Impactos Computacionais de uma Implementação de Criptografia Visual

Giuliano Berlatto Marques, Vinicius Gadis Ribeiro, Jorge Rodolfo Zabadal

Curso de Ciência da Computação, Centro Universitário La Salle,
Avenida Victor Barreto, 2288 - 92001-001, Canoas, Rio Grande do Sul, Brasil
giuliano@gmail.com
Curso de Sistemas de Informação, Centro Universitário Ritter dos Reis,
Rua Orfanotrófio, 555 Alto Teresópolis - 90840-440, Porto Alegre, Rio Grande do Sul, Brasil
vinicius@uniritter.edu.br
Universidade Federal do Rio Grande do Sul, Escola de Engenharia
Av. Osvaldo Aranha, 99 – 4º andar – Centro, 90.460-900 Porto Alegre – RS – Brasil
jorge.zabadal@ufrgs.br

Resumo

Este artigo realiza breve estudo sobre a Criptografia Visual, vindo a implementar um protótipo para fins de identificar características de processamento quando do emprego dessa tecnologia. Foi avaliado o impacto de recursos computacionais quando do uso do protótipo. Destaca-se já haver aplicações reais em mercado, o que pode oportunizar situações interessantes de investigação. Para o desenvolvimento do protótipo, considerando-se o seu emprego para fins didáticos, implementou-se em *Object Pascal* – no caso, Delphi, mas poder-se-ia empregar Lazarus ou Kylix - e, para fins de configurações do tratamento gráfico, a linguagem Lua. Medições de desempenho não se revelaram elevadas.

Palavras chave: Segurança Computacional, Criptografia, Criptografia Visual.

Abstract

This paper presents a prototype of visual cryptography, based in several formal recent studies – mathematics based. Some computational features were analyzed. As there are some real applications, this work can be useful to create new investigations situations to study. Considering using in classes, this prototype was developed in *Object Pascal* – Delphi, but could be done in Lazarus or Kylix -, and Lua language, considering graphic treatment configurations. Performance measurements were not elevated.

Keywords: Computer Security, Cryptography, Visual Cryptography.

1 INTRODUÇÃO

Por criptografia visual entende-se ao conjunto de métodos criptográficos que permitem cifrar informações empregando figuras, textos ou outro tipo de informação visual. Contudo, diferente da esteganografia – conjunto de técnicas que permite ocultar informações, por exemplo, em uma imagem -, a criptografia visual emprega um conjunto de técnicas que possibilita trabalhar separar informação escrita em partes - não legíveis - tais, que o sistema visual humano possa decodificá-lo, uma vez unidas as partes. Os primeiros trabalhos datam de 1994, com o trabalho pioneiro de Naor e Shamir **[4]**. Basicamente, o trabalho propunha o princípio da divisão de segredo **[4,5]**: a separação em n partes de tal forma que apenas quando as n partes estivessem juntas, seria possível visualizar o conteúdo da mensagem.

Criptografia visual pode ser empregada, por exemplo, para fins de votação, onde se pode manter um comprovante de votação que pode ser necessário, para fins de comprovação, em caso de verificação manual da contagem de votos ou identificação de discrepâncias. Esses comprovantes poderiam ser transparências, divididas em partes - por exemplo, uma, para o eleitor, uma para a central de votação e outra para a central de cadastramento. O presente trabalho apresenta uma implementação de criptografia visual, que foi desenvolvida empregando as linguagens *Object Pascal*

e Lua - essa última, empregada para fins de configuração dinâmica.

O presente trabalho está organizado da seguinte forma: breve referencial sobre Criptografia Visual é colocado na seção 2. A seguir, considerações sobre a implementação são levantadas. Na seção 4, são descritas a forma de validação do protótipo, bem como definido o processo de medição de indicadores de desempenho – memória empregada e tempo para geração de máscaras. Por fim, é apresentado breve levantamento do custo de processamento, com a finalidade de medir o impacto no processamento, e são apresentadas aplicações já existentes.

2 CRIPTOGRAFIA VISUAL

Há literatura específica apresentando o formalismo para justificar a viabilidade de se empregar criptografia visual em contextos ou situações específicas **[1,3]**. Naor **[4]** partiu de um modelo básico, o qual emprega uma situação especial - informação visual - do problema do compartilhamento de segredo: dada uma mensagem (escrita), como gerar n transparências de tal forma que a mensagem original só se torne legível se ao menos k cópias forem justapostas – sendo $k < n$. Qualquer número de cópias $i < k$, ao serem reunidas, não torna a mensagem legível.

Para tanto, supõe-se que a mensagem consiste, basicamente, de um conjunto de pixels pretos e brancos, e cada pixel é tratado individualmente. A figura que segue apresenta, de modo auto-explicativo, a lógica de sobreposição de pixels.

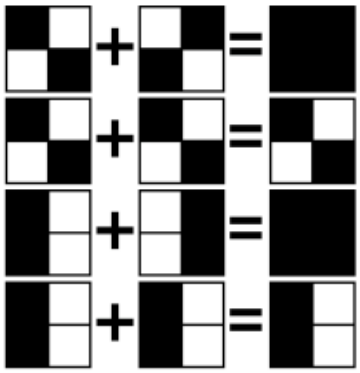

Figura 1 - sobreposição de pixels

Um exemplo de emprego é ilustrado na figura a seguir. A variedade de aplicações possíveis para a criptografia visual é extensa, uma vez que reúne atributos que a tornam, aparentemente, imune a ataques. Alguns exemplos de empregos são apresentados na última seção do presente trabalho.

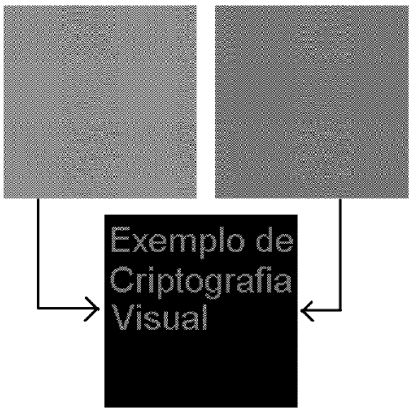

Figura 2 - exemplo de criptografia visual.

A seção seguinte aborda aspectos técnicos do protótipo desenvolvido.

3 O PROTÓTIPO DESENVOLVIDO

Um dos objetivos do presente trabalho envolve a definição e implementação de um programa protótipo capaz de realizar as operações referentes ao processo de criptografia visual. O desenvolvimento de tal programa foi vital para a condução de experimentos para validação do projeto.

3.1.Detalhamento do Processo

Entes de iniciar a utilização do programa, o usuário deve definir as dimensões da mensagem e a quantidade de subpixels a ser utilizada. Essa parametrização é realizada editando-se o script Lua responsável pela chamada das funções de baixo nível responsáveis pela criptografia visual propriamente dita. Embora a edição de um arquivo não seja uma forma muito prática para a entrada de parâmetros do usuário, optou-se por utilizar este método para facilitar a utilização do programa em várias plataformas, independentemente de interface. Como a parametrização é realizada através do script, pode-se utilizar interfaces simplificadas, sejam gráficas ou em linhas de comando.

Após a definição desses parâmetros iniciais, o usuário deve informar ao programa a localização da imagem que contém a informação que se deseja cifrar. O programa gera então uma imagem base, composta por pontos aleatoriamente distribuídos, com exatamente as mesmas dimensões da imagem selecionada. Essa imagem base pode ser melhor visualizada como sendo uma matriz m por n, sendo m e n a largura e altura da imagem em pixels, valores estes também definidos pelo usuário. O programa percorre essa "matriz" e, utilizando uma rotina de números pseudo-aleatórios (i.e., sem utilização de dados do mundo físico), "liga" determinados pixels. A imagem resultante é então submetida ao processo de codificação de pixels, produzindo assim a máscara utilizada como "chave" para decifrar a mensagem. A imagem base original é armazenada em disco, e utilizada para realizar a criptografia.

Após a geração da máscara para decifragem, é então realizada uma operação XOR entre a imagem base e a imagem contendo a mensagem, isto é, cada pixel da imagem base XOR o pixel correspondente da mensagem. O resultado dessa operação é uma terceira imagem contendo o texto cifrado. Essa imagem cifrada é então também submetida ao processo de codificação de pixels, gerando uma segunda máscara. Ambas as máscaras podem ser impressas ou, então, armazenadas em arquivo.

Quanto ao processo de codificação de pixels mencionado anteriormente, este consiste em desdobrar cada pixel em subpixels, de acordo com a opção do usuário, sendo que quanto maior o número de subpixels solicitado, maiores serão as máscaras resultantes. Por exemplo, para uma codificação de pixels, utilizando quatro subpixels (o número mínimo para o funcionamento da técnica **[3, 4]**), a imagem dobra de tamanho. Esse desdobramento é efetuado através de uma função que realiza operações matemáticas para criar uma nova imagem, representando a codificação dos pixels da imagem informada como parâmetro. O algoritmo da figura 3 exemplifica como é realizada uma codificação de pixels utilizando quatro subpixels:

```
maxX,maxY,X,Y: Inteiro;
maxX, maxY := Tamanho(Mensagem)
Resultado :=Imagem.Criar(2*maxX, 2*maxY)
para X := 1 até maxX faça
  para Y := 1 até maxY faça
    pixel := Mensagem[x,y];
    Resultado[2*x,2*y] := pixel;
    Resultado[2*x,2*y+1] := not pixel;
    Resultado[2*x+1,2*y] := not pixel;
    Resultado[2*x+1,2*y+1] := pixel;
  fim_para
fim_para
```

Figura 3 - Algoritmo para codificação dos pixels de uma imagem.

No presente projeto, foi explorado o aumento da quantidade de subpixels como forma de melhorar a definição da imagem reconstruída pela sobreposição das máscaras. Entretanto, uma vez que a modificação da quantidade de subpixels implica em um aumento das dimensões das imagens, esta não configura uma opção muito prática, considerando que a melhoria da definição é pouco considerável e tem pouca influência para a leitura da mensagem. Uma codificação utilizando 4 subpixels sobre uma imagem de 300 pixels x 300 pixels resulta em uma imagem impressa de 15cm x 15cm. O aumento da quantidade de subpixels aumenta proporcionalmente as dimensões da imagem impressa, assim uma codificação de 8 subpixels sobre a mesma imagem resultaria em uma impressão de 30cm x 30cm e uma codificação de 16 subpixels em uma impressão de 60cm x 60cm, sem entretanto propiciar uma melhoria significativa na definição da imagem reconstruída.

3.2 – Arquitetura do protótipo desenvolvido

A figura que segue representa a arquitetura do protótipo, na forma de um diagrama de blocos, representando as fases descritas nos parágrafos anteriores – definição de parâmetros, criação da imagem base e da mensagem, a criptografia propriamente dita e a codificação de pixels.

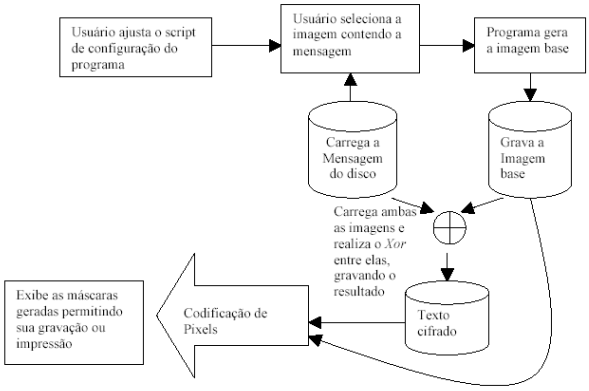

Figura 4 - arquitetura do protótipo.

O programa segue o método original da criptografia visual, definido por Naor e Shamir, produzindo duas imagens monocromáticas, as quais a partir de agora serão referidas como "máscaras". O protótipo permite aumentar a quantidade de subpixels de cada máscara, a fim de possibilitar a avaliação da influência da quantidade de subpixels na qualidade da mensagem "reconstruída".

3.2 Linguagens de programação utilizadas

Para a implementação do protótipo, foram escolhidas as linguagens *Object Pascal* e Lua, sendo esta última uma linguagem de construção de scripts criada no Brasil. A linguagem *Object Pascal* foi escolhida por sua grande agilidade, versatilidade e vasto material de referência e bibliotecas para manipulação de imagens disponíveis na Internet. Além disso, a linguagem *Object Pascal* possui diversos compiladores para inúmeras plataformas. Dentre estes, pode-se citar o programa "RAD" (*Rapid Application Development*) Delphi, para Windows, e o compilador Lazarus, para plataformas Unix em geral.

Quanto à escolha da linguagem LUA, esta foi motivada pela sua fácil integração com *Object Pascal*, possibilitando inclusive chamadas a funções internas do programa compilado. Dessa forma, o fluxo de execução do programa pode ser controlado, ou até mesmo alterado, de forma mais flexível que um programa compilado, bastando para isso realizar alterações no script Lua. Da mesma forma que a linguagem *Object Pascal*, há grande quantidade de material de referência e bibliotecas disponíveis para LUA, bem como interpretadores da linguagem para diversas plataformas. Para desenvolvedores que estão habituados a linguagens de programação desenvolvidas em países do hemisfério norte, Lua é uma honrosa exceção visto que foi projetada, implementada e desenvolvida no Departamento de Informática da PUC-Rio. Um histórico da evolução de Lua foi apresentado em junho 2007 na HOPL III, a 3a Conferência da ACM sobre a

História das Linguagens de Programação que ocorre a cada 15 anos. Segundo o índice TIOBE, encontra-se entre as 20 mais populares na Internet e é a única de impacto desenvolvida fora do primeiro mundo. Para maiores detalhes sobre a linguagem ver referências encontradas no sítio da linguagem LUA [6,7].

A porção executável do protótipo, escrita em *Object Pascal*, "hospeda" a segunda parte do protótipo, escrita na linguagem Lua. A utilização de uma linguagem de scripts dá maior dinamicidade ao protótipo, uma vez que o programa não necessita ser inteiramente recompilado para toda e qualquer alteração feita. Assim, pode-se alterar a forma como o programa é executado simplesmente alterando o script Lua, sem necessidade de recompilar o programa. Na linguagem *Object Pascal* foram escritas as porções do código com maior custo computacional, ou seja, as funções para geração da máscara, para realização da operação XOR entre as imagens, e o processo de codificação de pixels. Em LUA está desenvolvida a "parte lógica" do programa, isto é, as atribuições de variáveis, e chamadas de funções. Também no script LUA se encontram as variáveis de configuração do protótipo. O trecho de código LUA abaixo representa parte do script de configuração do protótipo.

```
X,Y=300,300
CaminhoMensagem = VerificarCaminhoMensa
gem()
If (CaminhoMensagem="")

       SinalizaErro([[Caminho da mensagem
não definido. Impossível prosseguir]])

Else
         DefineSubPixels(4)
         GerarBase(X,Y)

       CriptografiaVisual(X,Y,CaminhoMensa
       gem)
end
```

Figura 5 - Trecho do script de configuração do programa.

No script LUA acima, X e Y representam o tamanho definido para a máscara e a mensagem. As funções VerificarCaminhoMensagem, SinalizaErro, DefineSubPixels, GerarBase e CriptografiaVisual encontram-se implementadas no programa Object Pascal, compilado. VerificarCaminhoMensagem verifica se o usuário informou a localização da imagem que deseja criptografar. SinalizaErro instrui o programa a exibir uma mensagem de erro. DefineSubPixels informa a quantidade de subpixels que o programa deverá utilizar no processo de codificação dos pixels das imagens. GerarBase cria a imagem que será utilizada como base para o processo de criptografia, e CriptografiaVisual dispara o processo de criptografia propriamente dito.

O mecanismo de integração de LUA permite que o script realize chamadas a funções e métodos do programa hospedeiro, possibilitando alterar seu comportamento. Assim, o script acima primeiramente executa uma função do programa hospedeiro para verificar se foi informada uma mensagem. Caso positivo, é disparado o processo de criptografia visual. Caso contrário, o programa hospedeiro é instruído a exibir uma mensagem de erro. Os operadores "[[" e "]]" são um tipo especial de limitador de strings utilizado por LUA, o qual permite que sejam inseridas quebras de linha dentro da string.

Graças a sua portabilidade, estas linguagens permitem que o protótipo seja independente de plataforma, desde que exista para o sistema operacional em questão um interpretador para a linguagem LUA e um compilador *Object Pascal* compatível com a arquitetura. Em princípio, o protótipo foi desenvolvido tendo em vista a plataformas Windows, por ser a mais comum atualmente, embora as linguagens escolhidas sejam suportadas em vários outros sistemas operacionais como, por exemplo Solaris, Linux e outras variações do sistema Unix.

Embora a utilização de uma linguagem de scripts, interpretada e potencialmente mais lenta, possa prejudicar a velocidade de certas operações, reforça-se que o desempenho do protótipo não é uma das preocupações do projeto, embora trabalhos futuros possam ter por objetivo a melhoria do desempenho do mesmo.

A próxima seção aborda a validação funcional e a medição dos recursos computacionais necessários.

4 AVALIAÇÃO DA FUNCIONALIDADE E DO CUSTO COMPUTACIONAL

Para fins de validação da funcionalidade do protótipo, foram realizados testes de percepção com seres humanos, separados em dois grupos: um grupo estava completamente consciente do processo tendo, inclusive, o conhecimento do conteúdo da mensagem. Para esse grupo, deveria mostrar onde – em que ponto – a mensagem tornava-se visível. O segundo grupo tinha o conhecimento prévio do processo, devendo, da mesma forma que o primeiro, apresentar o ponto onde a mensagem se tornava visível. Já o terceiro grupo era composto de pessoas que não conheciam o processo, mas que deveriam indicar qualquer informação visível, se essa ocorresse. Assim, ter-se-ia uma adaptação do teste de Solomon com três grupos, buscando prever os efeitos teste (1^o e 2^o grupos) e efeito história (os primeiros grupos e o 3^o grupo).

Assim, utilizando o programa desenvolvido, foram geradas seis máscaras (empregando transparências) e uma mensagem cifrada, e distribuídas aleatoriamente a cada uma das seis pessoas cada uma das seis máscaras - sendo que apenas uma das seis máscaras poderia revelar a verdadeira mensagem, enquanto duas das restantes foram preparadas para mostrar mensagens falsas. As máscaras foram então distribuídas a seis pessoas de cada grupo, e então a mensagem foi circulada entre as mesmas.

Apenas o portador da máscara correta foi capaz de ler a mensagem verdadeira, enquanto dentre os demais, três não foram capazes de compreender a mensagem, e dois apenas obtiveram as mensagens falsas preparadas. Esse teste fora repetido 37 vezes, obtendo-se sempre o mesmo resultado – o que, numa primeira análise, valida o processo realizado.

Com relação aos recursos computacionais requeridos para a execução do mesmo, foi desenvolvido um pequeno programa monitor que obtinha os dados de memória empregada, bem como o tempo da operação de criação das máscaras a partir da mensagem original.

5 CONSIDERAÇÕES FINAIS E APLICAÇÕES REAIS

A presente seção apresenta algumas limitações, aplicações reais que já se encontram disponíveis e as considerações sobre o trabalho realizado.

Uma das suposições iniciais fora que o aumento na quantidade de subpixels utilizada no processo propiciaria melhoria no processo em geral, quando da reconstrução da mensagem original. Por ocasião dessa análise observou-se uma melhoria sutil na definição das imagens reconstruídas; porém, esse processo detém um efeito colateral: produz um aumento drástico nas dimensões das imagens impressas, dificultando ou mesmo impossibilitando sua impressão em equipamentos comuns – foram empregadas impressoras *laserjet* com resolução de 600 dpi. Dessa forma, em termos de relação custo-benefício, tende-se a crer que é preferível utilizar a codificação básica com quatro subpixels. Estudos em andamento buscam estimar que fatores colaborem para ocorrer esse aumento, bem como a quantificação do mesmo.

Outra limitação constatada foi o fato de que o alinhamento das máscaras deve ser perfeito para permitir a visualização da mensagem. O menor deslocamento impede que a informação seja corretamente visualizada – no experimento com as pessoas, esse foi o fator que maior tempo demandou.

5.1 Aplicação de Criptografia Visual: Proteção de Marcas

A empresa Trustmark, sediada em Singapura, apresenta um serviço para autenticação de produtos baseado em criptografia visual. O serviço possibilita a incorporação de informações ao rótulo ou embalagem de produtos, através de criptografia visual. O comprador pode averiguar a autenticidade dos produtos adquiridos utilizando uma pequena peça plástica contendo uma máscara impressa.

Ao posicionar a "Lens Key" em certas áreas do rótulo do produto, informações sobre a procedência do mesmo tornam-se visíveis **[7]**.

Figura 6 - emprego da Lens Keys da Trustmark sobre um rótulo, visando garantir a autenticidade do produto.

5.2 Aplicação de Criptografia Visual: Votação Remota

Em abril de 2003, foi apresentado, nos Estados Unidos, um sistema para votação autenticada via web. Nesse sistema, o eleitor recebe do cartório eleitoral uma máscara impressa e um código. Ao receber a máscara, o eleitor acessa o site de votação e digita o código que recebeu com a máscara. O sistema solicita então algumas informações pessoais e, após a confirmação da identidade, uma segunda máscara é exibida na tela. O eleitor então posiciona sua própria máscara sobre a tela, revelando uma senha que lhe possibilitará emitir seu voto **[2]**, possibilitando permanecer com uma evidência física – um comprovante de seu voto.

5.3 Considerações finais

Ao contrário do que se supunha inicialmente, o custo computacional não se revelou elevado. O emprego da linguagem Lua permitiu o desenvolvimento da interface de configurações, o que facilitou o gerenciamento das mesmas de modo automatizado.

Após 37 experimentações, chegou-se ao tempo médio de geração das transparências de 44,5 s, a partir da imagem base. A quantidade média de memória requerida para o processo chegou a 322 kB.

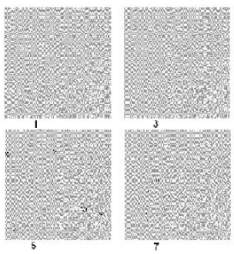

Figura 7 a - transparências contendo a imagem base

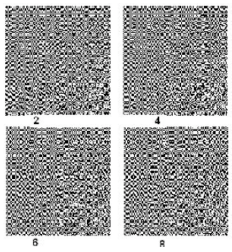

Figura 7b - transparências contendo a mensagem.

Assim, pode-se considerar que o emprego de criptografia visual é viável do ponto de vista de requisitos de desempenho computacional.

AGRADECIMENTOS
Os autores agradecem aos revisores as sugestões que serviram para melhorar a qualidade do texto.

REFERÊNCIAS
[1].ATENIENSE, G.; BLUNDO C. DE SANCTIS, A. and STINSON, D. Visual Cryptography for General Access Structures. Disponível na Internet em http://citeseer.ist.psu.edu/ateniense96visual.html. Acesso em 25.09.2006.
[2].EVANS, D., Paul, N., RUBIN, A., WALLACH, D. Authentication for Remote Voting, Workshop on Human-Computer Interaction and Security Systems, April, 2003.
[3].NAOR, M. and PINKAS, B. Visual Authentication and Identification. *Advances in Cryptology: CRYPT '97*. Disponível na Internet em http://citeseer.ist.psu.edu/67294.html . Acesso em 12.08.2006.
[4].NAOR, M. and SHAMIR, A. Visual Cryptography. *Advances in Cryptology: EUROCRYPT '94*, A. De Santis, ed., Lecture Notes in Computer Science 950 (1995), 1-12. Disponível na Internet em http://citeseer. ist.psu.edu/naor95visual. Acesso em 27.09.2006.
[5].SCHNEIER, B. Applied Cryptography: Protocols, Algorithms and Source Code in C. New York: John Wisley, 1994.
[6].Site oficial da linguagem Lua. Acesso em novembro de 2006. http://www.lua.org
[7].Site da empresa Trustcopy. Acesso em julho de 2006. http://www.trustcopy.com

Giuliano Berlatto Marques possui graduação em Ciência da Computação pelo UniLaSalle (2002), e é especializando em Tecnologias Aplicadas a Sistemas de Informação (UniRitter). Atua em diversas organizações como consultor de empresas.

Vinicius Gadis Ribeiro possui graduação em Ciências Náuticas pelo Ministério da Marinha (1984), graduação em Ciências da Computação pela Universidade Federal do Rio Grande do Sul (1994), mestrado em Administração pela Universidade Federal do Rio Grande do Sul (1997) e doutorado em Ciências da Computação pela Universidade Federal do Rio Grande do Sul (2005). Atualmente é professor da Faculdade Cenecista Nossa Senhora dos Anjos e professor adjunto do Centro Universitário Ritter dos Reis. Tem experiência na área de Ciência da Computação, com ênfase em Segurança da Informação, atuando principalmente nos seguintes temas: segurança computacional, criptografia, redes de computadores, criptografia de chave pública e sistemas de detecção de intrusão. Suas áreas de interesse incluem ainda a Computação Simbólica e a Simulação de Fenômenos em Ambientes Naturais.

Jorge Rodolfo Silva Zabadal possui graduação em Engenharia Química pela Universidade Federal do Rio Grande do Sul (1987), mestrado em Engenharia Mecânica pela Universidade Federal do Rio Grande do Sul (1990) e doutorado em Engenharia Mecânica pela Universidade Federal do Rio Grande do Sul (1994). Atualmente é Professor Adjunto da Universidade Federal do Rio Grande do Sul. Tem experiência na área de Engenharia Mecânica, com ênfase em Fenômenos de Transporte. Atuando principalmente nos seguintes temas: transporte de nêutrons, Método Nodal, Equação SN.