superiores à idade do universo.

3.3.2 Algoritmo Genético

O método de geração de chaves funciona a partir de uma entrada: dois números primos. É justamente a geração desses números o objetivo do AG desenvolvido no TCC e implementado no artefato de *software*. No contexto do Algoritmo Genético, um indivíduo é composto por três genes, dois dos quais são os números primos usados para a geração do módulo da chave criptográfica e o terceiro outro número primo – este não é gerado – que serve como expoente público do modelo RSA. A Figura 4 ilustra a sequência do AG.

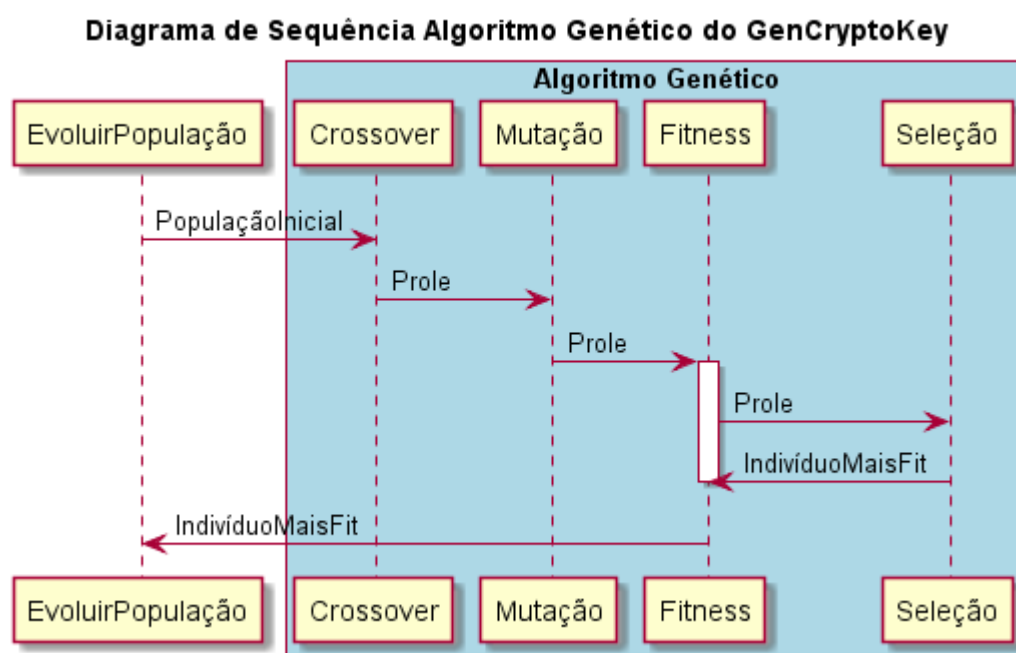


Figura 4. Diagrama de sequência do Algoritmo Genético

Muitas maneiras distintas de se produzir esses números poderiam ser aplicadas. Uma delas seria usar *Random Number Generator* (RNG) ou um *Pseudorandom Number Generator* (PRNG) para se obter números diversos, seguido da aplicação de um (ou mais) teste(s) de primalidade, com a finalidade de validar, ou não, se o número gerado é primo e se, portanto, poderia ser usado. No entanto, bons e criptograficamente seguros PRNGs não são facilmente obtidos e muito menos desenvolvidos. Além disso, para números primos grandes – e por “grande” entende-se maiores que 1024 *bits* – mesmo com o algoritmo desenvolvido por Agrawal et al. (2003), o mais rápido teste de primalidade conhecido até hoje (executa em tempo polinomial), testar a primalidade continua uma tarefa que consome bastante tempo para o propósito de geração de grandes sequências de números primos.

Assim sendo, a alternativa escolhida foi a de usar um método de geração de números que provavelmente são primos, proveniente da própria *Application Programming Interface* (API) da linguagem de programação usada para o desenvolvimento: Java.

A API provê um método chamado *probablePrime* da classe *BigInteger* que retorna um número de tamanho arbitrário (definido pelo programador), provavelmente primo, com uma probabilidade padrão do número retornado ser composto menor do que 2^{-100} (JAVADOC, 2014). Em outras palavras, a probabilidade do número retornado não ser primo é extraordinariamente baixa a ponto de ser negligenciável. Além da baixa probabilidade, o método permite que a precisão padrão seja alterada, possibilitando ainda um aumento, mas isso não foi necessário.

Tamanha qualidade na geração desses números é resultado de uma série de testes Miller-Rabin (RABIN, 1980), combinada com um teste Lucas-Lehmer (LEHMER, 1930) de primalidade.

Entretanto, apenas gerar números primos grandes não seria suficiente para garantir a robustez buscada como objetivo. Por essa razão, logo após a obtenção dos números pelo método da API que o Algoritmo Genético é finalmente iniciado, com o propósito único e exclusivo de: um, aumentar a aleatoriedade dos dois primos finais que servem de entrada para o RSA; e, dois, aumentar a entropia dos mesmos. A consequência disso é um par final de números primos com bons valores de *fitness*, que por sua vez indica bons candidatos para o processo de geração de chaves.

Primeiramente, a aplicação gera uma população inicial de indivíduos. A partir dela, é que então o algoritmo pode, de fato, começar.

O ponto de entrada do AG é o *Crossover*, cuja função, como o próprio nome sugere, é a de realizar cruzamentos entre genes de diferentes pais, de modo a criar novos indivíduos e, conseqüentemente, aumentar e diversificar a população. Em seguida, dada uma probabilidade, os indivíduos da população resultante podem sofrer mutação em um ou mais de seus genes.

Uma vez aplicadas as devidas mutações, dá-se início então à parte mais importante do algoritmo: o cálculo de *fitness* dos indivíduos. Essa etapa é explicada em mais detalhes no item 4.3.3.

Quando todos os indivíduos de uma geração têm seus valores de *fitness* calculados e a eles atribuídos, uma seleção por *rank* preserva um grupo de indivíduos que continuará a evolução da população na geração seguinte. E, ao final do número de gerações, o algoritmo retorna o indivíduo com mais valor de *fitness* da última geração - teoricamente o mais bem adaptado – para ser usado como módulo do par de chaves RSA. A Figura 5 ilustra as atividades do AG.

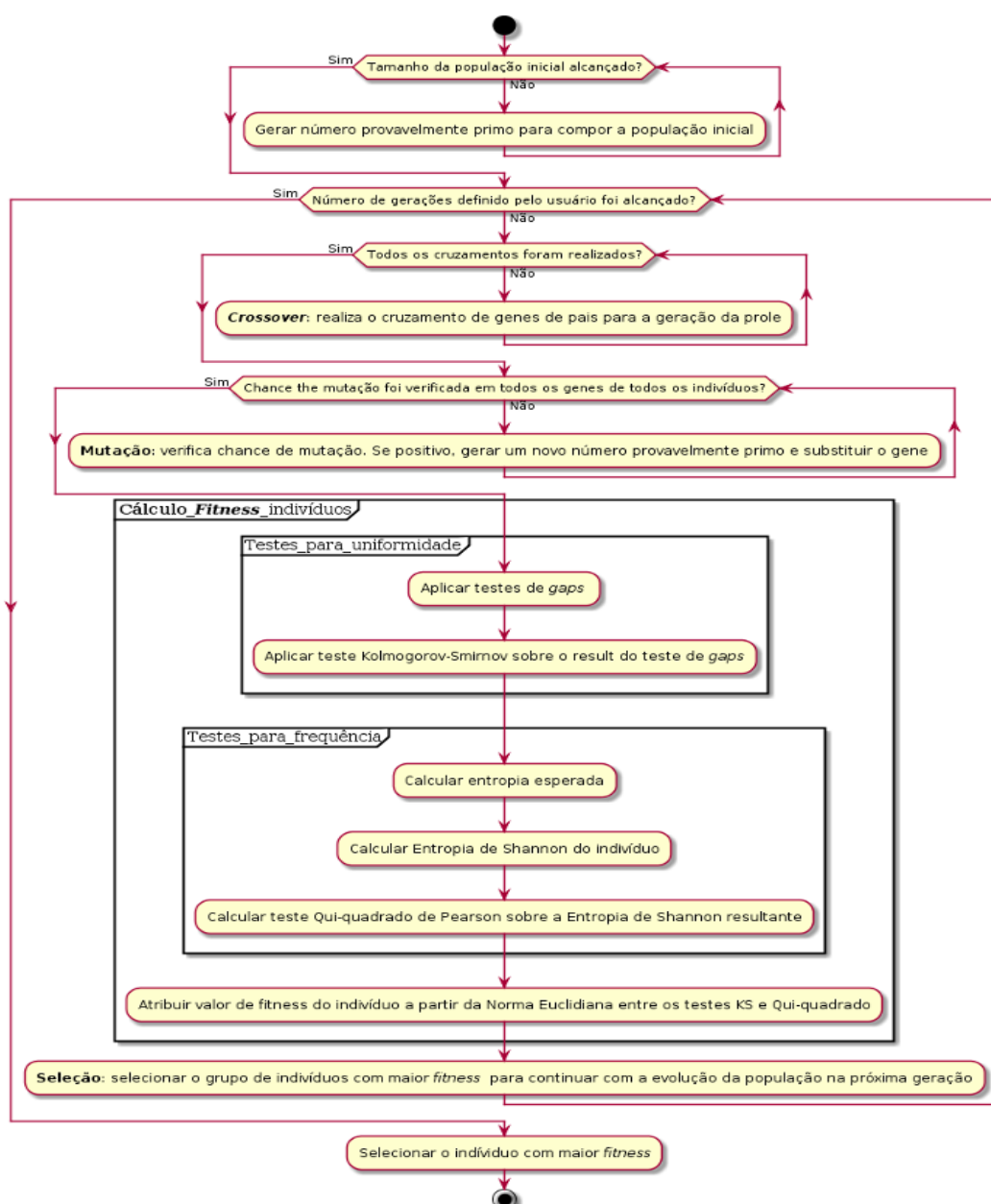


Figura 5. Diagrama de atividades do Algoritmo Genético

A última informação necessária para a composição do par de chaves é o número usado como expoente público, necessário para a cifragem de mensagens. Esse, como dito anteriormente, não é gerado em nenhum momento; pelo contrário, é constante: 65537.

Existem outros valores que podem servir como expoente público, mas a escolha desse não é arbitrária. O expoente público e deve ser um valor que satisfaça $1 < e < \varphi(n)$ e o máximo divisor comum entre e e $\varphi(n)$ seja igual a 1, onde n é o módulo RSA e φ é a função *phi* de Euler (Rivest et al., 1977). No entanto, quanto maior for o número escolhido, mais demoradas se tornam as operações de cifragem no momento de uso do modelo, o que é indesejado, ao passo que valores muito pequenos para e comprometem a segurança da mensagem cifrada, como já foi mostrado no estudo de Boneh (1999).

Por essas razões, o valor 65537 é amplamente utilizado na indústria e até mesmo adotado como padrão em muitos sistemas, o que significa que a escolha de outro valor implicaria em incompatibilidade.

3.3.3 Cálculo de *Fitness*

A etapa de cálculo de *fitness* dos indivíduos em um algoritmo genético pode ser considerada a mais importante e, também, a mais delicada e sensível a erros, pois é, fundamentalmente, através dos valores de *fitness* atribuídos que se determina a partir de quais indivíduos a população continua evoluindo.

Isso quer dizer que uma função de *fitness* ruim pode levar a um resultado final não ótimo, desperdiçando-se assim o potencial do AG para a escolha de um indivíduo para dar entrada no RSA.

A dificuldade da elaboração de uma boa função de *fitness* está ligada ao fato de não haver uma regra; é completamente arbitrária (dentro do contexto no qual se está aplicando o AG, obviamente). Na natureza, de onde são inspirados os Algoritmos Evolutivos, *fitness* não é uma qualidade alcançada através de fórmula(s), mas sim um estado de desenvolvimento resultante de

inúmeros fatores intrínsecos à evolução e às constantes mudanças e adaptações dos seres vivos, aspectos que não são controlados. Em contrapartida, no processo de elaboração do AG, é responsabilidade do programador entender qual é a maneira mais adequada de se avaliar um indivíduo.

Como sabe-se que a característica principal sendo buscada é a aleatoriedade, deve-se avaliar a uniformidade e a independência (SANTOS, 1999) dos *bits* dos quais são compostos os indivíduos, o que pode ser realizado através de testes estatísticos (há uma explicação detalhada na seção 6.2). Portanto, para se avaliar isso aplicou-se alguns dos mais convencionais testes: Kolmogorov-Smirnov com *Gap Test* para uniformidade e Qui-quadrado de Pearson (mais conhecido pelo nome em inglês, *Chi-square test*, ou ainda por χ^2 -test), com entropia de Shannon para independência. O Anexos A e B apresentam as tabelas de valores críticos que devem ser levados em consideração quando da aplicação dos testes Kolmogorov-Smirnov e *Chi-square*, respectivamente.

Na avaliação da uniformidade, no caso de sequências binárias, o *Gap Test* conta o número de *bits* de um mesmo valor que ocorre entre repetições do *bit* de valor contrário (KNUTH, 1998). Por exemplo, na sequência “01110110” existem dois *gaps*, ou intervalos, de *bits* 1 entre os *bits* 0 e seus tamanhos são 3 e 2, respectivamente.

Sobre o *Gap Test* é então aplicado o Kolmogorov-Smirnov, para se poder comparar o resultado obtido com o número de *gaps* esperado para a dada sequência. Esse tipo de análise serve para se medir o grau de aderência entre a distribuição de uma amostra supostamente aleatória (como é o caso dos indivíduos gerados na evolução do AG) e a distribuição uniforme teórica, ou seja, aquela que é esperada para aquele tipo de sequência, com o mesmo tamanho.

A segunda parte da avaliação, para independência, é por conta do *Chi-square test*. Esse, por sua vez, é mais aplicado sobre distribuições de números contínuos, o que não é o caso das sequências dos indivíduos do AG, pois são binárias. Por essa razão, de modo a normalizar a entrada para o teste, optou-se por calcular a entropia do indivíduo. O termo “entropia”, no contexto de

criptografia, pode ser entendido como desordem ou imprevisibilidade, como descrito por Shannon (1948).

Finalmente, de posse dos resultados de ambos os testes, é possível atribuir-se uma nota a cada indivíduo. Tal nota, entretanto, não pode ser meramente uma soma dos resultados de cada um dos testes, pois isso provocaria a geração de notas não significativas para comparação. Em outras palavras, com essa abordagem poder-se-ia obter uma nota considerada boa para um indivíduo que é avaliado positivamente (sucesso) em um dos testes, porém negativamente (fracasso) no outro, ou seja, o fracasso poderia estar sendo mascarado pela nota do sucesso.

Para contornar o problema, a nota atribuída é uma norma euclidiana dos valores obtidos nos testes, assim resultando em um valor ponderado. Essa nota (*fitness*), é usada para julgar a qualidade de dado indivíduo no momento da seleção.

4 FUNCIONALIDADES DO ARTEFATO

Este capítulo tem por objetivo apresentar e detalhar as funcionalidades incorporadas no GenCryptoKey.

4.1 Apresentação e *Help*

Conforme pode ser observado na Figura 6, o aplicativo possui uma interface simplista, desenvolvida com a intenção de promover facilidade de uso.

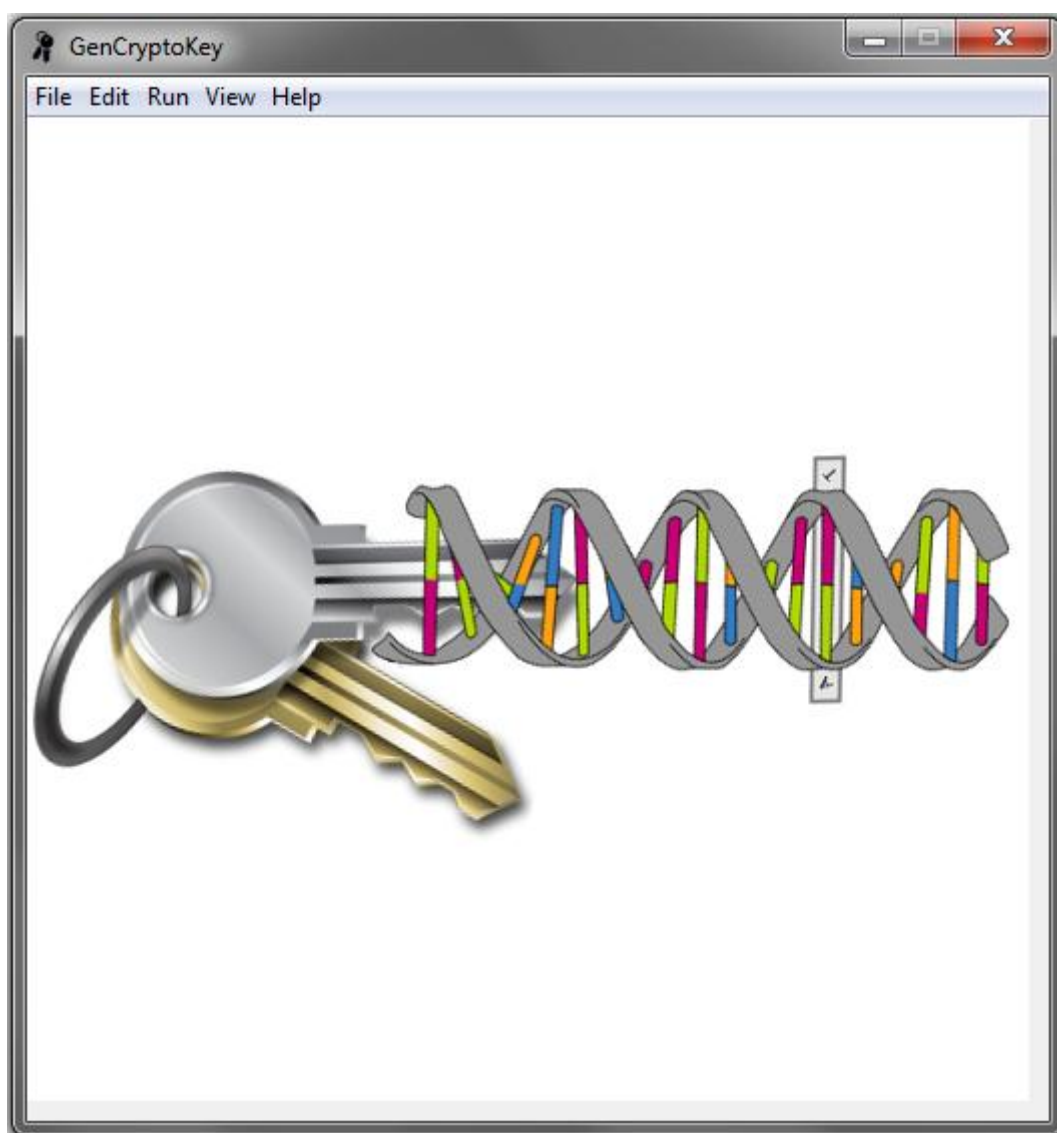


Figura 6. Tela inicial da aplicação GenCryptoKey

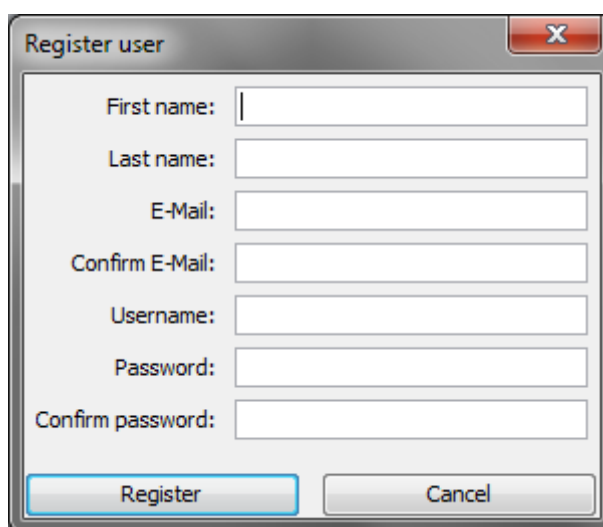
É a partir da tela inicial, que o usuário tem acesso a todas as funcionalidades da aplicação, através das opções no *menu* na parte superior. A

opção *Help*, por exemplo, contém um *submenu* para acesso a uma janela que contém informações referentes ao autor e ao orientador (nome e *e-mail* para contato) e à aplicação, tais como número de versão e ano de desenvolvimento.

4.2 Cadastro de Usuário e *Login*

Por se tratar de um aplicativo que se insere no contexto de segurança da informação e proteção de dados sensíveis, o GenCryptoKey foi concebido visando-se o uso particular de um único usuário, isto é, no espaço em disco de um determinado usuário em um dado sistema operacional, apenas uma instância do aplicativo estará disponível, à qual também um único usuário terá acesso.

Assim sendo, uma vez que o aplicativo é instalado e executado no sistema, a primeira medida que o usuário deve tomar é registrar-se, de modo a tornar-se dono daquela instância. A Figura 7 ilustra a tela de registro do usuário.

A imagem mostra uma janela de diálogo intitulada "Register user" com uma barra de título cinza e um botão de fechar (X) no canto superior direito. O corpo da janela é branco e contém sete campos de texto empilhados verticalmente, cada um com um rótulo à esquerda: "First name:", "Last name:", "E-Mail:", "Confirm E-Mail:", "Username:", "Password:", e "Confirm password:". Abaixo dos campos, há dois botões: "Register" (destacado com uma borda azul) e "Cancel" (cinza).

First name:	<input type="text"/>
Last name:	<input type="text"/>
E-Mail:	<input type="text"/>
Confirm E-Mail:	<input type="text"/>
Username:	<input type="text"/>
Password:	<input type="password"/>
Confirm password:	<input type="password"/>
<input type="button" value="Register"/>	
<input type="button" value="Cancel"/>	

Figura 7. Tela de cadastro do usuário

Os dados coletados pelo formulário de cadastro são utilizados para identificação, validação de *login* e possível necessidade de recuperação de senha. Por essas razões, filtros rigorosos foram aplicados de modo a não permitir que dados inconsistentes, tais como endereços de *e-mail* inválidos ou senhas consideradas fracas, sejam aceitos na composição do cadastro.

Senhas fracas geralmente são sequências de poucas letras, muitas vezes não contendo sequer um dígito e/ou um caractere especial. Como a senha

é a peça mais importante na segurança do uso do aplicativo, o usuário é obrigado a criar uma senha de ao menos oito caracteres, sem conter espaço em branco, além de cumprir com alguns requisitos que a tornam minimamente robusta:

- Uma letra minúscula;
- Uma letra maiúscula;
- Um dígito;
- Um caractere especial (@, #, \$, %, ^, &, +, =);

Conforme são inseridos, os os dados cadastrais passam por um filtro validador e então a interface encarrega-se de informar ao usuário os campos que contém dados válidos (em verde) e inválidos (em vermelho), como mostra a Figura 8.

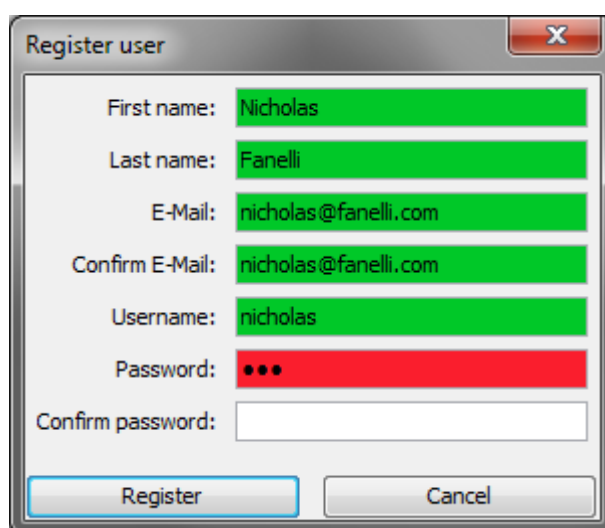
A imagem mostra uma janela de software intitulada "Register user" com uma barra de título cinza e um botão de fechar "X" no canto superior direito. O formulário contém os seguintes campos: "First name:" com o valor "Nicholas" em um campo verde; "Last name:" com o valor "Fanelli" em um campo verde; "E-Mail:" com o valor "nicholas@fanelli.com" em um campo verde; "Confirm E-Mail:" com o valor "nicholas@fanelli.com" em um campo verde; "Username:" com o valor "nicholas" em um campo verde; "Password:" com três pontos vermelhos em um campo vermelho; e "Confirm password:" com um campo branco vazio. Na base da janela, há dois botões: "Register" (destacado com uma borda azul) e "Cancel".

Figura 8. Cadastro do usuário com exemplos do filtro validador

Uma vez concluída essa etapa de cadastro, o usuário está apto a realizar *login* e, ao mesmo tempo, a opção de cadastro é desabilitada, de tal modo que nenhum outro usuário poderá ser cadastrado para utilizar aquela dada instância do aplicativo.

A opção de *login*, por sua vez, só é habilitada quando já existe um usuário registrado. É através dela que o usuário insere os dados que registrou no momento do cadastro (nome de usuário e senha) para obter acesso aos demais recursos do aplicativo.

É também através dessa interface que o usuário pode solicitar a recuperação de senha, que, uma vez solicitada, gera uma mensagem que é disparada para o endereço de *e-mail* cadastrado, contendo instruções para a recuperação da senha.

4.3 Configuração de Parâmetros

Uma das características mais fortes do GenCryptoKey é sua flexibilidade quanto à parametrização do AG para a geração do par de chaves. Isso se dá através de uma interface bastante simples, ilustrada na Figura 9, por meio da qual o usuário tem a liberdade de escolher os valores que mais lhe convém para a geração de um determinado par de chaves, respeitando-se, entretanto, alguns limites impostos pelo próprio aplicativo para que não se perca sentido e utilidade, tais como a obrigatoriedade de valores positivos para a taxa de mutação.

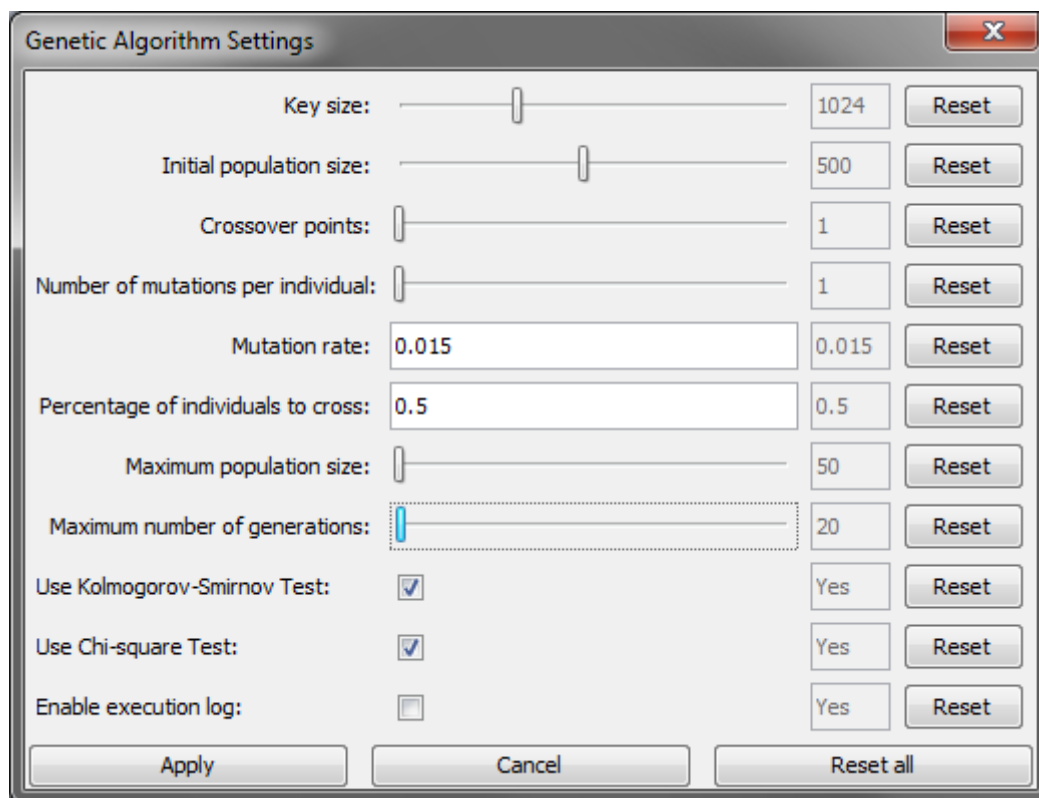


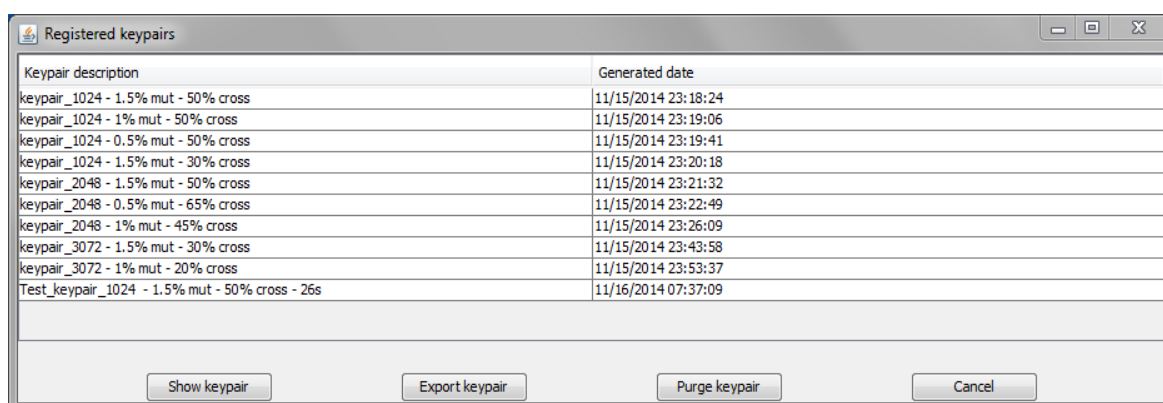
Figura 9. Tela de configuração dos parâmetros para o Algoritmo Genético

Essa configuração, no entanto, pode ser alterada a qualquer momento, exceto durante a geração do par de chaves, momento no qual o AG está fazendo uso dos parâmetros previamente definidos.

4.4 Visualização de Chaves Geradas

Como a aplicação provê ao usuário a possibilidade de armazenar chaves geradas numa base de dados própria, uma das funcionalidades imprescindíveis é a visualização, a qualquer momento, das chaves previamente geradas.

A Figura 10 ilustra a tela de visualização dessas chaves. Além de prover a visualização, a tela também dá acesso a dois outros recursos importantes no que diz respeito ao tratamento das chaves: a remoção (eliminação da base de dados) e a exportação, que é explicada em mais detalhes na seção seguinte.



Keypair description	Generated date
keypair_1024 - 1.5% mut - 50% cross	11/15/2014 23:18:24
keypair_1024 - 1% mut - 50% cross	11/15/2014 23:19:06
keypair_1024 - 0.5% mut - 50% cross	11/15/2014 23:19:41
keypair_1024 - 1.5% mut - 30% cross	11/15/2014 23:20:18
keypair_2048 - 1.5% mut - 50% cross	11/15/2014 23:21:32
keypair_2048 - 0.5% mut - 65% cross	11/15/2014 23:22:49
keypair_2048 - 1% mut - 45% cross	11/15/2014 23:26:09
keypair_3072 - 1.5% mut - 30% cross	11/15/2014 23:43:58
keypair_3072 - 1% mut - 20% cross	11/15/2014 23:53:37
Test_keypair_1024 - 1.5% mut - 50% cross - 26s	11/16/2014 07:37:09

Buttons: Show keypair, Export keypair, Purge keypair, Cancel

Figura 10. Tela de visualização de chaves armazenadas no banco de dados

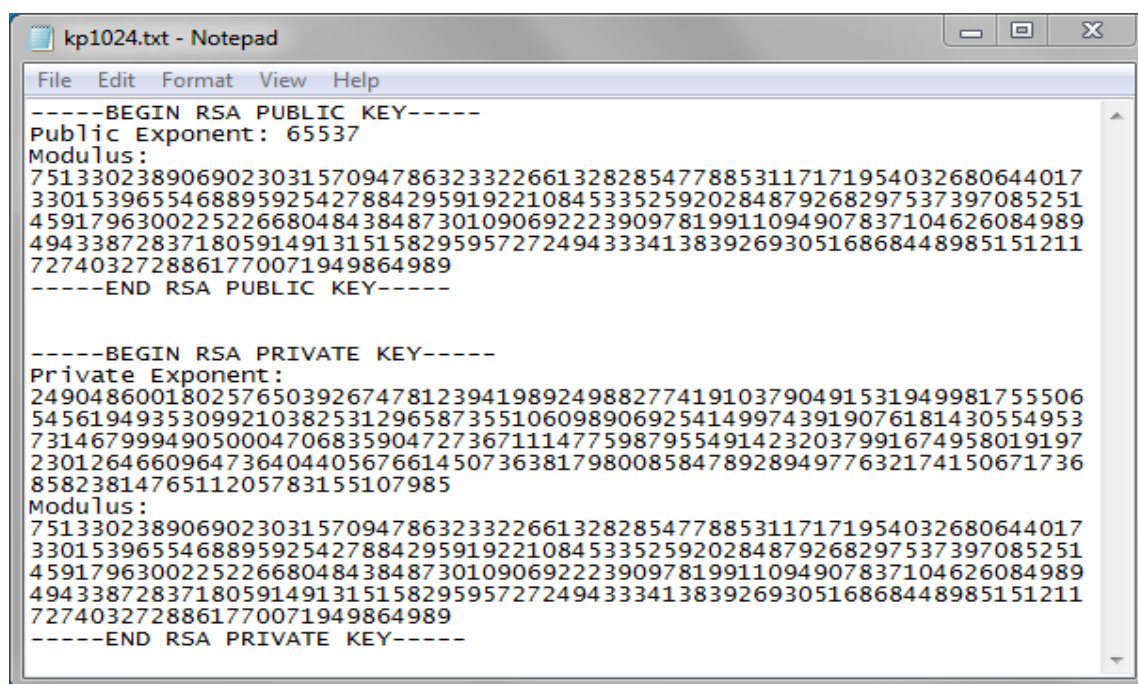
4.5 Exportação

Apesar de não visar a aplicação prática da solução no estado atual de desenvolvimento e maturidade, o GenCryptoKey foi desde o início concebido para simular o cenário real, isto é, provendo funcionalidades que estariam presentes no produto final.

Em outras palavras, isso quer dizer que de nada serviria uma confiável geração de chaves criptograficamente robustas sem que se pudesse fazer uso delas. Portanto, existindo ao menos um par de chaves que foi gerado e armazenado no banco de dados, o usuário é capaz de solicitar a exportação daquele para um arquivo externo à aplicação.

O arquivo criado contém os dados das chaves RSA necessários para o uso na prática: o módulo e os expoentes público e privado, como pode ser observado na Figura 11.

Idealmente, os dados do arquivo deveriam ser formatados de alguma forma que não expusesse os números que compõem os elementos das chaves, preferencialmente usando-se o padrão *Privacy Enhanced Email* (PEM), o qual é definido nas *Request For Comment* (RFC) 1421 a 1424, e mesmo sendo mais comumente usado por *softwares* de código aberto (“*free software*”) é amplamente reconhecido e aceito em toda a comunidade. Porém, considerando que exportar arquivos nesse formato requer uma tradução dos *bytes* para uma notação específica (a saber, *Abstract Syntax Notation One* – ASN.1), o que viria a requerer um estudo não prioritário para escopo do TCC, essa providência não foi tomada, mas será contemplada nas futuras melhorias ao GenCryptoKey.



```

kp1024.txt - Notepad
File Edit Format View Help
-----BEGIN RSA PUBLIC KEY-----
Public Exponent: 65537
Modulus:
7513302389069023031570947863233226613282854778853117171954032680644017
3301539655468895925427884295919221084533525920284879268297537397085251
4591796300225226680484384873010906922239097819911094907837104626084989
4943387283718059149131515829595727249433341383926930516868448985151211
7274032728861770071949864989
-----END RSA PUBLIC KEY-----

-----BEGIN RSA PRIVATE KEY-----
Private Exponent:
2490486001802576503926747812394198924988277419103790491531949981755506
5456194935309921038253129658735510609890692541499743919076181430554953
7314679994905000470683590472736711147759879554914232037991674958019197
2301264660964736404405676614507363817980085847892894977632174150671736
8582381476511205783155107985
Modulus:
7513302389069023031570947863233226613282854778853117171954032680644017
3301539655468895925427884295919221084533525920284879268297537397085251
4591796300225226680484384873010906922239097819911094907837104626084989
4943387283718059149131515829595727249433341383926930516868448985151211
7274032728861770071949864989
-----END RSA PRIVATE KEY-----

```

Figura 11. Exemplo de chave exportada para arquivo

5 AVALIAÇÃO E RESULTADOS

Este capítulo apresenta as informações quanto à avaliação sob a qual foi submetido o artefato de *software* desenvolvido no TCC e a análise dos resultados desta confrontada com o objetivo proposto para o Trabalho: a geração de chaves criptográficas robustas.

5.1 Avaliação

Como já foi explicado anteriormente, produzir uma chave criptográfica robusta não é uma tarefa trivial. Mais ainda, a própria definição para robustez nesse contexto é constantemente discutida e aprimorada.

Seria impróprio, além de ineficiente, tentar-se provar que um par de chaves gerado pela aplicação é robusto através de força bruta. Tampouco teria sentido confiar a avaliação no julgamento dado por uma pessoa sobre apenas a aparência das chaves (KNUTH, 1998), por mais especialista e experiente que essa pessoa ser na área de criptografia. Uma chave criptográfica, no seu âmago, nada mais é do que uma longa sequência binária, portanto, computadores são capazes de avaliar muitas mais em muito menos tempo, independentemente da característica que se deseja avaliar.

Dada essa conjuntura, o mais apropriado é usar testes estatísticos capazes de extrair informações relevantes das sequências binárias que analisam, tais como a independência entre os *bits*, a proporção de distribuição, entre outras, de modo a se poder extrair conclusões quantitativas e objetivas sobre a qualidade de uma determinada sequência.

5.2 Testes Estatísticos

“Aleatoriedade é uma propriedade probabilística, isto é, as propriedades de uma sequência aleatória podem ser caracterizadas e descritas em termos de probabilidade” (NIST, 2010) e tal propriedade é avaliada por um teste estatístico.

Um teste estatístico, por sua vez, busca a presença ou ausência de um determinado padrão em uma sequência qualquer, que se detectado, poderia indicar a não aleatoriedade da mesma. Como não existe, entretanto, um conjunto determinado de padrões, existem infinitos possíveis testes estatísticos.

Os conceitos que envolvem testes estatísticos são apresentados e exemplificados detalhadamente nos trabalhos de Kenny (2005) e do NIST (2010). As seções a seguir apresentam pontos fundamentais a respeito de cada um deles, necessários para o entendimento mínimo, os quais foram aplicados tanto na avaliação do TCC quanto no cálculo de *fitness* do AG.

5.2.1 Hipótese Nula e Hipótese Alternativa

Cada teste estatístico é formulado para testar uma “hipótese nula” (H_0) específica, contraposta por uma “hipótese alternativa” (H_a). As hipóteses nulas, no caso dos testes para uniformidade e independência, por exemplo, usados no cálculo de *fitness* dos indivíduos do AG, afirmam: os números de uma sequência binária são distribuídos uniformemente no intervalo $[0,1]$ e que os números da sequência são independentes uns dos outros. Em contrapartida, as hipóteses alternativas afirmam: os números não são distribuídos uniformemente no intervalo $[0,1]$ e os números não são independentes uns dos outros, respectivamente.

Uma vez que para aceitar-se a hipótese nula seria necessária uma sequência de tamanho infinito, é dito que uma sequência qualquer é bem sucedida em um teste estatístico quando não é possível rejeitar a hipótese nula (para facilitar, pode-se dizer que a H_0 é “verdadeira”), isto é, por exemplo, a não detecção de evidências que comprovem a dependência entre os números, no caso dos testes de independência. Em contrapartida, se é detectada qualquer evidência capaz de rejeitar a hipótese nula, então a hipótese alternativa é tomada como verdadeira e conclui-se que a sequência falhou no teste.

5.2.2 Nível de Significância

Em alguns casos, é possível que se julgue uma sequência aleatória como sendo não aleatória, com base no resultado de um teste estatístico. E o

inverso também é verdade. Esses casos são conhecidos como erro tipo I e erro tipo II, respectivamente.

O erro tipo I é comumente chamado de nível de significância, denotado por α , o qual geralmente é estabelecido antes do teste. O α então passa a ser entendido como a probabilidade de se concluir verdadeira a H_a para uma sequência que outrora seria bem sucedida no teste, ou seja, a probabilidade de se cometer o erro tipo I. Por outro lado, o erro tipo II é denotado por β .

O Quadro 3 apresenta como esses erros podem aparecer na conclusão de um teste estatístico.

Quadro 3. Possíveis erros na conclusão de um teste estatístico.

Situação Real	Conclusão	
	Não rejeitar H_0	Aceitar H_A
Sequência é aleatória (H_0 é “verdadeira”)	Correto	Erro tipo I
Sequência não é aleatória (H_a é verdadeira)	Erro tipo II	Correto

Percebe-se, portanto, que a qualidade de um teste está na probabilidade de um erro tipo II não ocorrer, pois esse é muito mais prejudicial à aplicação final da sequência avaliada: um esquema criptográfico, por exemplo, como é o caso no contexto apresentado nessa monografia.

O nível de significância escolhido para os testes estatísticos contemplados no cálculo de *fitness* do AG foi de 5%, enquanto que para a avaliação das chaves geradas pelo GenCryptoKey foi escolhido 1%, de modo a se fazer uma avaliação mais rigorosa. No contexto de criptografia, 1% e 5% são valores tipicamente escolhidos para α .

5.2.3 *P-value*

Cada teste é baseado no cálculo de um valor estatístico, que é função da sequência avaliada. Esse valor estatístico – o Qui-quadrado, no caso de todos os testes usados na avaliação – é usado para se calcular um atributo, conhecido como *p-value*, que exprime a força da evidência observada na sequência avaliada contra a hipótese nula. Em outras palavras, cada *p-value* é a probabilidade com a qual um RNG perfeito produziria uma sequência menos aleatória que aquela testada, dado o tipo de não aleatoriedade julgado pelo teste. Portanto, se o *p-value* calculado para um teste é 1, então a sequência aparenta ser perfeitamente aleatória, enquanto que 0 indica que a sequência aparenta ser completamente não aleatória NIST (2010).

5.2.4 Valor Crítico

Para cada estatística (Qui-quadrado, Kolmogorov-Smirnov, entre outros), uma distribuição teórica de referência sob a hipótese nula a ser testada é calculada usando-se métodos matemáticos. A partir dessa distribuição de referência é determinado um valor crítico, com o qual deve ser comparado o valor estatístico obtido durante o teste de cada sequência.

Se o valor calculado no teste exceder o valor crítico, a hipótese nula deve ser rejeitada (a sequência não é aleatória e, portanto, a hipótese alternativa deve ser aceita), caso contrário, a hipótese nula não é rejeitada, o que indica o sucesso da sequência no teste.

Os valores críticos dos testes Kolmogorov-Smirnov e Qui-quadrado de Pearson são apresentados nas tabelas dos Anexos A e B, respectivamente.

5.3 Suíte de Testes

Como já explicado anteriormente, existem infinitos testes estatísticos possíveis. Por essa razão, nenhum conjunto de testes pode ser dito “completo”. De qualquer maneira, existem algumas suítes de testes que são tomadas como referência para o teste de aleatoriedade em aplicações comerciais reais.

Abaixo são revistas as mais comuns e bem conceituadas.

- Knuth: apesar de antigo, Knuth (2005) ainda é a referência mais citada para o teste estatístico de RNGs. O conjunto de testes proposto por ele foi, por muito tempo, o padrão *de facto*. Apesar de ser a literatura fundamental para esse contexto, está desatualizado, pois os testes do seu conjunto são atualmente considerados fracos para aplicações de criptografia.
- Diehard: o conjunto de testes proposto por George Marsaglia inclui testes mais rigorosos do que os clássicos propostos por Knuth. Foi publicado em 1995 e até os dias de hoje é considerado um dos mais amplos para a detecção de não aleatoriedade, além de ser usado por diversas aplicações comerciais. Marsaglia, no entanto, faleceu em 2011 e, até onde se sabe, ninguém assumiu a continuidade do desenvolvimento do conjunto.
- Crypt-X: é uma suíte de testes estatísticos desenvolvida por pesquisadores do Centro de Pesquisa em Segurança da Informação da Universidade de Tecnologia de Queensland, na Austrália, mas é distribuído como um *software* de uso comercial. Os testes dessa suíte são aplicados de acordo com o tipo de algoritmo que está sendo testado, portanto, possivelmente seria uma suíte interessante para avaliar as sequências geradas especificamente por um AG. No entanto, o valor da licença para uso acadêmico é, relativamente, proibitivo: €350,00.
- National Institute of Standards and Technology (NIST): desenvolvida ao longo de 4 anos e lançada em 2001, é, atualmente, a suíte padrão no mundo de testes de RNGs. É composta por 15 testes – clássicos da literatura somados a novos – que foram desenvolvidos para testar aleatoriedade de sequências de tamanhos arbitrariamente longos, produzidas por *software* ou *hardware*.

Dentre as suítes revisadas, o autor optou pela suíte de testes estatísticos do NIST, por compreender um rigoroso conjunto de testes,

ser bem conceituada e usada por aplicações comerciais (o serviço Random.org é um exemplo), além de ser disponibilizada gratuitamente no *site* do instituto.

5.4 Testes Escolhidos

Dentre os 15 testes contemplados pela suíte do NIST, foram escolhidos 3 para realizar a avaliação do GenCryptoKey.

a. *Runs*

O foco desse teste é o número total de 0s e 1s presentes na sequência sob avaliação. Uma *run*, ou “corrida”, é uma subsequência ininterrupta de *bits* de um mesmo valor (0 ou 1), sendo também de mesmo valor os *bits* imediatamente antes e imediatamente após a sequência.

O propósito é determinar se o número de *runs* de 0s e 1s, de diversos tamanhos cada, presentes na sequência, está de acordo com o esperado teoricamente. Em outras palavras, mede-se a oscilação entre tais sub-sequências é muito rápida ou muito devagar.

b. *Serial*

Esse teste foca a frequência de cada padrão possível de m -bits na sequência, com o propósito de determinar se o número de ocorrências dos 2^m padrões de m -bits é aproximadamente o mesmo ao que seria esperado para uma sequência aleatória. Os padrões podem se sobrepor na sequência e o valor de m é escolhido pelo usuário. No caso, o valor escolhido para m foi 8, de tal forma que 2^8 padrões de 8 *bits* foram levados em consideração.

c. *Linear Complexity*

Os dois primeiros testes são clássicos do mundo de testes estatísticos, descritos por Knuth (1998), e bastante convencionais. Este, por sua vez, é mais novo e bastante rigoroso, além de, inclusive, ser o que demanda processamento e, conseqüentemente, mais tempo para ser executado.

O foco é testar o tamanho do menor *Linear Feedback Shift-Register* (LFSR) capaz de gerar a sequência sob avaliação, com o propósito de determinar se a sequência é suficientemente complexa para ser considerada aleatória. Sequências aleatórias são caracterizadas por LFSRs longos, enquanto que LSFRs curtos indicam não aleatoriedade (NIST, 2010).

Apesar de possuir falhas, como mostrado por Hamano, Sato e Yamamoto (2009), é o único teste na suíte capaz de testar a dificuldade de previsão da sequência sob análise, portanto continua sendo fundamental.

5.5 Análise dos Resultados

A avaliação do TCC foi realizada sobre um conjunto de 20 sequências de 1048 *bits*, conforme orientação do professor orientador. Os parâmetros usados pelo AG para a geração dessas sequências são apresentados na Tabela 1.

Tabela 1. Parâmetros do AG para testes de avaliação

Parâmetro	Valor
Tamanho do indivíduo (em <i>bits</i>)	1024
Tamanho da população inicial	500
Número de pontos de <i>crossover</i>	1
Número de mutações por indivíduo	1
Taxa de mutação	1,5%
Taxa de indivíduos para <i>crossover</i>	50%
Tamanho máximo da população (por geração)	50
Número de gerações para parada	20

De acordo com as recomendações do NIST (2011), esquemas com chaves de tamanho menor que 2048 *bits* passaram a ser obsoletos a partir de 2013. Entretanto, escolheu-se 1024 e os outros valores apresentados na Tabela

1, primordialmente, pela implicância que têm no tempo necessário para se executar o AG de modo a se gerar 20 indivíduos. Outras configurações, com maior tamanho de indivíduo, mais taxa de *crossover* e maior número de gerações possivelmente agregariam mais robustez às sequências geradas, mas também implicariam em um tempo muito maior para geração das sequências. Uma vez que a aplicação provê configuração de parâmetros, os valores escolhidos para avaliação não têm impacto para o usuário final.

Outro fator que poderia aumentar ainda mais a relevância do resultado da avaliação é o número de sequências avaliadas, porém o valor fixado em 20 também se deu pela questão de tempo. Os valores escolhidos, no entanto, não causam irrelevância ao resultado da avaliação, que é apresentado na Tabela 2.

Tabela 2. Resultados consolidados da avaliação do TCC

Teste Estatístico	<i>P-value</i>	Proporção de sucesso
<i>Runs</i>	0.834308	20/20
<i>Serial</i>	0.213309	20/20
<i>Serial</i>	0.834308	20/20
<i>Linear Complexity</i>	0.017912	18/20

Conforme pode ser observado na Tabela 2, todas as sequências obtiveram sucesso nos testes *Runs* e *Serial*. No teste *Linear Complexity*, porém, duas das sequências avaliadas fracassaram, isto é, possuem alguma evidência que indicou não aleatoriedade, segundo o critério do teste.

Entretanto, os valores de proporção de sucesso estão todos iguais ou acima do mínimo esperado para uma amostra de tamanho 20, que é de 18. Isso, portanto, é suficiente para caracterizar o sucesso do *software* na busca pelo objetivo de gerar chaves criptográficas robustas.

O Apêndice F apresenta os resultados brutos da avaliação os quais serviram de base para a composição da Tabela 2.

6 CONCLUSÃO

Este documento apresenta em detalhes um artefato computacional desenvolvido ao longo de, aproximadamente, nove meses, como parte essencial do TCC, além dos conceitos que estudados que envolveram a sua criação.

O *software* em questão, denominado GenCryptoKey, foi desenvolvido com a intenção de alcançar o objetivo proposto pelo autor como motivo do TCC: gerar chaves criptográficas robustas. De acordo com os resultados de testes estatísticos convencionais na área de criptografia, foi possível concluir que o GenCryptoKey obteve sucesso e alcançou o objetivo esperado, conforme explicado na seção 6.4.

Deve-se ressaltar também que além de alcançar o objetivo proposto, o GenCryptoKey incorpora também uma vantagem muitas vezes esquecida ou ignorada por outros artefatos computacionais e que, especialmente no contexto ao qual se aplica, é fundamental: alto grau de configuração, por meio de parâmetros.

Os resultados da avaliação de robustez das chaves criptográficas geradas, no entanto, não foram os únicos. Por meio da experiência obtida no decorrer da execução do TCC, o autor teve a oportunidade de aprender e desenvolver conceitos, métodos e uso de ferramentas que não foram abordadas no curso de Engenharia de Computação, além de colocar em prática muitos daqueles que já haviam sido estudados (e, conseqüentemente, facilitaram o desenvolvimento do TCC). Além disso, proporcionou que o autor gerenciasse individualmente todo o processo de desenvolvimento e aprendesse a superar as dificuldades, desde técnicas até outras como de organização pessoal e prazos. Todos esses fatores, por fim, agregaram enorme valor à vida acadêmica e profissional do autor.

A seguir são ressaltados o grau de complexidade do *software* desenvolvido, os aspectos de qualidade por ele atingidos, as dificuldades encontradas ao longo do desenvolvimento e as propostas de melhorias futuras.

6.1 Grau de Complexidade

Esta seção tem o intuito de apresentar o grau de complexidade do algoritmo que é executado pelo GenCryptoKey quando da geração de chaves criptográficas, a partir dos conceitos de análise de complexidade de algoritmos (TOSCANI; VELOSO, 2012).

O grau de complexidade de um algoritmo é sintetizado pelo grau de complexidade particular da rotina que demanda maior quantidade de processamento dentre aquelas que compõem o algoritmo. Assim sendo, afirma-se que o grau de complexidade do GenCryptoKey pode ser extraído da análise da rotina responsável pelo método *crossover* do AG executado pelo artefato.

A rotina de *crossover* toma como entrada um vetor contendo os indivíduos que constituem a população em evolução em uma determinada geração. A partir desse vetor, o número de indivíduos que serão considerados pais (sofreram cruzamento para geração de novos indivíduos), chamados de indivíduos pais, é calculado multiplicando-se o tamanho da população por uma taxa que representa a porcentagem da população que passará pela operação *crossover*, variável entre 2 e 100%, estipulada pelo usuário.

Após isso, a rotina então realiza execução de dois laços encadeados sobre esse número calculado de indivíduos pais; o laço externo realiza uma iteração a menos do que o número de indivíduos pais, para que sejam evitados cruzamentos repetidos, isto é, dois cruzamentos entre os mesmos pais.

Dado esse cenário, pode-se dizer que o número máximo de iterações a serem realizadas nessa rotina é de $\frac{n^2}{2}$, limitado superiormente pela taxa de indivíduos pais (quando for fixada em 1).

Entretanto, a rotina de *crossover* é executada em cada uma das gerações pelas quais passa a população até que se chegue ao fim da evolução, que é determinado pelo número de gerações que o usuário estipula para o AG. Por essa razão, o número máximo de iterações da rotina de *crossover* passa então a ser de $\frac{k \times n^2}{2}$. A Figura 12 apresenta o pseudocódigo da rotina explicada.

```

/* Inicializa vetor população */

vetorPopulação = nova população;

/* Repete pelo número de gerações definidas pelo usuário */

i = 0;
Faça enquanto (i < número total de gerações) {

    /* Operação crossover */

    j = 0;
    k = j+1;

    Faça enquanto (j < tamanho da população) {

        Faça enquanto (k < tamanho da população) {

            /* Cruza genes dos indivíduos pais para gerar novos indivíduos */
            novoIndivíduo_1 = vetorPopulação[j].gene1 + vetorPopulação[k].gene2
            novoIndivíduo_2 = vetorPopulação[k].gene1 + vetorPopulação[j].gene2
        }

    }

}

```

Figura 12. Pseudocódigo da rotina *crossover*

Como valores constantes não são considerados em análises de complexidade, conclui-se, portanto, que o grau de complexidade do *software* é $O(k \times n^2)$. A Figura 13 apresenta o gráfico de tempo de execução do algoritmo em função da variação do tamanho de chave desejado.

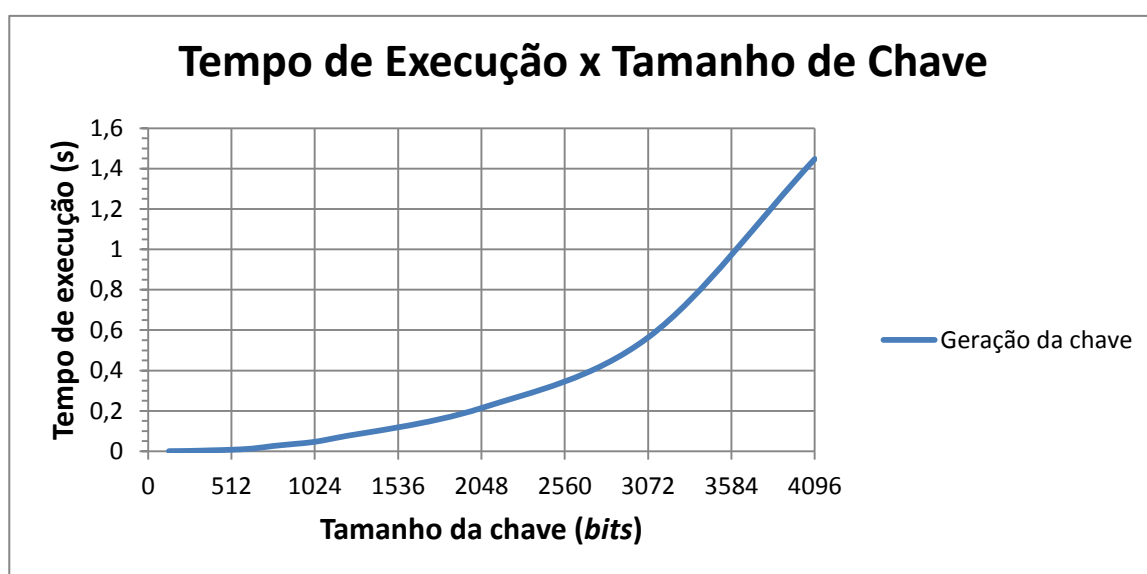


Figura 13. Tempo de execução do AG em função do tamanho de chave

6.2 Dificuldades Encontradas

Por conta de muitos dos conceitos que envolvem o tema do TCC terem sido inéditos para o autor durante o seu desenvolvimento, algumas dificuldades apareceram, muitas provindas, inclusive, dos aspectos de aprimoramento, como já era previsto. Essas dificuldades provocaram o maior consumo de tempo nas atividades previstas para os *sprints*, chegando até mesmo a extrapolar o tempo previsto para a maioria deles. Essa seção relata as principais dificuldades e as soluções adotadas para superá-las.

6.2.1 Backup e Versionamento de Código

A primeira dificuldade encontrada no TCC apareceu logo no momento de preparação do ambiente de desenvolvimento, inserido na *sprint* 0, com a configuração da políticas de *backup*.

O autor optou por ferramentas nativas do sistema operacional que possibilitassem a execução do *backup* de forma automática e em *background*, no entanto, não tinha conhecimento das ferramentas (*rsync* e *Crontab*) e tampouco da linguagem usada para escrever a rotina da ação (*Bash script*). Por conta disso, foi necessário estudo e diversas experimentações incrementais até que se chegasse ao resultado esperado.

Também inseridas no contexto de preparação do ambiente de desenvolvimento estavam a escolha e a configuração da ferramenta de versionamento de código. Como não se tinha experiência suficiente com a ferramenta escolhida (*Git*), também houve dificuldade com a configuração do mesmo.

Ambas essas dificuldades contribuíram para a extrapolação do tempo previsto para a *sprint* 0, que teve essas atividades finalizadas em horas extras, paralelamente ao início da *sprint* 1.

6.2.2 Entendimento e Implementação do Algoritmo Genético

A principal dificuldade enfrentada no início do TCC foi a compreensão do funcionamento de um AG, pois isso seria essencial para o correto desenvolvimento do artefato de *software*.

Porém, através da leitura de diversos artigos sobre o tema e frequentes discussões com o orientador, o autor foi capaz de entender na sua plenitude o funcionamento esperado de um AG. Os encontros com o coorientador também foram de extrema importância para a sua superação dessa dificuldade.

6.2.3 Testes Estatísticos

A maior dificuldade do TCC esteve na compreensão dos testes estatísticos que são necessários para se avaliar sequências binárias de acordo com um conceito de aleatoriedade.

A maioria dos testes empregados depende de equações matemáticas não triviais que, ao mesmo tempo, são essenciais para os resultados dos mesmos e, portanto, necessitaram ser implementadas para a etapa de cálculo de *fitness* dos indivíduos que passam pelo AG.

Além disso, os materiais de referência encontrados estavam, na grande maioria, escritos com linguagem técnico e não eram agradavelmente didáticos.

Essa dificuldade na compreensão dos testes causou também um atraso na definição da composição do atributo *fitness*, o que, por sua vez, causou um enorme atraso no cronograma de atividades e, conseqüentemente, fez com que o autor necessitasse dedicar inúmeras horas extras não previstas para compensar o atraso.

6.2.4 Integração com a Suíte de Testes

Não sendo prudente implementar os testes estatísticos, escolhidos para a avaliação do TCC, sem possuir vasto conhecimento dos conceitos inerentes aos mesmos, houve a necessidade de integrar o GenCryptoKey com uma suíte de testes (no caso, do NIST), para que a avaliação pudesse realizada de forma automática.

Essa integração, no entanto, não foi trivial pois exigiu uso de alguns recursos da linguagem de programação que não eram conhecidos do autor, portanto diversas horas foram gastas até que se conseguisse plenamente realizar a comunicação desejada entre as duas aplicações.

A superação dessa dificuldade só foi possível com a ajuda da comunidade de desenvolvedores do fórum StackOverflow (www.stackoverflow.com), que apontaram erros, sugestões e fizeram correções e melhorias no código desenvolvido pelo autor para a dita integração.

6.3 Qualidade de Software

De acordo com a norma ISO/IEC 9126-2001/NBR 13596 (ABNT, 2003), um produto de *software* tem sua qualidade medida através de seis aspectos principais:

- Funcionalidade: a capacidade do produto de *software* de prover as funcionalidades que satisfazem o usuário;
- Confiabilidade: indica se o produto é confiável no sentido de manter o nível de desempenho dele esperado, dentro das condições estabelecidas;
- Usabilidade: a capacidade do produto de ser compreendido, ter funcionalidades aprendidas, ser operado com facilidade e ainda ser atraente ao usuário;
- Eficiência: indica se o tempo de resposta do *software* e os recursos por ele usados estão dentro das especificações e são compatíveis com o nível de desempenho;
- Manutenibilidade: a facilidade com que o produto de *software* pode ser modificado, seja para melhorias e extensão de funcionalidades ou para a correção de possíveis defeitos, falhas e erros;
- Portabilidade: a capacidade do produto de *software* de ser transferido de um ambiente para outro qualquer, levando-se em

consideração condições de infraestrutura, recursos de *hardware*, entre outros.

Com base nesses aspectos, pode-se avaliar o aplicativo GenCryptoKey.

O GenCryptoKey é capaz de realizar adequadamente e com acurácia tudo o que se propõe a fazer. Uma de suas principais características, muito cobrada pelo cliente do TCC, é a possibilidade de ampla configuração de parâmetros para o AG.

Além disso, o aplicativo não só restringe o acesso às funcionalidades por ele providas ao usuário autorizado como mantém um banco de dados próprio, o qual mantém os dados cifrados, de forma a proteger toda a informação gerada. Por essas razões, pode-se afirmar que o GenCryptoKey atinge integralmente o objetivo do aspecto funcionalidade.

O aspecto confiabilidade não foi considerado de grande relevância para o desenvolvimento do GenCryptoKey, especificamente porque a duração do TCC é muito curta para a criação de um produto de *software* e, conseqüentemente, o período de testes é mais curto ainda, de tal forma que não se pode dizer que o artefato desenvolvido é integralmente maduro. Apesar disso, o aplicativo é parcialmente tolerante a falhas e não deixa de funcionar mediante qualquer problema em tempo de execução. Portanto, considera-se que o GenCryptoKey contempla parcialmente o aspecto confiabilidade.

Justamente pelo fato de ter um objetivo muito claro e uma funcionalidade principal bastante óbvia, o GenCryptoKey é de aprendizagem e operação extremamente fáceis, essa última auxiliada por uma interface simples e enxuta. Assim sendo, pode-se dizer que o aplicativo está adequado quanto ao aspecto de usabilidade.

Ao passo que os outros aspectos analisados foram meras conseqüências de algumas boas escolhas e diligência no trabalho de desenvolvimento, o aspecto eficiência foi fortemente levado em consideração. Em todos os momentos do desenvolvimento foram pensadas e repensadas maneiras

de se otimizar o algoritmo, o que resultou em tempos de resposta bastante aceitáveis. Além disso, a execução do GenCryptoKey não utiliza uma quantidade de recursos da máquina grande ao ponto de incapacitar outras aplicações de serem executadas concorrentemente. Por isso, pode-se dizer que o GenCryptoKey satisfaz completamente o aspecto de eficiência.

No que diz respeito à manutenibilidade, o artefato de *software* não só foi desenvolvido seguindo-se boas práticas de programação e o bastante difundido padrão arquitetural MVC, como também teve seu código amplamente comentado, permitindo que sua manutenção seja fácil para qualquer pessoa que se disponha a fazê-la, mesmo sem conhecimento prévio. Portanto, afirma-se que o GenCryptoKey cumpre com as diretrizes do aspecto de manutenibilidade.

Por fim, o GenCryptoKey é uma aplicação autocontida, desenvolvida na linguagem Java, o que permite sua execução em diversas plataformas, mediante apenas a instalação da máquina virtual Java e seu núcleo de bibliotecas - conjunto conhecido como Java *Runtime* – de tal maneira que pode-se afirmar que a aplicação é extremamente portátil e, portanto, satisfaz integralmente o aspecto de portabilidade.

Em suma, o GenCryptoKey enquadra-se integralmente em 5 dos 6 aspectos de qualidade e parcialmente em 1, sendo eles, respectivamente, funcionalidade, usabilidade, eficiência, manutenibilidade, portabilidade e confiabilidade.

6.4 Propostas para Futuras Melhorias

Infelizmente, o tempo disponível para o desenvolvimento do artefato de *software* do TCC não foi suficiente para a inclusão de algumas funcionalidades e melhorias que foram identificadas ao longo do processo de desenvolvimento e que agregariam mais valor ao produto final. Essas são descritas a seguir, como propostas de melhorias a serem futuramente incorporadas à aplicação.

- Perfis de configurações: como o GenCryptoKey pode vir a ser usado para a geração de chaves que serão aplicadas em cenários

cotidianos distintos, seria interessante a possibilidade do usuário armazenar um ou mais perfis de configuração dos parâmetros disponíveis, de tal maneira que não fosse necessário ajustá-los manualmente cada vez que se desejasse, por exemplo, gerar chaves de tamanhos diferentes;

- Interface de acompanhamento gráfico do AG: para facilitar o processo de correção de *bugs*, por exemplo, pensou-se no desenvolvimento de uma interface gráfica interativa que permitisse o acompanhamento das operações do AG em tempo real. Essa interface não foi desenvolvida no âmbito do TCC por causa da complexidade decorrente da mesma e pelo fato de não contribuir diretamente para o objetivo principal do trabalho;
- Exportação de chave em padrão convencional: essa é possivelmente a melhoria mais relevante, pois colocaria o GenCryptoKey em posição capaz de competir com outros produtos de *software* para a geração de chaves criptográficas. Atualmente, a exportação das chaves geradas para um arquivo passa as informações “em claro”, isto é, sem nenhuma codificação e com formatação diferente do padrão convencional, que determina que o arquivo seja do tipo PEM, ou, em outras palavras, tenha os dados primeiramente escritos de acordo com a notação sintática ASN.1 e então traduzidos para base 64. Essa característica não foi implementada dada a complexidade inerente ao padrão, que necessita um estudo específico para sua correta implementação;
- Multi-threading para geração de chaves: a execução da rotina de geração de chaves através de múltiplas *threads* poderia acelerar o tempo de execução além de permitir a geração de múltiplas chaves concorrentemente, agregando assim mais eficiência e usabilidade ao GenCryptoKey;
- Exportação do banco de dados da aplicação: uma funcionalidade desse tipo daria liberdade ao usuário de armazenar suas chaves

criptográficas no repositório que lhe fosse mais conveniente, e não somente no banco de dados contido na aplicação.

- Encapsulamento dos testes estatísticos: possivelmente a melhoria mais complexa, pois a correta implementação dos testes estatísticos usados para avaliação requerem um alto grau de detalhe de conhecimento dos mesmos. Por outro lado, seria de grande valia, pois permitiria que o a qualidade das chaves criptográficas geradas pelo *software* fosse reavaliada a qualquer momento de mudança / melhoria da rotina de geração sem a dependência da suíte de testes estatísticos do NIST.

REFERÊNCIAS

AGRAWAL, Manindra; KAYAL, Neeraj; SAXENA, Nitin. *PRIMES is in P*. Annals Of Mathematics of the Princeton University & Institute for Advanced Study, Iss. 2, vol. 160, p. 781-793. Mar. 2003.

BENNET, Charles H.; BERNSTEIN, Ethan; BRASSARD, Gilles; VAZIRANI, Umesh. *Strengths and Weaknesses of Quantum Computing*, SIAM Journal on Computing (SICOMP), p. 1510-1523. Dec. 1996.

BIN PRESS. *Learn Objective-C, Design Patterns: Model-View-Controller*, Aug. 2011. Available at: <<http://www.binpress.com/tutorial/learn-objectivec-design-patterns-modelviewcontroller/87>>. Cited 11 nov. 2014.

BONEH, Dan. *Twenty Years of Attacks on the RSA Cryptosystem*, Notices of the American Mathematical Society (AMS), vol. 46, p.203-213, 1999.

HAMANO, Kenji; SATO, Fumio; YAMAMOTO, Hirosuke. *A new Randomness Test Based on Linear Complexity Profile*, The Institute of Electronics, Information and Communication Engineers (IEICE) Transactions on Fundamentals, vol. E92-A, p. 166-172, Jan. 2009.

JAVADOC. *Java API*. 2014. Available at: <<https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html>>. Cited 21 sep. 2014.

JHAJHARIA, Smita; MISHRA, Swati; BALI, Siddharth. *Public Key Cryptography Using Particle Swarm Optimization and Genetic Algorithms*, International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), p. 832-839. Jun 2013.

JHAJHARIA, Smita; MISHRA, Swati; BALI, Siddharth. *Public Key Cryptography using Neural Networks and Genetic Algorithms*, Contemporary Computing (IC3), p. 137-142. Aug 2013.

KENNY, Charmaine. *Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators*, 2005, 107 p. Available at: <<http://www.random.org/analysis/Analysis2005.pdf>>. Cited 15 nov. 2014.

KNUTH, Donald E. *The Art Of Computer Programming*, Vol. 2, 3rd Ed., Addison-Wesley, 1998, 762p.

LEHMER, Derrick H. *An extended theory of Lucas' functions*. Annals Of Mathematics of the Princeton University & Institute for Advanced Study, 2nd ser., vol. 31, p. 419-448. Jul. 1930.

MISHRA, Swati; BALI, Siddharth. *Public Key Cryptography Using Genetic Algorithm*. International Journal of Recent Technology and Engineering (IJRTE), p. 150-154. May 2013.

MITCHELL, Melanie. *An Introduction to Genetic Algorithms*, 5th Ed., Cambridge: MIT Press, 1996. 158p.

NIST – National Institute of Standards and Technology. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, Apr. 2010.

NIST – National Institute of Standards and Technology. *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*, Jan. 2011.

PAAR, Christof; PELZL, Jan. *Understanding Cryptography: A Textbook for Students and Practitioners*, 1st Ed., Springer, 2010. 372p.

RABIN, Michael O. *Probabilistic Algorithm for Primality Testing*. Journal Of Number Theory, Iss. 1, vol. 12, p. 128-138. Feb. 1980

RIVEST, Ron L.; SHAMIR, Adi; ADLEMAN, Leonard. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM (CACM), p. 120-126. Feb. 1978.

SANTOS, Mauricio P. S. *Introdução à Simulação Discreta*, 1999, cap. 2, p. 31-46. Disponível em: <<https://www.scribd.com/doc/59087371/30/Testes-estatisticos-para-a-uniformidade>>. Acesso em: 11 nov. 2014.

SCHNEIER, Bruce. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd Ed., John Wiley & Sons, 1996, cap. 1, p. 24-26.

SCHWABER, Ken; SUTHERLAND, Jeff. *The Scrum Guide*, p. 3-16, July 2013. Available at: <<http://www.scrum.org>>. Cited 14 mar. 2014.

SHANNON, Claude E. *A Mathematical Theory of Communication*. The Bell System Technical Journal, Iss. 3, vol. 27, p.379-423. Jul. 1948

SHOR, Peter. *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*, Annual Symposium on Foundations Of Computer Science (FOCS), 35th, 1994, Santa Fe: IEEE, 1994. p.124-134.

TOSCANI, Laira V.; VELOSO, Paulo A. *Complexidade de algoritmos*. 3 Ed. Sagra-Luzzatto, 2012. 202p.

APÊNDICES

Apêndice A Diagrama de sequência descrito em texto para PlantUML

```

@startuml
title <b>Diagrama de Sequência</b> Algoritmo Genético do GenCryptoKey

EvoluirPopulação -> Crossover: PopulaçãoInicial

box "Algoritmo Genético" #LightBlue
    participant Crossover
    participant Mutação
    participant Fitness
    participant Seleção
end box

Crossover -> Mutação: Prole

Mutação -> Fitness: Prole
activate Fitness

Fitness -> Seleção: Prole

Fitness <- Seleção: IndivíduoMaisFit
deactivate Fitness

EvoluirPopulação <- Fitness: IndivíduoMaisFit

@enduml

```

Apêndice B Diagrama de sequência descrito em texto para PlantUML

```

@startuml
title <b>Diagrama de Atividade</b> Algoritmo Genético do GenCryptoKey

start

while(Tamanho da população inicial alcançado?) is (Não)
    :Gerar número provavelmente primo
    de tamanho definido pelo usuário
    para compor a população inicial;
endwhile (Sim)

while(Número de gerações definido pelo usuário foi alcançado?) is (Não)

while(Todos os cruzamentos foram realizados?) is (Não)
    ://Crossover//: seleciona número de pais definidos pelo usuário
    e realiza operação para geração de prole;
endwhile (Sim)

while(Chance the mutação foi verificada em todos os genes de todos os indivíduos?) is (Não)
    :Mutação: verifica chance de mutação. Se positivo, gerar um
    novo número provavelmente primo e substituir o gene;
endwhile (Sim)

partition Cálculo_//Fitness// _indivíduos {
    partition Testes_para_uniformidade {
        :Aplicar testes de //gaps// ;
        :Aplicar teste Kolmogorov-Smirnov sobre
        o result do teste de //gaps//;
    }
    partition Testes_para_frequência {
        :Calcular entropia esperada;
        :Calcular entropia de Shannon do indivíduo;
        :Calcular teste Qui-quadrado de Pearson sobre a
        Entropia de Shannon resultante;
    }

    :Atribuir valor de fitness do individuo a partir da
    Norma Euclidiana entre os testes KS e Qui-quadrado;
}

:Seleção: selecionar o grupo de indivíduos com maior //fitness //
de acordo com o tamanho de grupo definido pelo usuário para
evoluir para a próxima geração;

endwhile (Sim)
:Selecionar o indivíduo com maior //fitness//;

stop
@enduml

```

Apêndice C Ferramentas para tarefas de *Backup*

Software	Versão	Disponível em
rsync	3.0.9	Sistemas Unix- <i>like</i>
Crontab	-	Sistemas Unix- <i>like</i>
Dropbox	2.6.31	Multiplataforma

Apêndice D *Bash Script* para *backup* com o *rsync*

```
#!/bin/bash

# Snapshots diários em diretórios do tipo
# "daily-4-Thu", "daily-5-Fri", e assim por diante.
if [[ "$1" == "daily" ]]
then
    path=daily-`date +%u-%a`
fi

# Snapshots semanais em diretórios do tipo
# "weekly-1", onde 1 é o dia do mês
if [[ "$1" == "weekly" ]]
then
    path=weekly-`date +%d`
fi

# Snapshots mensais em diretórios do tipo "monthly-04-Apr"
if [[ "$1" == "monthly" ]]
then
    path=monthly-`date +%m-%b`
fi

# Executa o script com o comando "go" como segundo parâmetro
# para executar o rsync, caso contrário imprime o comando que
# teria sido executado.
# -a, --archive : archive (resumo de -rlptgoD, que usa recursão
#                  e preserva quase tudo)
# -v, --verbose : verbosity (mais informação nos logs)
# -z, --compress : compressão de dados
# --delete : remove os arquivos presentes no diretório de destino
#             que não estão presentes no diretório fonte

if [[ "$2" == "go" ]]
then
    rsync -avz --delete /home/nicholas/TCC /home/nicholas/Dropbox/TCC_backups/$path
else
    echo rsync -avz --delete /home/nicholas/TCC /home/nicholas/Dropbox/TCC_backups/$path
fi
```

Apêndice E Entradas no Crontab do sistema

```
40 21 * * * /home/nicholas/TCC/rsync_backup_script.sh daily go
00 22 * 7 * /home/nicholas/TCC/rsync_backup_script.sh weekly go
15 22 30 * * /home/nicholas/TCC/rsync_backup_script.sh monthly go
```

Apêndice F Relatórios de resultados finais de avaliação

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is </home/nicholas/TCC/1024_bit_keys>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
4	2	2	2	2	2	3	2	1	0	0.834308	20/20	Runs
0	1	3	4	2	0	4	3	3	0	0.213309	20/20	Serial
1	2	1	3	3	0	3	2	2	3	0.834308	20/20	Serial
3	0	1	1	4	0	1	0	6	4	0.017912	18/20	LinearComplexity

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 18 for a sample size = 20 binary sequences.

For further guidelines construct a probability table using the MAPLE program provided in the addendum section of the documentation.

FILE = /home/nicholas/TCC/1024_bit_keys ALPHA = 0.0100

```

BITSREAD = 1024 0s = 482 1s = 542
BITSREAD = 1024 0s = 502 1s = 522
BITSREAD = 1024 0s = 512 1s = 512
BITSREAD = 1024 0s = 545 1s = 479
BITSREAD = 1024 0s = 521 1s = 503
BITSREAD = 1024 0s = 535 1s = 489
BITSREAD = 1024 0s = 525 1s = 499
BITSREAD = 1024 0s = 515 1s = 509
BITSREAD = 1024 0s = 523 1s = 501
BITSREAD = 1024 0s = 521 1s = 503
BITSREAD = 1024 0s = 527 1s = 497
BITSREAD = 1024 0s = 505 1s = 519
BITSREAD = 1024 0s = 486 1s = 538
BITSREAD = 1024 0s = 508 1s = 516
BITSREAD = 1024 0s = 496 1s = 528
BITSREAD = 1024 0s = 526 1s = 498
BITSREAD = 1024 0s = 534 1s = 490
BITSREAD = 1024 0s = 516 1s = 508
BITSREAD = 1024 0s = 502 1s = 522
BITSREAD = 1024 0s = 515 1s = 509

```

ANEXOS

Anexo A Tabela de valores críticos do teste Kolmogorov-Smirnov

Kolmogorov–Smirnov Tables

Critical values, $d_{\alpha}(n)^a$, of the maximum absolute difference between sample $F_n(x)$ and population $F(x)$ cumulative distribution.

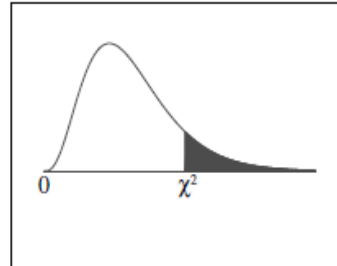
Number of trials, n	Level of significance, α			
	0.10	0.05	0.02	0.01
1	0.95000	0.97500	0.99000	0.99500
2	0.77639	0.84189	0.90000	0.92929
3	0.63604	0.70760	0.78456	0.82900
4	0.56522	0.62394	0.68887	0.73424
5	0.50945	0.56328	0.62718	0.66853
6	0.46799	0.51926	0.57741	0.61661
7	0.43607	0.48342	0.53844	0.57581
8	0.40962	0.45427	0.50654	0.54179
9	0.38746	0.43001	0.47960	0.51332
10	0.36866	0.40925	0.45662	0.48893
11	0.35242	0.39122	0.43670	0.46770
12	0.33815	0.37543	0.41918	0.44905
13	0.32549	0.36143	0.40362	0.43247
14	0.31417	0.34890	0.38970	0.41762
15	0.30397	0.33760	0.37713	0.40420
16	0.29472	0.32733	0.36571	0.39201
17	0.28627	0.31796	0.35528	0.38086
18	0.27851	0.30936	0.34569	0.37062
19	0.27136	0.30143	0.33685	0.36117
20	0.26473	0.29408	0.32866	0.35241
21	0.25858	0.28724	0.32104	0.34427
22	0.25283	0.28087	0.31394	0.33666
23	0.24746	0.27490	0.30728	0.32954
24	0.24242	0.26931	0.30104	0.32286
25	0.23768	0.26404	0.29516	0.31657
26	0.23320	0.25907	0.28962	0.31064
27	0.22898	0.25438	0.28438	0.30502
28	0.22497	0.24993	0.27942	0.29971
29	0.22117	0.24571	0.27471	0.29466
30	0.21756	0.24170	0.27023	0.28987
31	0.21412	0.23788	0.26596	0.28530
32	0.21085	0.23424	0.26189	0.28094
33	0.20771	0.23076	0.25801	0.27677
34	0.20472	0.22743	0.25429	0.27279
35	0.20185	0.22425	0.26073	0.26897
36	0.19910	0.22119	0.24732	0.26532
37	0.19646	0.21826	0.24404	0.26180
38	0.19392	0.21544	0.24089	0.25843
39	0.19148	0.21273	0.23786	0.25518
40 ^b	0.18913	0.21012	0.23494	0.25205

^aValues of $d_{\alpha}(n)$ such that $p(\max)|F^n(x) - F(x)|d^{\alpha}(n) = \alpha$.

^b $N > 40 \approx \frac{1.22}{N^{1/2}}, \frac{1.36}{N^{1/2}}, \frac{1.51}{N^{1/2}}$ and $\frac{1.63}{N^{1/2}}$ for the four levels of significance.

Anexo B Tabela de valores críticos do teste Qui-quadrado de Pearson

Chi-Square Distribution Table



The shaded area is equal to α for $\chi^2 = \chi^2_{\alpha}$.

df	$\chi^2_{.995}$	$\chi^2_{.990}$	$\chi^2_{.975}$	$\chi^2_{.950}$	$\chi^2_{.900}$	$\chi^2_{.100}$	$\chi^2_{.050}$	$\chi^2_{.025}$	$\chi^2_{.010}$	$\chi^2_{.005}$
1	0.000	0.000	0.001	0.004	0.016	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	9.236	11.070	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	10.645	12.592	14.449	16.812	18.548
7	0.989	1.239	1.690	2.167	2.833	12.017	14.067	16.013	18.475	20.278
8	1.344	1.646	2.180	2.733	3.490	13.362	15.507	17.535	20.090	21.955
9	1.735	2.088	2.700	3.325	4.168	14.684	16.919	19.023	21.666	23.589
10	2.156	2.558	3.247	3.940	4.865	15.987	18.307	20.483	23.209	25.188
11	2.603	3.053	3.816	4.575	5.578	17.275	19.675	21.920	24.725	26.757
12	3.074	3.571	4.404	5.226	6.304	18.549	21.026	23.337	26.217	28.300
13	3.565	4.107	5.009	5.892	7.042	19.812	22.362	24.736	27.688	29.819
14	4.075	4.660	5.629	6.571	7.790	21.064	23.685	26.119	29.141	31.319
15	4.601	5.229	6.262	7.261	8.547	22.307	24.996	27.488	30.578	32.801
16	5.142	5.812	6.908	7.962	9.312	23.542	26.296	28.845	32.000	34.267
17	5.697	6.408	7.564	8.672	10.085	24.769	27.587	30.191	33.409	35.718
18	6.265	7.015	8.231	9.390	10.865	25.989	28.869	31.526	34.805	37.156
19	6.844	7.633	8.907	10.117	11.651	27.204	30.144	32.852	36.191	38.582
20	7.434	8.260	9.591	10.851	12.443	28.412	31.410	34.170	37.566	39.997
21	8.034	8.897	10.283	11.591	13.240	29.615	32.671	35.479	38.932	41.401
22	8.643	9.542	10.982	12.338	14.041	30.813	33.924	36.781	40.289	42.796
23	9.260	10.196	11.689	13.091	14.848	32.007	35.172	38.076	41.638	44.181
24	9.886	10.856	12.401	13.848	15.659	33.196	36.415	39.364	42.980	45.559
25	10.520	11.524	13.120	14.611	16.473	34.382	37.652	40.646	44.314	46.928
26	11.160	12.198	13.844	15.379	17.292	35.563	38.885	41.923	45.642	48.290
27	11.808	12.879	14.573	16.151	18.114	36.741	40.113	43.195	46.963	49.645
28	12.461	13.565	15.308	16.928	18.939	37.916	41.337	44.461	48.278	50.993
29	13.121	14.256	16.047	17.708	19.768	39.087	42.557	45.722	49.588	52.336
30	13.787	14.953	16.791	18.493	20.599	40.256	43.773	46.979	50.892	53.672
40	20.707	22.164	24.433	26.509	29.051	51.805	55.758	59.342	63.691	66.766
50	27.991	29.707	32.357	34.764	37.689	63.167	67.505	71.420	76.154	79.490
60	35.534	37.485	40.482	43.188	46.459	74.397	79.082	83.298	88.379	91.952
70	43.275	45.442	48.758	51.739	55.329	85.527	90.531	95.023	100.425	104.215
80	51.172	53.540	57.153	60.391	64.278	96.578	101.879	106.629	112.329	116.321
90	59.196	61.754	65.647	69.126	73.291	107.565	113.145	118.136	124.116	128.299
100	67.328	70.065	74.222	77.929	82.358	118.498	124.342	129.561	135.807	140.169