

UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Luís Otávio Villela Antunes

**PROPOSTA DE UM CATÁLOGO DE PADRÕES DE SEGURANÇA DA
INFORMAÇÃO**

BAURU - SP

2020

LUÍS OTÁVIO VILLELA ANTUNES

PROPOSTA DE UM CATÁLOGO DE PADRÕES DE SEGURANÇA DA INFORMAÇÃO

Trabalho de Conclusão de Curso do Curso de
Bacharelado em Ciência da Computação da
Universidade Estadual Paulista “Júlio de
Mesquita Filho”, Faculdade de Ciências, campus
Bauru.

Orientador: Prof. Dr. Kleber Rocha de Oliveira

BAURU

2020

PROPOSTA DE UM CATÁLOGO DE PADRÕES DE SEGURANÇA DA INFORMAÇÃO

Trabalho de Conclusão de Curso do Curso de
Bacharelado em Ciência da Computação da
Universidade Estadual Paulista “Júlio de
Mesquita Filho”, Faculdade de Ciências, campus
Bauru.

Orientador: Prof. Dr. Kleber Rocha de Oliveira

Banca examinadora

Prof. Dr. Kleber Rocha de Oliveira

Campus Experimental de Rosana
Universidade Estadual Paulista “Júlio de Mesquita Filho”

Profa. Dra. Simone das Graças Domingues Prado

Departamento de Computação
Faculdade de Ciências
Universidade Estadual Paulista “Júlio de Mesquita Filho”

Prof. Dr. Kelton Augusto Pontara da Costa

Departamento de Computação,
Faculdade de Ciências
Universidade Estadual Paulista “Júlio de Mesquita Filho”

BAURU

2020

RESUMO

Desenvolver um *software* de forma adequada e com qualidade é o objetivo de qualquer empresa do setor, e o aspectos de segurança é um dos pontos a serem considerados durante sua implementação e todo do ciclo de processo do *software*. Desde as primeiras etapas de desenvolvimento até a distribuição de uma ferramenta de *software*, as práticas de segurança desempenham um papel importante para garantir que o usuário final possa ter acesso a uma ferramenta confiável. Este é o desafio que as empresas têm que enfrentar, pois para desenvolver um *software* seguro é necessário a adoção de métodos que garantam a qualidade durante todas as etapas do desenvolvimento, além da realização de uma análise do processo de *software*. Nessa missão, é importante escolher conceitos e métodos que já foram aplicados com sucesso na área da computação e por isso essa pesquisa sugere a aplicação de padrões, conhecidos como *Patterns*. Padrões tratam de soluções, projetos reutilizáveis e interações de objetos. Neste trabalho, propõe-se a construção de catálogo de padrões de Segurança da Informação, baseando-se nos conceitos de Segurança da Informação da ISO 27000, nos principais conceitos relacionados a segurança do sistema de informação e os próprios fundamentos de *Patterns*. Desta forma, extrairá os principais padrões relacionados aos aspectos importante, sugerindo soluções para os problemas de segurança em forma de catálogo incluindo sua implementação, que contribuam para qualidade do *software* produzido.

Palavras-chave: catálogo padrões, ISO 27000, padrões, segurança de *software*.

ABSTRACT

To adequately develop quality software is the main objective of any software company, the security aspects are to be considered along the entirety of its implementation and development cycles. From the first development steps until its distribution, security practices play an important role to ensure the end-user access to a reliable tool. This is the challenge many software companies must face, so to develop secure software it is necessary to adopt methods that guarantee the quality along all development steps, alongside the software process analysis. During this task it is paramount to choose concepts and methods already successfully applied in computing and therefore this research suggests the application of Patterns. Patterns are solutions, reusable projects, and objects. In this work, is proposed the construction of an Information Security Patterns Catalog, based in the concepts of ISO 27000, the main security related concepts and Patterns fundamentals. This way, it will extract the main Patterns, suggesting solutions to security related issues in a catalog format with its implementation included, contributing to the quality of the made software.

Keywords: patterns catalog, ISO 27000, patterns, software security.

Lista de Gráficos

Gráfico 1- O custo total médio relacionado a vazamentos de dados	12
Gráfico 2 - Distribuição das causas de vazamento de dados da amostra de referência.....	13

Lista de Figuras

Figura 1 - Tríade ilustrativa dos três objetivos da CIA em Segurança Cibernética	14
Figura 2 - Tentativa de Registro 1.....	32
Figura 3 - Tentativa de Registro 2	33
Figura 4 - Tentativa de Registro 3	33
Figura 5 - Cadastro bem sucedido	34
Figura 6 - Sucesso login	37
Figura 7 - Senha ou Usuário Incorreto	37
Figura 8 - Seleção de Arquivo para Backup.....	41
Figura 9 - Seleção de Destino de Backup.....	41
Figura 10 - Resultado do Backup	42
Figura 11 - Tela de Login do Auditor.....	44
Figura 12 - Janela de Interação do Auditor	44
Figura 13 - Janela de Leitura de <i>Logs</i>	45
Figura 14 - Seleção de Arquivo para criptografia	48
Figura 15 - Senha Digitada.....	48
Figura 16 - Chave Gerada.....	49
Figura 17 - Salvar a chave	49
Figura 18 - Encriptando o arquivo.....	50
Figura 19 - Arquivo encriptado	50
Figura 20 - Carregamento da Chave.....	51
Figura 21 - Decriptar arquivo	51
Figura 22 - Arquivo decriptado	51
Figura 23 - Janela de interação do usuário autorizado	53
Figura 24 - Logs desativados.....	53
Figura 25 - Inserção de arquivos	54
Figura 26 - Arquivos criptografados	54
Figura 27 - Arquivo criptografado.....	55
Figura 28 - Arquivos decriptografados.....	55
Figura 29 - Arquivo decriptografado.....	56
Figura 30 - Timeout por inatividade.....	60

Lista de Fluxogramas

Fluxograma 1 - Os Conceitos de Segurança e Relacionamentos	15
Fluxograma 2 - Criptografia Simétrica.....	19

Lista de Quadros

Quadro 1 - Computadores e Ativos de Rede, com exemplos de ameaças.....	18
Quadro 2 - Arquivo de registro de Usuários	34
Quadro 3 - Informações de Registro de Usuário	35
Quadro 4 - Arquivo de Registro de Logins	39
Quadro 5 – Padrões Desenvolvidos.....	30

SUMÁRIO

1	INTRODUÇÃO	11
1.1	Problema	12
2	FUNDAMENTAÇÃO TEÓRICA.....	14
2.1	Segurança da Informação	14
2.1.1	Disponibilidade.....	15
2.1.2	Confidencialidade	16
2.1.3	Integridade	17
2.1.4	Irretratabilidade	17
2.1.5	Ameaças.....	18
2.2	Técnicas de Segurança	19
2.2.1	Criptografia.....	19
2.2.2	Backup	19
2.2.3	Logs	20
2.2.4	Proteção contra Malware	20
2.2.5	Tela Limpa e Mesa Limpa.....	20
2.3	Normas de Segurança.....	21
2.3.1	ISO 27000.....	21
2.3.2	ISO 27001	21
2.3.3	ISO 27002.....	21
2.4	Política de Segurança	22
2.5	Auditoria	22
2.6	Gestão de Identidade e Autenticação.....	23
2.7	Gestão de Autorização	23
3	PADRÕES	24
3.1	Histórico	24
3.2	Descobrimento	25
3.3	Formas de Padrões	25
3.3.1	Minimalista.....	25
3.3.2	Gamma	26
3.3.3	The Open Group	27
3.3.4	Posa.....	27

4	DESENVOLVIMENTO DE PADRÕES	29
4.1	Senha Segura.....	31
4.1.1	Exemplo de Uso	32
4.2	Autenticação.....	36
4.2.1	Exemplo de Uso	37
4.3	Registro de Ocorrência	38
4.3.1	Exemplo de Uso	39
4.4	Replicação de Dados.....	40
4.4.1	Exemplo de Uso	41
4.5	Auditoria	43
4.5.1	Exemplo de Uso	44
4.6	Proteção de Dados	46
4.6.1	Exemplo de Uso	48
4.7	Autorização de Usuário.....	52
4.7.1	Exemplo de Uso	52
4.8	Gerenciamento de Chaves Criptográficas.....	57
4.8.1	Exemplo de Uso	57
4.9	Timeout de Sessão.....	59
4.9.1	Exemplo de Uso	60
5	CONCLUSÃO.....	61
	REFERÊNCIAS	63
	APÊNDICE A – SENHA SEGURA	66
	APÊNDICE B – AUTENTICAÇÃO	68
	APÊNDICE C – REGISTRO DE OCORRÊNCIA	69
	APÊNDICE D – REPLICAÇÃO DE DADOS	73
	APÊNDICE E – PROTEÇÃO DE DADOS	75
	APÊNDICE F – AUTORIZAÇÃO	79
	APÊNDICE G – GERENCIAMENTO DE CHAVES CRIPTOGRÁFICAS.....	80
	APÊNDICE H – TIMEOUT DE SESSÃO	81
	APÊNDICE I – CAPTAR DADOS DE CONEXÃO	82
	APÊNDICE J – GERADOR DE SALT.....	83
	APÊNDICE K – HASH DE SENHAS	84

INTRODUÇÃO

No contexto da informática, a história da segurança da informação é descrita por Lynett (2015) em sua evolução ao longo das décadas de 1960-2010.

Na década de 1960, organizações passam a bloquear o acesso aos computadores com uso de senhas. Já em 1970, no início das redes de computadores, as organizações iniciam o processo de conexão de seus sistemas de informação através de linhas de telefone. Percebendo essa oportunidade, os primeiros grupos de *hackers* surgem com o intuito de roubar tais informações. Apesar de o roubo ou interceptação de informações não ser uma realidade criada no Século XX, estes fatos apenas ocorriam com o acesso físico onde a informação estava armazenada ou no momento de sua transmissão, o que obviamente reduzia a oportunidade de ação. Com o advento da conexão entre computadores pela rede telefônica qualquer um com o acesso a mesma poderia, remotamente, roubar informações sigilosas.

Avançando para 1980, tais atividades criminosas começam a chamar a atenção das autoridades governamentais. Um grupo de adolescentes conseguiu invadir mais de 60 computadores de instituições do governo, com a grande cobertura midiática sobre o caso os legisladores começaram a aprovar leis para criminalizar a intrusão à sistema de informação. Até aquele momento, as punições eram leves.

Passando para 1990 e 2000, a intrusão de sistemas de informação tornou-se uma atividade lucrativa para o crime organizado e consequentemente a força da lei passou a realmente punir severamente os agentes maliciosos. A partir da década de 2010, praticamente todos os dados pessoais estão armazenados em sistemas de informação. Mesmo com punições severas e todo o aparato segurança empregado (anti-virus, *firewall*, filtros de *spam*, etc) as intrusões a sistemas de informação continuam a aumentar.

Para Devanbu e Stubblebine (2000) praticamente todos os sistemas de *software* precisam ser seguros, defendendo-se de possíveis agentes maliciosos. A dependência de nossa sociedade moderna de tais sistemas informáticos é crítica. As ameaças de segurança são potencialmente catastróficas. Como no caso da empresa de emissão de certificados digitais DigiNotar, que após emitir mais de 500 certificados fraudulentos em decorrência de uma falha de segurança, decretou falência em 2011 (ZETTER, 2011).

Entretanto, o roubo de informações não é a única ameaça a uma organização. As perdas de tais dados também podem ocorrer devido a acidentes nos locais de armazenamento (incêndios, terremotos, inundações), erro humano (deletar arquivos por engano, falta de

treinamento), arquivos corrompidos (falhas no sistema operacional durante escrita em arquivos). Deduz-se que nem todas as ameaças são maliciosas.

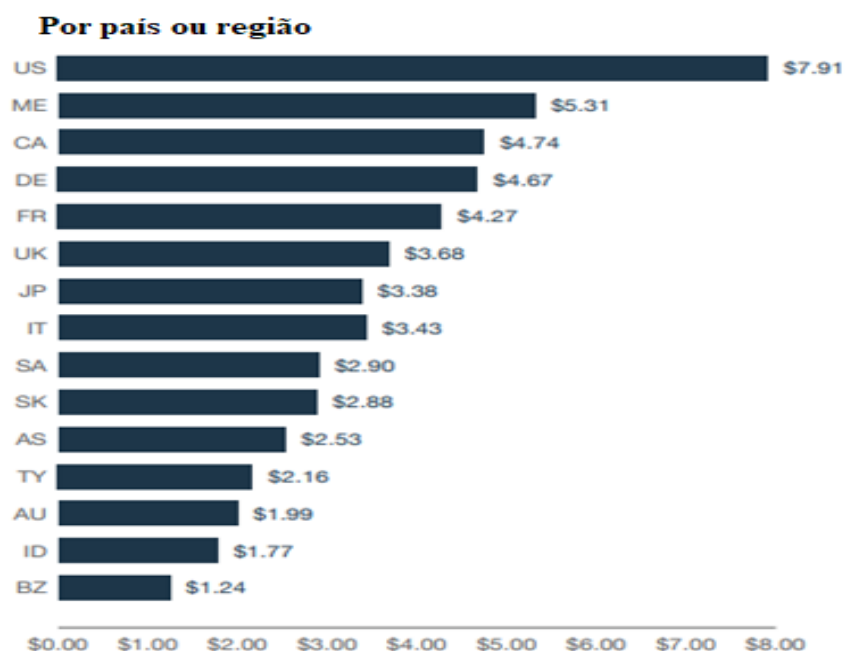
Para auxiliar a organização, Dougherty (et al.) (2009) definem padrões como soluções gerais reutilizáveis visando solucionar problemas comuns de projeto. No que tange à segurança, os padrões existem para eliminar a inserção accidental de vulnerabilidades no código-fonte e reduzir os impactos de tais vulnerabilidades.

Este projeto propõe um catálogo de padrões de segurança do ponto de vista dos ativos digitais das empresas, que fornecerem o ciclo de vida da informação, como entradas, armazenamento, processamento e distribuição na geração de informação e construção do conhecimento.

1.1 Problema

Um grande problema das organizações, em especial as corporativas, é segurança de seus ativos digitais. O vazamento, a destruição ou bloqueio de acesso dos dados ou informações armazenadas ou do funcionamento de seus sistemas informáticos causam grandes perdas financeiras e morais para a organização. As multas e processos judiciais decorrentes desses incidentes são potencialmente letais para a organização, como pode-se observar no gráfico a seguir, que ilustra o custo total médio relacionado a vazamentos de dados em milhões de dólares americanos por país ou região.

Gráfico 1 - O custo total médio relacionado a vazamentos de dados



Fonte: Adaptado de Ponemon Institute LLC (2018, p. 15)

A falta de consciência da legislação, de padrões de segurança e o treinamento deficitário dos colaboradores e gestores das organizações e a manutenção falha são as grandes ameaças à segurança dos sistemas de informação.

Em adição, Yoder & Barcalow (1998) argumentam que sistemas de informação são desenvolvidos desconsiderando os critérios de segurança da informação durante seu ciclo de implementação ou até mesmo na política de controle dos dados e uso da informação gerados por estes sistemas, conforme pode-se observar no gráfico 2.

Gráfico 2 – Distribuição das causas de vazamento de dados da amostra de referência



Fonte: Adaptado de Ponemon Institute LLC (2018, p. 19)

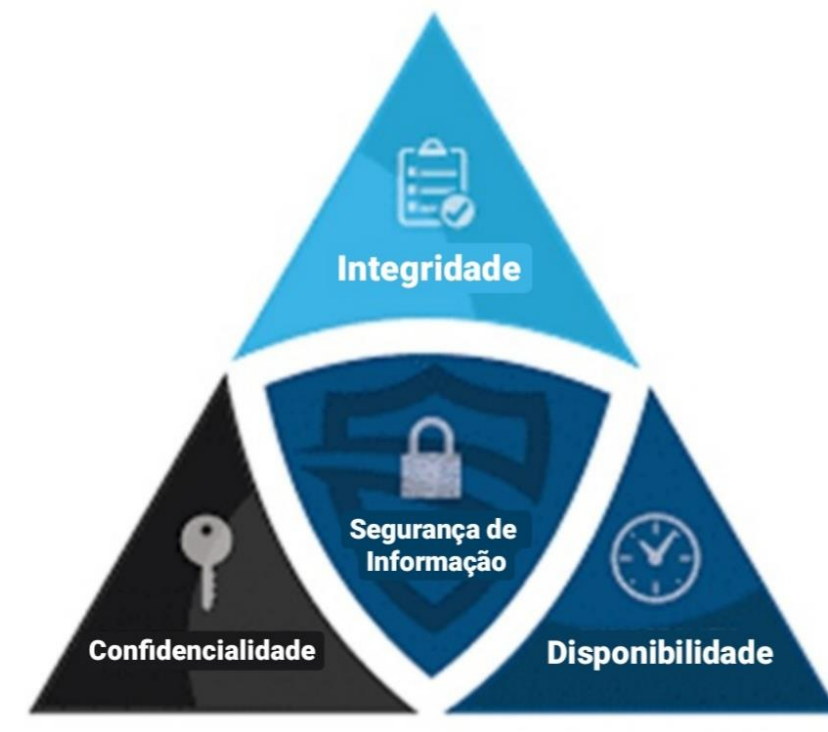
2 FUNDAMENTAÇÃO TEÓRICA

2.1 Segurança da Informação

A publicação 800-12 *An Introduction to Information Security* do *National Institute of Standards and Technology* (NIELES; DEMPSEY; PILLITTERI, 2017) define Segurança de Computadores (no contexto de Segurança da Informação) como “A proteção da informação e sistemas de informação de acessos, uso, divulgação, interrupção, modificação, ou destruição não autorizados em ordem para se garantir a confidencialidade, integridade e disponibilidade”.

Adicionalmente, a Norma 27000:2018 da ISO (*International Organization for Standardization*) define Segurança da Informação como a “preservação da confidencialidade, integridade e disponibilidade da informação”.

Figura 1-Tríade ilustrativa dos três objetivos da CIA em Segurança Cibernética

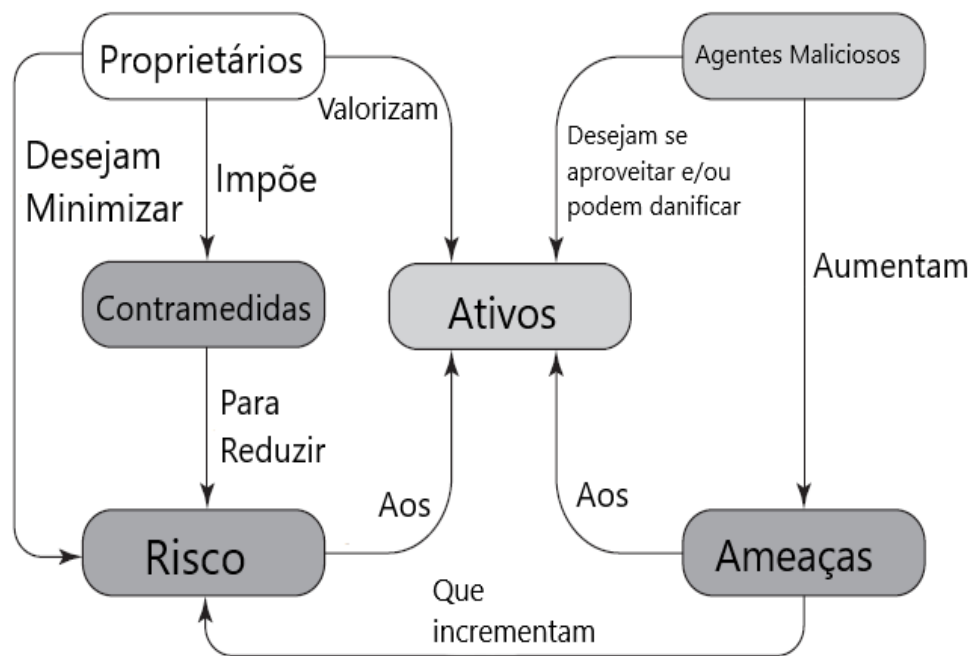


Fonte: Adaptado de Preferred IT Group (2019, p.1)

Segundo Peltier et al. (2005), a função da Segurança da Informação é proteger os recursos de uma organização, tais como informações, *hardware* (computadores, dispositivos de rede, por exemplo) e *software* (programas de gerenciamento de bancos de dados).

Através da seleção e utilização de métodos e controles apropriados, a segurança da informação auxilia a organização a alcançar seus objetivos protegendo seus ativos (tangíveis ou intangíveis), recursos físicos e financeiros, reputação, colaboradores e posição legal.

Fluxograma 1 – Os Conceitos de Segurança e Relacionamentos



Fonte: Adaptado de Stallings e Brown (2014, p. 18)

2.1.1 Disponibilidade

A norma ISO 27000:2018 define disponibilidade como “Propriedade da informação e/ou sistema de informação estar disponível e utilizável sob demanda por uma entidade autorizada” Adicionalmente, o catálogo NIST 800-12, traz a seguinte definição “De forma que o acesso seja feito em tempo hábil e de maneira confiável”. E de acordo com Stallings e Brown (2014), "Garante que os sistemas funcionem prontamente e que não haja negação de serviço a usuários autorizados".

A disponibilidade, é o estado onde recursos e dados estão disponíveis aos usuários quando são requisitados. Tal condição pode ser violada quando, por exemplo, ocorrem:

- Ataques de negação de serviço
 - Neste tipo de ataque, o sistema informático é sobrecarregado com uma quantidade de requisições muito maior do que consegue gerenciar, causando a impossibilidade de acesso por seus usuários legítimos.
- Falhas de *Hardware* e/ou *Software*
 - Arquivos críticos de *software* são corrompidos ou deletados (por exemplo, uma DLL essencial para o funcionamento do *software*).
 - O *software* é configurado de maneira incorreta.
 - Um componente físico do sistema deixa de funcionar corretamente ou é danificado:
 - Ocasionado por seu uso excessivo
 - Ocasionado por superaquecimento
 - Ocasionado por incêndio
 - Ocasionado por surto elétrico
- Falha em sistemas de comunicação.

2.1.2 Confidencialidade

A norma ISO 2700:2018 define confidencialidade como “Propriedade da informação e/ou sistema de informação estar indisponível à indivíduos, entidade e processos não autorizados”.

Segundo Olson et al. (2011) assegurar que terceiros não acessem informações sigilosas é uma tarefa complexa. Entretanto, tal tarefa pode ser separada em três grandes passos:

- 1- A informação deve ter mecanismos de proteção que impeçam o acesso por certos usuários.
- 2- Limitações à informação deve estar em funcionamento de forma que apenas aqueles com autorização possam ter acesso.
- 3- Um sistema de autenticação deve estar em funcionamento para verificar e identificar os indivíduos com acesso à informação.

Em adição, Stallings e Brown (2014) abrangem o conceito da privacidade, argumentando que o indivíduo deve estar no controle de quais informações podem ser coletadas e armazenadas e para quem e por quem tais informações podem ser reveladas.

Para satisfazer a condição da Confidencialidade, o acesso à informação deve ser restrito apenas a entidades autorizadas. Tal condição é violada quando ocorrem, por exemplo:

- Um departamento acessa informações de outro.
 - Desta forma, cada departamento deve ter acesso a apenas às informações que são de seu interesse.
- A conta de um usuário é acessada por um agente malicioso.
 - Uma forte política de senhas deve ser obrigatória, impedindo que senhas simples sejam utilizadas.
 - Os dados armazenados devem ser criptografados. Assim, mesmo que o acesso à conta seja comprometido, as informações contidas não poderão ser lidas.

2.1.3 Integridade

De maneira, bastante sucinta, porém precisa, a norma ISO 2700:2018 define integridade como “Propriedade de completude e acurácia”.

Adicionalmente, Stallings e Brown (2014) complementam o conceito, "Garante que informações e programas sejam alterados somente de maneira especificada e autorizada.”

A integridade, visa garantir que informações armazenadas estejam corretas e livres de modificações não autorizadas. Desta forma, para garantir tal condição devem ser aplicados métodos de controle de acesso (impedir o acesso não autorizado) e controle de integridade (*Hash*).

2.1.4 Irretratabilidade

De acordo com Stallings e Brown (2014), “Irretratabilidade impede que o remetente ou destinatário negue uma mensagem transmitida.”. Ademais, a ISO 27000:2018 define: “Habilidade de provar-se a ocorrência de um evento ou ação alegados por suas entidades originárias”.

A irretratabilidade garante que um indivíduo não possa negar a validade de uma ação atrelada à sua pessoa. Como por exemplo uma compra digital autorizada por sua impressão digital.

2.1.5 Ameaças

Stallings e Brown (2014) definem ameaças à segurança da informação como “uma potencial violação à segurança, que existem uma circunstância, capacidade, ação ou evento que podem causar danos. Ameaça é um risco que explora uma vulnerabilidade”.

Ademais, a definição de ameaça dada pela ISO 27000:2018 foi a seguinte: “Causa de um incidente indesejado em potencial, no qual causa danos ao sistema ou organização”.

Nesse sentido, através do quadro abaixo, pode-se notar algumas situações de ameaça devidamente classificadas.

Quadro 1 - Computadores e Ativos de Rede, com exemplos de ameaças

	Disponibilidade	Confidencialidade	Integridade
Hardware	Equipamento é roubado ou desabilitado, impedindo a sua utilização.	Um CD-ROM ou DVD descriptografado é roubado.	
Software	Programas são deletados, negando aos seus usuários o acesso.	Uma cópia não autorizada de um software é feita.	Um programa é modificado, para ou causar uma falha durante a sua execução ou para causar uma ação indesejada.
Dados	Arquivos são deletados, negando aos seus usuários o acesso.	Uma leitura não autorizada de dados é efetuada. Uma análise de dados estatísticos revela as informações armazenadas.	Arquivos preexistentes são modificados ou novos arquivos são forjados.
Linhas de Comunicação e Rede	Mensagens são destruídas ou deletadas. Linhas ou redes de comunicação tornam-se indisponíveis.	Mensagens são lidas. O padrão de tráfego das mensagens é observado.	Mensagens são modificadas, atrasadas, reordenadas ou duplicadas. Mensagens falsas são forjadas.

Fonte: Adaptado de Stallings e Brown (2014, p. 23)

2.2 Técnicas de Segurança

Nesta seção, serão discutidas técnicas de segurança da informação. Tais estas como criptografia, *backup*, *logs*, Proteção Contra *Malware*, e os conceitos de Tela e Mesa Limpas.

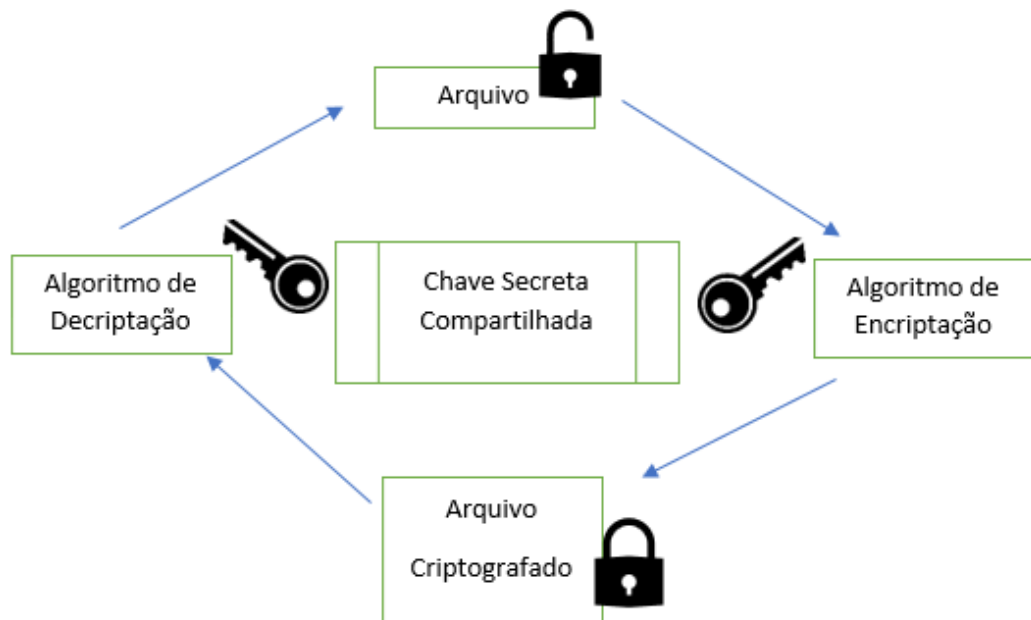
2.2.1 Criptografia

O dicionário Michaelis define criptografia como “Arte ou processo de escrever em caracteres secretos ou em cifras”.

No contexto da Segurança da informação, a Criptografia almeja garantir a confidencialidade da informação. Desta forma, a informação pode ser transmitida através de sistemas e redes inseguras. DHILLON (2007).

O Fluxograma 2 descreve o processo convencional de criptografia.

Fluxograma 2 – Criptografia Simétrica



Fonte: Adaptado de Dhillon (2007)

2.2.2 Backup

Stallings e Brown (2014) definem: “*Backup* é o processo de fazer cópias de dados em intervalos regulares, permitindo a recuperação de dados perdidos ou corrompidos”.

Logo, como apontado pelos mesmos Satllings e Brown (2014), o *Backup* é um controle que visa garantir a integridade de dados e sistemas de informação.

2.2.3 Logs

Logs, ou registro de eventos, é uma forma de controle reativa. Ou seja, é utilizada quando o problema já aconteceu. O seu uso é investigativo, assim permite-se identificar de maneira rápida e precisa o problema; possibilitando a concentração de esforços na recuperação. STALLINGS E BROWN (2014).

É importante salientar que os *logs* podem registrar qualquer atividade no sistema de informação, como: hora de *boot* e desligamento; endereços I.P. & MAC; usuário; estação; data e hora de acesso a um arquivo. Além disso, as organizações devem estar atentas aos requisitos legais, seja armazenar tais informações por período pré-determinado de tempo e/ou estar à disposição da Justiça em caso de investigações criminais e decisões judiciais.

2.2.4 Proteção contra *Malware*

Stallings e Brown (2014) trazem a definição do NIST (2005): *Malware* é “um programa que é inserido em um sistema, usualmente às escondidas, com a intenção de comprometer a confidencialidade, a integridade ou a disponibilidade dos dados, aplicações ou sistema operacional da vítima”.

Stallings e Brown (2014) citam algumas medidas preventivas:

Uma das primeiras contramedidas que devem ser empregadas é assegurar que todos os sistemas estejam em sua versão mais atualizada possível, com todos os *patches* aplicados, de modo a reduzir o número de vulnerabilidades que poderiam ser exploradas no sistema. A seguinte é estabelecer controles de acesso adequados nas aplicações e dados armazenados no sistema, para reduzir o número de arquivos que qualquer usuário pode acessar e, por conseguinte, potencialmente infectar ou corromper, como resultado da execução de algum código de *malware*

2.2.5 Tela Limpa e Mesa Limpa

As políticas de Tela Limpa e Mesa Limpa visam garantir que informações confidenciais (física ou digital) estejam protegidas mesmo quando a estação de trabalho não esteja supervisionada. GYSEN (2008).

A norma ISO 27002 formaliza algumas considerações:

- a) Informações críticas ou sigilosas da organização, por exemplo papéis ou mídia de armazenamento eletrônico, devem ser trancados quando não requisitados, especialmente quando o local de trabalho estiver vazio.
- b) Computadores e terminais devem permanecer desligados ou protegidos por um sistema de bloqueio com tela e teclado controlado por uma senha, *token* ou mecanismo de autenticação de usuário quando deixados sem supervisão e devem ser protegidos por senhas ou outros controles quando não estiverem em uso

- c) Uso não autorizado de foto copiadoras e outros dispositivos de reprodução (por exemplo, scanners, câmeras digitais).
- d) Mídia contendo informações sigilosas ou sensíveis devem ser removidas de impressoras imediatamente.

2.3 Normas de Segurança

Nesta seção falaremos de forma sucinta sobre algumas normas de segurança da ISO (International Standardization Organization).

2.3.1 ISO 27000

Esta norma serve como uma visão geral sobre segurança da informação, de seu vocabulário e as intenções para as normas da família de padrões 27000.

Define, de maneira sucinta, porém extremamente precisa, as definições sobre Segurança da Informação e sua tríade, Integridade, Disponibilidade e Confidencialidade.

2.3.2 ISO 27001

Esta norma formaliza os requisitos para o desenvolvimento e operação de um Sistema de Gerenciamento de Segurança da Informação. Define controles para gerenciar e reduzir os riscos aos ativos das organizações. Tudo isso de acordo com a ISO 27000:2018.

Desta forma, estabelece um modelo para implementar, operar, monitorar, revisar e melhorar os controles de um Sistema de Segurança da Informação.

2.3.3 ISO 27002

Conforme definição presente na ISO 27000:2018: "Este documento fornece uma lista de objetivos de controles comumente aceitos e controles de melhores práticas a serem usados como guia de implementação quando selecionando e implementado controles para garantir a Segurança da Informação"

2.4 Política de Segurança

Stallings e Brown (2014), definem que: “Política de Segurança é uma declaração formal de regras e práticas que especificam ou regulamentam como um sistema ou organização provê serviços de segurança para proteger ativos de sistemas sensíveis e críticos”. Além disso, complementam “Tal política de segurança formal deve ser cumprida pelos controles técnicos do sistema, bem como seus controles gerenciais e operacionais”.

A norma ISO 27002:2013 recomenda a implementação de uma Política de Segurança. Esta, deve estar de acordo com as regulações, legislações e contratos; e com as ameaças à segurança da informação.

A Política de Segurança deve ser genérica. Questões técnicas devem ser tratadas em documentos específicos. Por exemplo: a Política de Segurança determina que as estações de trabalho devem ser encriptadas; um documento específico (geralmente chamado de *Baseline*) indica os parâmetros e medidas necessárias para satisfazer esta condição de segurança.

2.5 Auditoria

Auditoria, é definida por Stallings e Brown (2014):

Revisão e Exame Independentes dos registros e atividades de um sistema para determinar a adequação de controles de sistema, assegurar conformidade à política e procedimentos de segurança estabelecidos, detectar violações em serviços de segurança e recomendar as mudanças indicadas relativas a contramedida.

Stallings e Brown (2014) definem o Controle de Auditoria:

Criar, proteger e manter registros de auditoria do Sistema de Informação pelo período que seja necessário para facilitar o monitoramento, análise, investigação e relato de atividades ilegítimas, não autorizadas ou inadequadas do Sistema de Informação; e assegurar que as ações de usuários individuais do Sistema de Informação possam ser rastreadas exclusivamente até esses usuários e cobrar deles a responsabilidade de suas ações.

2.6 Gestão de Identidade e Autenticação

Stallings e Brown (2014) definem Identificação e Autenticação nos seguintes termos: “Identificar usuários do Sistema de Informação, processos que agem em nome de usuários ou dispositivos e autenticar as identidades desses usuários, processos ou dispositivos, como pré-requisito para permitir acesso a Sistemas de Informações Organizacionais”.

2.7 Gestão de Autorização

Definido por Stallings e Brown (2014): “A concessão de um direito ou permissão à uma entidade do sistema para acessar um recurso do sistema. Esta função determina quem é confiável para dada finalidade.”

Desta forma, a partir da confirmação da identificação do usuário (autenticação), o sistema informático busca em sua base de dados os recursos ou subsistemas que o usuário legítimo tem acesso e os libera de acordo.

3 PADRÕES

Dougherty, et al.(2009) definem Padrão (*Pattern*) como uma solução geral, reutilizável, para um problema recorrente em um projeto. É uma descrição ou modelo de como se resolver um problema, sendo possível utilizá-lo em diversas situações.

De acordo com Craig (2009), os padrões visam formalizar, de maneira estruturada, a experiência e o conhecimento que profissionais experientes perceberam ao se depararem com problemas repetitivos. Desta maneira, transmitem-se as melhores soluções aos seus sucessores menos experientes, proporcionando-lhes uma grande oportunidade de evitar os mesmos erros e dificuldades.

No contexto da Segurança da Informação para Dougherty, et al. (2009), os Padrões visam eliminar a inserção accidental de vulnerabilidades no código e reduzir as consequências de tais vulnerabilidades.

3.1 Histórico

Dougherty, et al. (2009) apontam um breve histórico de Padrões:

- 1977/79 - O arquiteto Christopher Alexander introduz o conceito de *design patterns* no que tange à arquitetura de construções e cidades.
- 1987 – Beck e Cunningham testam a aplicação de padrões na programação de computadores e os apresenta na OOPSLA (*Object-Oriented Programming, Systems, Languages & Applications*)
- 1994/95 - A “*Gang of Four*” (composta pelos autores Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides) publicam um livro contendo uma grande quantidade de padrões de linguagens de programação orientadas a objeto.
- 1997 – Yoder e Barcalow publicam um artigo descrevendo alguns padrões de segurança

3.2 Descobrimento

Naturalmente, cogita-se que novos padrões são inventados ou criados. Entretanto, Larman (2007) aponta que este não é o caso. O padrão formaliza uma solução verdadeira e já testada. Logo, o padrão é descoberto, reconhecido, identificado.

Blakeley e Heath (2004) sintetizam o trabalho de James Coplien nesta questão, trazendo uma série de perguntas a serem feitas para identificar um padrão:

- É a solução para um problema, dado um contexto?
- É possível dizer ao solucionador de problemas o que deve fazer para resolvê-lo?
- É uma solução comprovada?
- É algo que você não inventou por si mesmo?
- Pode ser implementada diversas vezes, sem ser idêntica? Baseia-se na percepção do solucionador de problemas/
- A solução pode ser formalizada ou automatizada?
- Contém um conjunto de forças interativas que são independentes das forças em outros padrões?
- Escreve-lo é trabalho duro?

3.3 Formas de Padrões

Nesta seção serão descritas algumas formas utilizadas na descrição de Padrões. É importante notar que apesar de não existir uma única forma de se descrever um padrão, todas possuem o mesmo “núcleo” comum: um nome, a descrição de seu problema, a descrição de sua solução e as consequências da utilização do padrão.

3.3.1 Minimalista

É a forma mais simples. Blakeley e Heath (2004) destacam o absoluto mínimo para se definir um padrão.

- **Nome:** Deve ser memorável e sucinta para facilitar sua identificação.
- **Problema:** Descreve o contexto para a utilização do padrão.
- **Solução:** Descrição de como se resolve o problema.
- **Consequências:** Descreve as consequências (positivas e negativas) do padrão.

3.3.2 Gamma

É uma das formas mais detalhadas e extensas, sendo a mais apropriada para padrões de projeto e implementação de *software* (GAMMA et al, 1995).

- **Nome e Classificação:** Expressa, de maneira sucinta, a essência do padrão.
- **Intenção:** Descreve o que o padrão faz e qual problema este resolve.
- **Também Conhecido como:** Outros nomes conhecidos para o padrão, se existirem.
- **Motivação:** Descrever um cenário que ilustra o problema, mostrando como a classe e estrutura do objeto resolvem o problema.
- **Aplicação:** Descreve em quais situações o padrão pode ser aplicado.
- **Estrutura:** Uma representação gráfica da estrutura de classes do padrão. Normalmente utiliza o diagrama de classes OTM.
- **Participantes:** Classes e/ou objetos participantes no padrão e suas respectivas responsabilidades.
- **Colaborações:** Como os participantes executam suas responsabilidades na solução do problema.
- **Consequências:** Descreve as consequências resultantes do uso do padrão.
- **Implementação:** Descreve técnicas e orientações ao se implementar o padrão.
- **Amostra de Código:** Uma sugestão de como o padrão pode ser implementado.
- **Usos conhecidos:** Exemplos reais onde o padrão pode ser encontrado.
- **Padrões Relacionados:** Descreve padrões relacionados, tanto similares quanto outros padrões que possam ser usados em conjunto.

3.3.3 *The Open Group*

Apresentado por Blakeley e Heath (2004), é uma notação quase idêntica à forma Gamma. Sua única diferença, em termos de tópicos, é a ausência de uma Amostra de Código.

- **Nome:** Expressa, de maneira sucinta, a essência do padrão. Um nome facilmente identificável é essencial para a formação de um vocabulário.
- **Intenção:** Uma declaração sucinta que responde as seguintes questões. O que este padrão faz? Qual sua intenção? Qual problema ele pretende corrigir?
- **Também Conhecido como:** Outros nomes conhecidos para o padrão, se existirem.
- **Motivação:** Descrever um cenário que ilustra o problema, e como a estrutura do padrão resolve o problema.
- **Aplicação:** Descreve em quais situações o padrão pode ser aplicado.
- **Estrutura:** Uma representação gráfica da estrutura de classes do padrão.
- **Participantes:** Classes e/ou objetos participantes no padrão e suas respectivas responsabilidades.
- **Colaborações:** Como os participantes executam suas responsabilidades na solução do problema.
- **Consequências:** Descreve as consequências resultantes do uso do padrão, como o padrão sustenta seus objetivos.
- **Implementação:** Descreve técnicas e orientações ao se implementar o padrão.
- **Usos conhecidos:** Exemplos reais onde o padrão pode ser encontrado.
- **Padrões Relacionados:** Descreve padrões relacionados, tanto similares (indicando suas diferenças notáveis) quanto outros padrões que possam ser usados em conjunto.

3.3.4 POSA

Adotado na publicação POSA (*Pattern-Oriented Software Architecture – A System of Patterns*) de Buschmann et al. (1996).

- **Nome:** Nome e resumo do padrão

- **Também Conhecido Como:** Outros nomes para o padrão, se existirem.
- **Exemplo:** Um exemplo real demonstrando a existência do problema e a necessidade do uso do padrão.
- **Contexto:** Situações onde o padrão pode ser aplicado.
- **Problema:** O problema que o padrão corrige, incluindo suas forças.
- **Solução:** Descrição da solução do padrão.
- **Estrutura:** Detalhamento dos aspectos estruturais do padrão. Utiliza-se o diagrama de classes OMT e cartões CRC para cada componente.
- **Dinâmicas:** Descrição do comportamento do padrão.
- **Implementação:** Orientações para a implementação do padrão.
- **Exemplo:** Discussão de detalhes importantes que não foram discutidos nas seções anteriores.
- **Variantes:** Descrição breve de variações ou especializações do padrão.
- **Usos Conhecidos:** Descrição de exemplos reais
- **Consequências:** Descreve benefícios e fardos em potencial.
- **Veja Também:** Referências à padrões que resolvem problemas similares e que possam refinar o padrão.

4 DESENVOLVIMENTO DE PADRÕES

Este trabalho consiste no desenvolvimento de um catálogo de padrões de segurança da informação, com todas as suas respectivas implementações na linguagem de programação C#.

Foram feitas pesquisas bibliográficas e exploratórias sobre segurança da informação, um estudo consistente sobre o conceito de padrões e buscas de artigos acadêmicos e profissionais relacionadas à padrões de segurança. Desta forma, os padrões de segurança sugeridos pelo autor baseiam-se em modelos e estrutura de padrões consolidados que visam garantir a disponibilidade, a confidencialidade e integridade da informação e, adicionalmente, gestão de identidade e irretratabilidade.

Nesse sentido, Lea (200-? apud Blakley e Heath, 2004) criou uma *checklist* para identificar um bom padrão e que algumas destas serviram de norteammento para este trabalho:

- Descreve um único tipo de problema.
- Descreve o contexto em que o problema ocorre.
- Descreve a solução como uma *constructable software entity*.
- Descreve os *design steps* ou regras para a construção da solução
- Descreve as forças que levam à solução
- Descreve como a solução soluciona as forças
- Descreve detalhes que podem ou não variar
- Descreve um exemplo

Os padrões deste trabalho seguirão a seguinte forma:

- Nome
- Intenção
- Contexto
- Problema
- Forças
- Solução
- Exemplo
- Consequências
- Implementação

Os padrões apresentados neste trabalho estão resumidos no quadro 5.

Quadro 5 – Padrões Desenvolvidos

Nome	Função
Senha Segura	Criação e Gerenciamento de Senha
Autenticação	Gerenciamento de Identidade de Usuários
Registro de Ocorrência	Registro de ocorrências
Replicação de Dados	Backup de dados do sistema
Auditoria	Módulo de auditoria de registros
Proteção de Dados	Encriptação de dados sensíveis
Autorização de Usuário	Autorização de Recursos ao usuário
Gerenciamento de Chaves Criptográficas	Gerenciamento de Chaves Criptográficas
<i>Timeout</i> de Sessão	<i>Logout</i> de usuários inativos

Fonte: Elaborado pelo Autor

4.1 Senha Segura

Baseando-se no *pattern Password Design and Use* de Roser (2012), criou-se o padrão abaixo:

Nome: Senha Segura.

Intenção: Elucidar uma forma de criar e gerenciar senhas.

Contexto: Este *pattern* pode ser utilizado por quaisquer *softwares* que necessitam de autenticação por senha.

Problema: Senhas fracas possibilitam que pessoas não autorizadas possam se aproveitar da vulnerabilidade dessa condição.

Adicionalmente, senhas devem ser fáceis de serem lembradas, porém devem ser difíceis de serem adivinhadas.

Forças: Restrição na composição da senha; repetição de senhas; facilidade de lembrança; validade das senhas;

Solução: Deve-se levar em consideração algumas condições para a criação e o gerenciamento da senha.

- Seu comprimento mínimo
- Uso de números, caracteres especiais (ex: @, #, %), letras maiúsculas e minúsculas
- Tempo de uso da senha
- Adição de *salt*¹ à senha
- Método de segurança de armazenamento da senha

Exemplo: senha para acessar uma conta bancária na qual não se pode repetir números; senha de acesso a um *e-mail* onde não pode conter o login em seu corpo.

¹ Conjunto de caracteres aleatórios adicionados à senha antes do processamento pela função Hash. Desta forma, senhas iguais com *Salts* diferentes não gerarão o mesmo Hash.

Um exemplo prático para o uso das condições de senha dá-se:

- Comprimento mínimo de 10 caracteres
- Utilização obrigatória de pelo menos 01 caractere especial e 01 caractere maiúsculo e 01 caractere minúsculo
- Senha válida indefinidamente
- Deve-se adicionar o *salt* antes da senha ser criptografada
- A senha deve ser armazenada depois de criptografada por *Hash* SHA512

Consequências:

- Utilização de um algoritmo de *Hash* para armazenar a senha impede que o agente malicioso tenha acesso a conta comprometida
- A utilização do *salt* na senha impede que duas senhas iguais gerem o mesmo *hash*

Implementação:

As implementações dos métodos e classes deste padrão encontram-se nos apêndices A, J e K.

4.1.1 Exemplo de Uso

O uso prático deste padrão dá-se durante o registro do usuário, verificando, se a senha digitada condiz com as restrições de segurança aplicadas.

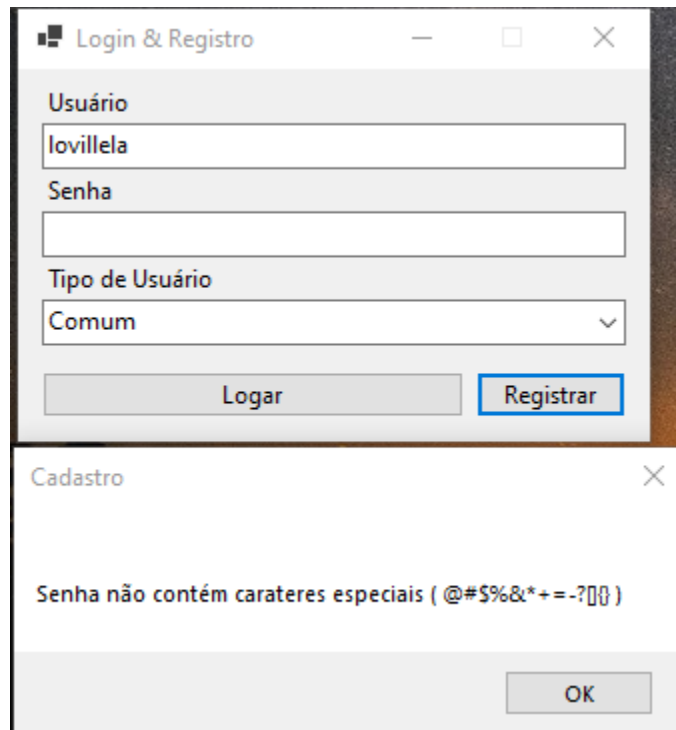
Na Figura 2, observa-se a tentativa do usuário em cadastrar uma senha com o comprimento menor do que o permitido. Entretanto, esta não é aceita e o usuário é informado do erro.

Figura 2 - Tentativa de Registro 1.

Fonte: Elaborado pelo Autor.

Na figura 3, observa-se a tentativa do usuário em cadastrar uma senha que não contém pelo menos um dos caracteres especiais definidos. Entretanto, esta não é aceita e o usuário é informado do erro.

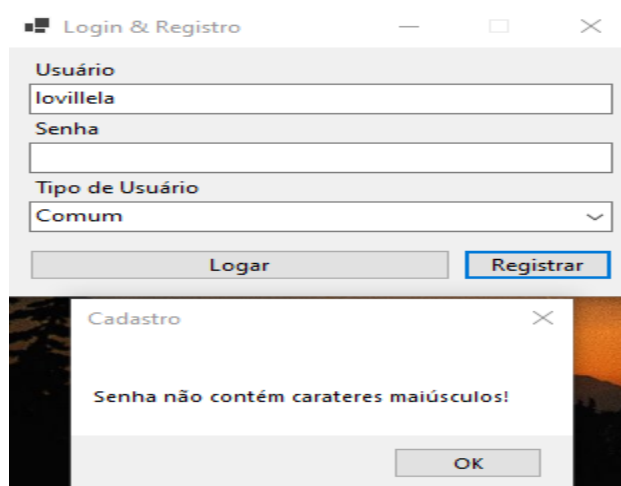
Figura 3 – Tentativa de Registro 2



Fonte: Elaborado pelo Autor

Na figura 4, observa-se a tentativa do usuário em cadastrar uma senha que não contém pelo menos um caractere maiúsculo. Entretanto, esta não é aceita e o usuário é informado do erro.

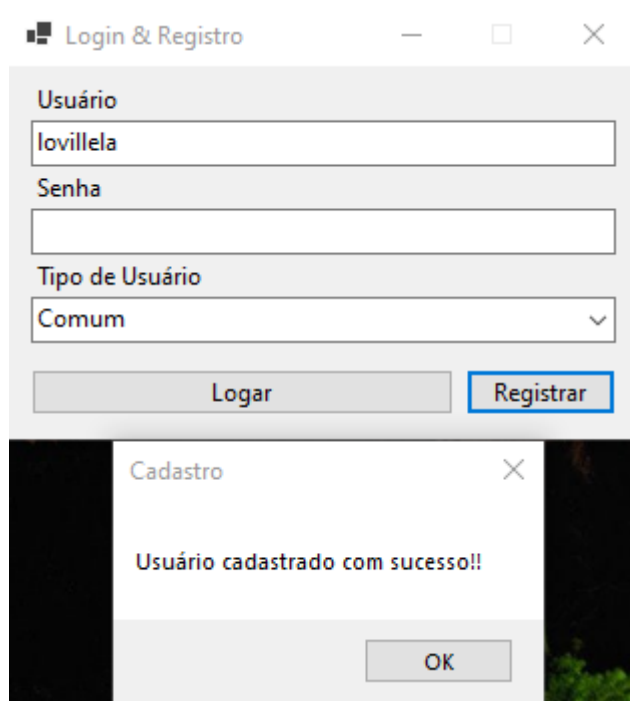
Figura 4 - Tentativa de Registro 3



Fonte: Elaborado pelo Autor

Na figura 5, observa-se que o usuário não violou as restrições de composição de senha. Logo, esta foi aceita como válida e será armazenada no sistema.

Figura 5 - Cadastro bem sucedido



Fonte: Elaborado pelo Autor.

Como mostra a Figura 5, o usuário conseguiu concluir o processo de cadastro.

O quadro 2 mostra as informações do usuário cadastrado

A coluna A armazena o nome de *login* do usuário; a coluna B armazena a senha depois criptografada pela função *Hash* SHA512; a coluna C armazena o *salt* associado à conta; e por fim, a coluna D armazena a função do usuário.

Quadro 2 – Arquivo de registro de Usuários

	A	B	C	D
1	lovillela	OVApG3S3mrXyVvK0Kly	:jg# fpFiPLN9P!c v9/gG8_ATmt	Comum

Fonte: Elaborado pelo Autor.

É importante salientar o processamento de dados efetuados pelas implementações nos apêndices J (Gerador de *Salt*) e K (*Hash* de Senhas). O resultado dá-se como pode ser visto no quadro 3.

Quadro 3 – Informações de Registro de Usuário

Usuário	lovillela
Senha	OVAp3S3mrxYV0KlySA3tyB6JMERmb1Tq1fVo TnkjAMugua6OOre3kxhb+QxD0IDiPPkRRoYLb8Ep6ORygjqA==
Salt	:jg# fpFiPLN9P!c v9/gG8_ATmtun!GSrMI8*5dtDNyE0+P} 7gCX=lBg9JNlbpWIA2@Wr=!U0dr18 YW{o_vGtR/yH zG@7immcn\$t;0g_[JJVu?6%?8QSKpW#çve#PwG![as8?JJ3rk
Tipo	Comum

Fonte: Elaborado pelo Autor.

Desta forma, a senha do usuário nunca é armazenada em seu texto original. Isto ocorre, pois o programa de registro utiliza as implementações dos Apêndices J e K. Logo, pode-se resumir o processo no passo-a-passo abaixo.

1. A senha é enviada para processamento.
2. O algoritmo do Apêndice J gera um *salt* aleatório.
3. Os caracteres gerados no passo 2 são adicionados à senha.
4. A senha com *salt* é enviada ao algoritmo do apêndice K e criptografada em *Hash* SHA512.
5. As informações de usuário, senha, *salt* (gerado no passo 2) e tipo de usuário são armazenadas.

4.2 Autenticação

Baseando-se no *pattern Authenticator* de Dangler (2013) e no *pattern Authenticator* de Roser (2012) criou-se o seguinte padrão:

Nome: Autenticação.

Intenção: Criar um sistema de gestão de identidade de usuários, garantido que apenas o indivíduo legítimo acesse o sistema.

Contexto: Para acessar o sistema, usuários necessitam comprar sua identidade.

Problema: Em sistemas informáticos com múltiplos usuários deve-se prover uma maneira de identificar um usuário específico.

Forças: fraude de identidade; *phishing*; comprovação de identidade;

Solução: Este *pattern* propõe a implementação de um sistema de confirmação de identidade no login do usuário.

Exemplo: Para acessar a sua conta, o usuário deve fornecer a senha corretamente.

Consequência:

- Apenas usuários, digitando suas respectivas senhas corretamente, terão acesso ao sistema.
- Uma forte política de senhas deve ser usada, para garantir que os usuários não utilizem senhas fáceis de serem adivinhadas.

Implementação:

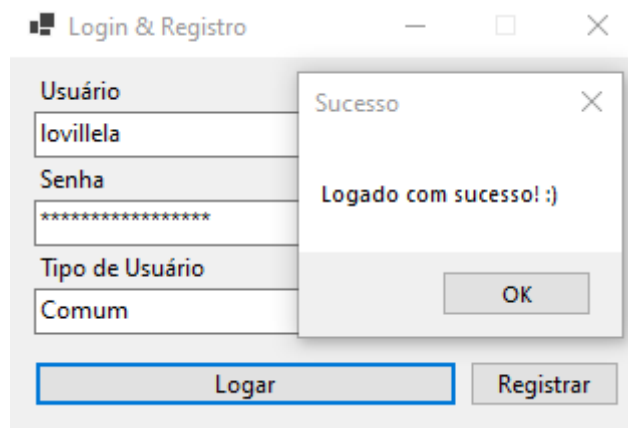
A implementação deste padrão encontra-se no Apêndice B.

4.2.1 Exemplo de Uso

O exemplo prático deste padrão dá-se durante o login do usuário, verificando se a senha informada é exatamente igual à armazenada no sistema.

Como pode ser visto na Figura 6, o login foi bem sucedido.

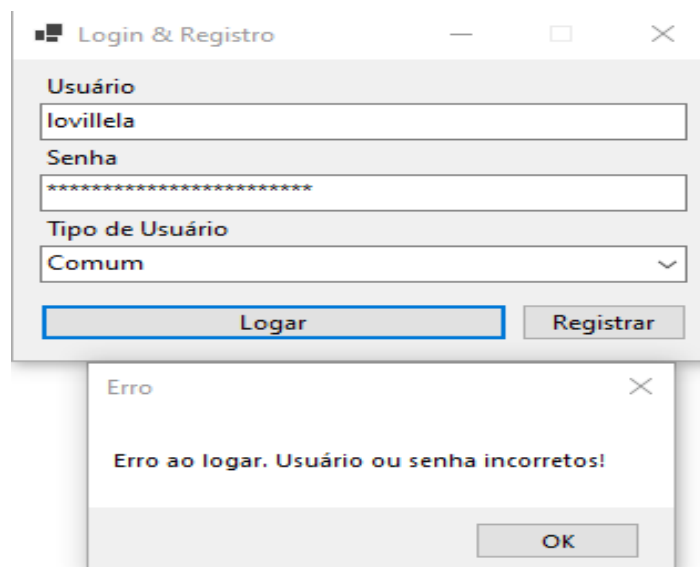
Figura 6 – Sucesso login



Fonte: Elaborado pelo Autor.

Porém, caso a senha digitada não corresponda à armazenada no sistema, o usuário é informado do erro, como pode ser visto na Figura 7. Observe que a mensagem de erro não discrimina o erro ocorrido. A mensagem “Erro ao logar. Usuário e senha incorretos.” não aponta qual o erro cometido pelo usuário (Senha errada ou login errado ou até mesmo ambos).

Figura 7 – Senha ou Usuário Incorreto



Fonte: O autor.

4.3 Registro de Ocorrência

Baseando-se no pattern *Secure Logger* de Roser (2012) e *Log for Audit* de Kienzle et al. (2002?), criou-se o padrão abaixo:

Nome: Registro de Ocorrência

Intenção: As ocorrências no sistema devem ser registradas. A alteração ou destruição do registro, até um período pré-determinado de tempo, deve ser impossível.

Contexto: É impossível prever e impedir acessos ou modificações não autorizadas. Desta forma, é necessário registrar as ocorrências para efetuar-se quaisquer auditorias ou investigações para identificar os responsáveis.

Problema: Aplicações confiáveis devem ter um registro de ocorrências imutável e centralizado. Estas informações podem ser necessárias para investigações forenses (criminais, ordens judiciais) e/ou auditoria (tanto interna quanto independente). Tais registros devem ser acessado somente por usuários autorizados.

Forças: auditoria; investigação forense; decisão judicial;

Solução: Dados de registro gerados pelo sistema são armazenados em um local centralizado, seguro e isolado.

Após armazenados por um período pré-determinado de tempo, os registros serão destruídos.

Exemplo: No sistema de registro de pacientes em um hospital, estão armazenados dados sigilosos como resultados de exames e prontuários médicos.

Certas informações (como o diagnóstico positivo para HIV, por exemplo) podem causar transtorno à um indivíduo caso sejam publicadas.

O registro de ocorrência deve armazenar quem acessou um arquivo ou sistema específico registrando as seguintes informações:

- Login de usuário

- Endereço MAC
- I.P. de acesso
- Hora de acesso

Consequências:

- Torna-se possível provar se um acesso não autorizado foi efetuado.
- Torna-se possível responsabilizar indivíduos
- Há a necessidade de medidas de segurança extensivas na tentativa de garantir a integridade dos registros.
- Registros que armazenam grandes quantidades de informações podem causar impactos negativos na performance do sistema

Implementação:

A implementação deste padrão encontra-se no Apêndice C.

4.3.1 Exemplo de Uso

Após as tentativas de login mostradas nas Figuras 6 e 7, bem-sucedidas ou não, juntamente com: o login do usuário em questão; seu I.P. público; o endereço M.A.C.; e a hora do acesso, estas informações são armazenadas em um arquivo de registro que pode ser visto no quadro 4.

Quadro 4 – Arquivo de Registro de Logins

	A	B	C	D	E
1	lovillela	08/07/2020 03:55	0.0.1.1	00:0A:A0:A0:0A:AA	VERDADEIRO
2	lovillela	08/07/2020 04:13	0.0.1.1	00:0A:A0:A0:0A:AA	FALSO

Fonte: Elaborado pelo Autor

No quadro 4, pode-se observar que as seguintes informações de login foram armazenadas: na coluna A o login do usuário; na coluna B a data e a hora da tentativa de login; na coluna C o I.P. público da conexão utilizada; na coluna D o endereço MAC físico da máquina utilizada; e na coluna E o resultado da tentativa de login.

É importante salientar que os dados das colunas C e D do Quadro 4 foram anonimizados, pois tratam-se de informações privadas do autor.

4.4 Replicação de Dados

Baseando-se no conceito fundamental de integridade dos dados, criou-se o padrão abaixo:

Nome: Replicação de Dados

Intenção: Todos os dados do sistema devem ser replicados na eventualidade de sua destruição (acidental ou não) ou alteração indesejada.

Problema: O valor das informações armazenadas em um sistema de informação é altíssimo. Desta forma, organizações em posse destas são alvo de ataques diários.

Além disso, hardware com uma falha física ou um sistema operacional corrompido podem acarretar na perda de informações.

Adicionalmente, uma alteração deve ser reversível.

Forças: perda de dados; acidentes; falhas de hardware; ataques;

Exemplo: O sistema de armazenamento de dados de uma empresa é, possivelmente, seu ativo intangível mais valioso. Desta forma, tal deve ser protegido de alterações ou ataques.

Solução: Este *pattern* propõe a implementação de um sistema de *backup* de dados automatizado ou manual com a possibilidade de *rollback*.

Consequência:

- Garantia da integridade dos dados.
- Recuperação em caso de falha
- Aumento de custos com infraestrutura

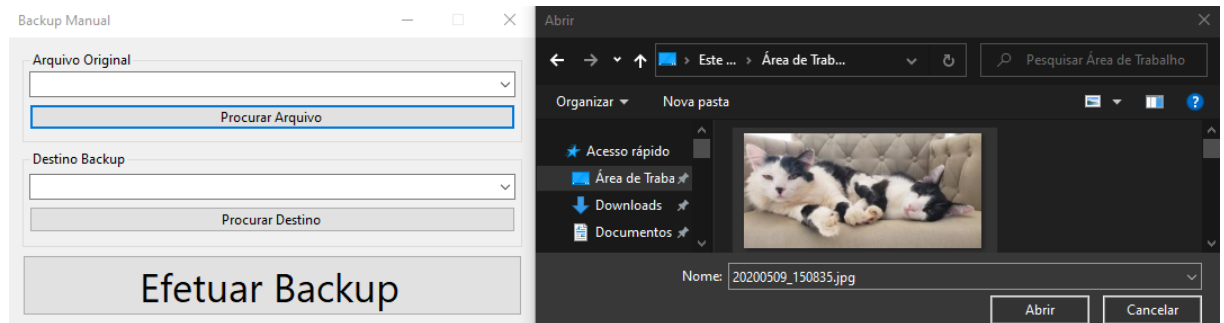
Implementação:

A implementação deste padrão encontra-se no Apêndice D.

4.4.1 Exemplo de Uso

O exemplo de uso deste padrão dá-se durante o *backup* de um arquivo escolhido pelo usuário, como pode ser visto na Figura 8

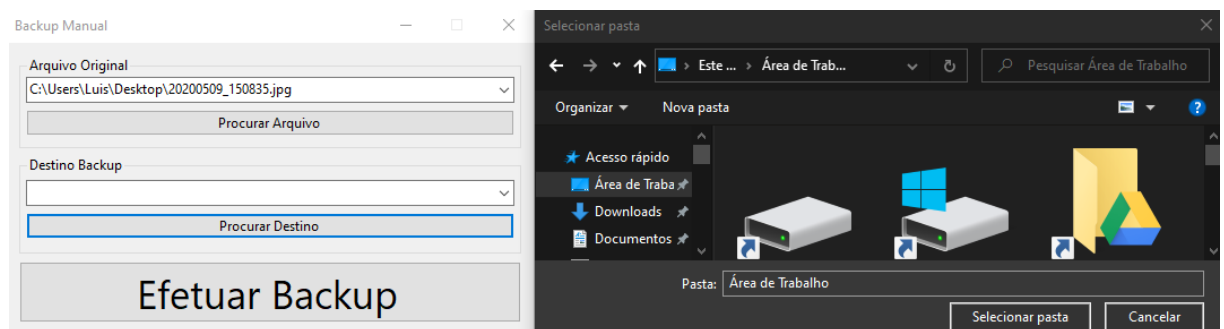
Figura 8 – Seleção de Arquivo para Backup



Fonte: Elaborado pelo Autor.

Selecionado o arquivo, o usuário seleciona o destino do backup, como visto na Figura 9.

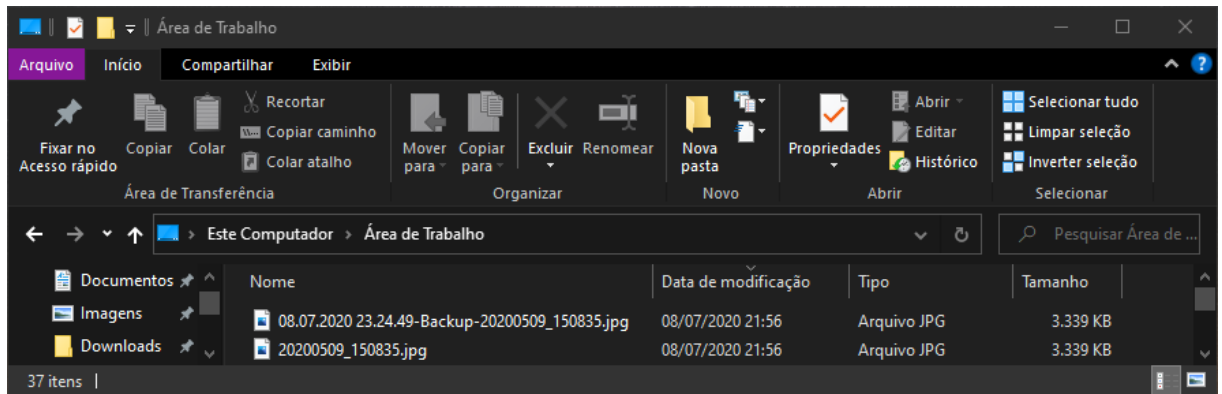
Figura 9 – Seleção de Destino de Backup



Fonte: Elaborado pelo Autor.

Ao pressionar-se o botão “Efetuar Backup”, um novo arquivo é criado no destino selecionado, adicionando-se ao nome do mesmo a Data e Horários do momento do *Backup*, como visto na Figura 10.

Figura 10 – Resultado do Backup



Fonte: Elaborado pelo Autor.

4.5 Auditoria

Baseando-se no pattern *Audit Requirements* de Schumacher, M. et al. (2006) criou-se o padrão abaixo:

Nome: Auditoria

Intenção: Este *pattern* propõe a criação de um módulo/subsistema de auditoria em um sistema informação

Problema: Nem todos os erros e ações maliciosos em um sistema de informação podem ser evitados. Desta forma, deve existir um módulo de auditoria para avaliar a responsabilidade da ação.

Forças: investigação; fraude; registros;

Solução: Implementar um módulo/subsistema para inspecionar as ações dos usuários.

Exemplo: Em um sistema de análise de crédito, o computador analisa e avalia o crédito de um cliente automaticamente.

Entretanto, o funcionário responsável pode alterar a avaliação de crédito do cliente (de não aprovado para aprovado; de um valor pequeno para um valor maior)

Consequência: Individualização do agente malicioso; não-repúdio

Implementação:

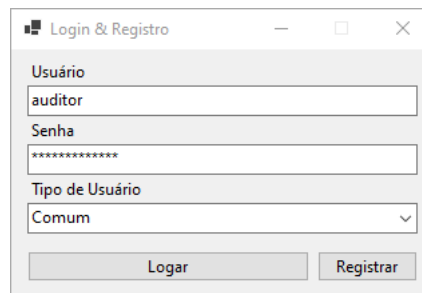
Não há implementação específica para este padrão. O auditor é um usuário do sistema com acesso à dados/funções diferentes dos usuários comuns. Desta forma, tal acesso é permitido ou bloqueado no nível da aplicação

.

4.5.1 Exemplo de Uso

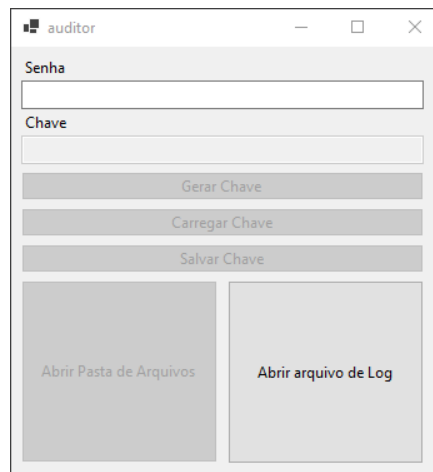
O exemplo de uso deste padrão dá-se como visto nas Figuras 11 e 12. Utilizando-se o padrão 4.1 Autenticação e 4.7 Autorização, o auditor acessa o sistema e tem acesso aos recursos exclusivos de sua função.

Figura 11 – Tela de *Login* do Auditor

A interface de login e registro do auditor. O formulário contém campos para 'Usuário' (preenchido com 'auditor'), 'Senha' (mascarada com pontos), e 'Tipo de Usuário' (menu suspenso com 'Comum' selecionado). Abaixo dos campos há dois botões: 'Logar' e 'Registrar'.

Fonte: Elaborado pelo Autor.

Figura 12 – Janela de Interação do Auditor

A janela de interação do auditor, intitulada 'auditor'. Ela contém campos para 'Senha' e 'Chave'. Abaixo desses campos, há três botões desativados: 'Gerar Chave', 'Carregar Chave' e 'Salvar Chave'. Na base da janela, há dois botões maiores: 'Abrir Pasta de Arquivos' e 'Abrir arquivo de Log'.

Fonte: Elaborado pelo Autor.

Como pode-se observar, os botões “Gerar Chave”, “Carregar Chave”, “Salvar Chave”, “Abrir Pasta de Arquivos” e o campo de entrada “Chave” estão desativados pois sua utilização não condiz com a função de auditor.

Pressionando-se o botão “Abrir arquivo de *Log*” o auditor acessa as informações contidas, como visto na Figura 13. É importante salientar que o auditor não possui acesso direto ao arquivo de *log*, mas apenas às informações contidas neste.

Figura 13 – Janela de Leitura de *Logs*

The screenshot shows two windows from an application named 'auditor'. The 'FormLeituraDeLogs' window is active and displays a table of login attempts. The table has five columns: 'Usuário', 'Horário', 'I.P.', 'M.A.C.', and 'Logou'. The data shows multiple failed login attempts for the user 'lovillela' between 16:53:42 and 16:55:11 on 11/07/2020. The last entry for 'auditor' shows a successful login ('Logou' is 'True').

Usuário	Horário	I.P.	M.A.C.	Logou
lovillela	11/07/2020 16:53:42	17 .	2 .	False
lovillela	11/07/2020 16:53:46	17 .	2 .	False
lovillela	11/07/2020 16:53:51	17 .	2 .	False
lovillela	11/07/2020 16:53:53	17 .	2 .	False
lovillela	11/07/2020 16:53:55	17 .	2 .	False
lovillela	11/07/2020 16:54:48	17 .	2 .	False
lovillela	11/07/2020 16:54:50	17 .	2 .	False
lovillela	11/07/2020 16:54:52	17 .	2 .	False
lovillela	11/07/2020 16:54:59	17 .	2 .	False
lovillela	11/07/2020 16:55:11	17 .	2 .	False
auditor	11/07/2020 16:55:29	17 .	2 .	True

Fonte: Elaborado pelo Autor.

Como pode-se observar, foram efetuadas tentativas sucessivas de login na conta do usuário “lovillela”. Todas resultaram em “False”, ou seja, a senha incorreta fora informada. Adicionalmente, as tentativas ocorreram entre as 16:53:42 e 16:55:11, sendo improvável que o usuário legítimo tenha efetuado todos estes acessos em um período tão curto de tempo. Desta forma, em posse dessas informações, o auditor pode determinar uma tentativa de penetração contra contas específicas do sistema.

4.6 Proteção de Dados

Baseando-se no pattern *Encrypted Storage* de Kienzle et al. (2002) e o pattern *Encryption with user-managed keys* do catálogo *Privacy Patterns* do campus de Berkley da Universidade da Califórnia criou-se o padrão abaixo:

Nome: Proteção de Dados

Intenção: Este padrão propõe um método de proteção de dados sensíveis armazenados no sistema informático no evento do roubo dos mesmos.

Contexto: Este padrão pode ser utilizado por qualquer sistema informático que necessite do armazenamento seguro de dados sigilosos.

Problema: Apesar de todos os métodos utilizados para a proteção de um sistema informático (ex: senhas, *firewalls*) serem eficazes em suas funções, estes não são à prova de falhas (podem existir vulnerabilidades desconhecidas).

Desta forma, o roubo dados sigilosos como, por exemplo, laudos médicos, números de cartões de crédito podem causar transtornos a seus proprietários e danos extensivos às organizações que os armazenam.

Forças: acesso não autorizado; armazenamento de dados sigilosos; gestão de chaves criptográficas

Solução: Para este *pattern*, há duas soluções possíveis;

1. Encriptação efetuada pelo servidor

Nesta solução o processo de deciptação e encriptação são efetuados pelo servidor, e naturalmente as chaves de criptografia são armazenadas no servidor. Desta forma, esta solução deve ser utilizada quando há a necessidade de terceiros acessarem os dados.

- Utilizando seu *login* e senha próprios, através de uma conexão segura (ex: HTTPS) o usuário acessa o sistema

- O servidor carrega a chave criptográfica no módulo de encriptação/decriptação
- No caso de envio de dados ao sistema
- Os dados são enviados ao módulo de encriptação/decriptação
 - Os dados são encriptados
 - Os dados são armazenados
- No caso de acesso de dados do sistema
 - O servidor busca os dados
 - Os dados são decriptados
 - Os dados são enviados ao cliente

2. Encriptação efetuada pelo cliente

Nesta solução o processo de decriptação e encriptação são efetuados pelo cliente, e naturalmente as chaves de criptografia estão em posse do mesmo. Desta forma, esta solução deve ser utilizada quando há a necessidade restringir terceiros de acessarem os dados.

Exemplo: Dados armazenados pelos os usuários devem ser acessados apenas pelos mesmos, no caso dos dados já serem armazenados encriptados.

Consequência:

- Apenas usuários ou servidores em posse das chaves criptográficas podem acessar os dados
- Impacto no desempenho do sistema, devido ao custo computacional elevado para encriptar e decriptografar os dados.
- Garantia da confidencialidade da informação
- Impossibilidade de acesso aos dados na eventualidade da perda das chaves criptográficas

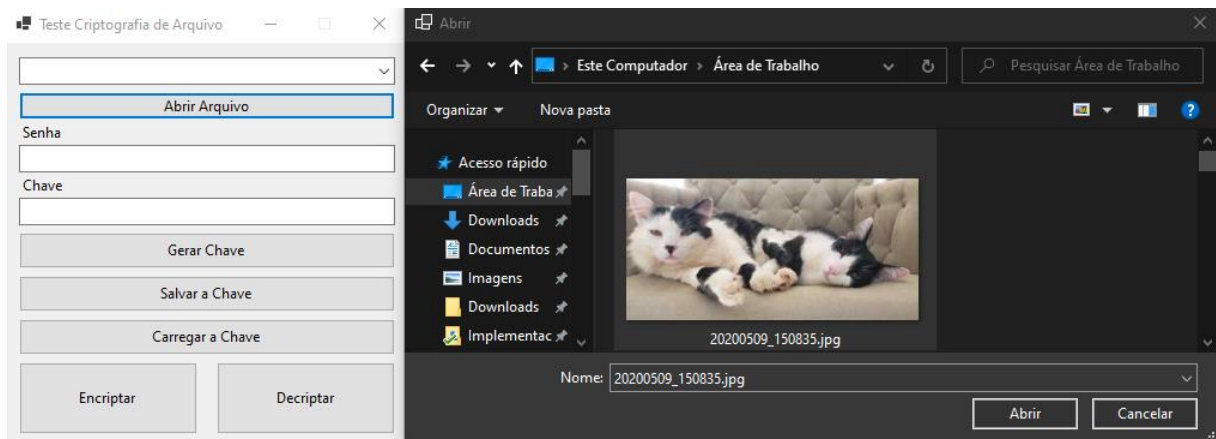
Implementação:

A implementação deste padrão encontra-se no Apêndice E.

4.6.1 Exemplo de Uso

O exemplo de uso deste padrão dá-se a como exemplificado na Figura 14, onde observa-se o usuário selecionando o arquivo que deseja criptografar.

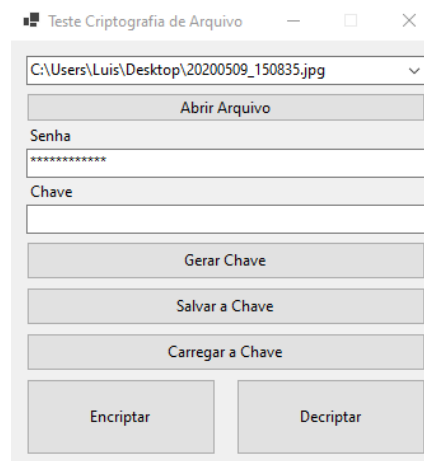
Figura 14 – Seleção de Arquivo para criptografia



Fonte: Elaborado pelo Autor.

Selecionado o arquivo desejado, o usuário digita a senha de sua escolha como visto na Figura 15.

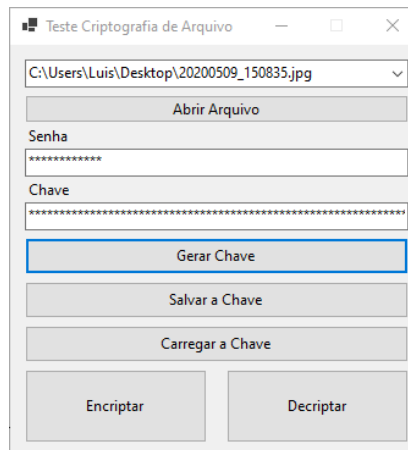
Figura 15 – Senha Digitada



Fonte: Elaborado pelo Autor..

O próximo passo deve-se gerar uma chave. Neste caso, é utilizada a função “Gerador de Salt” do Apêndice J. O resultado é observado na Figura 16

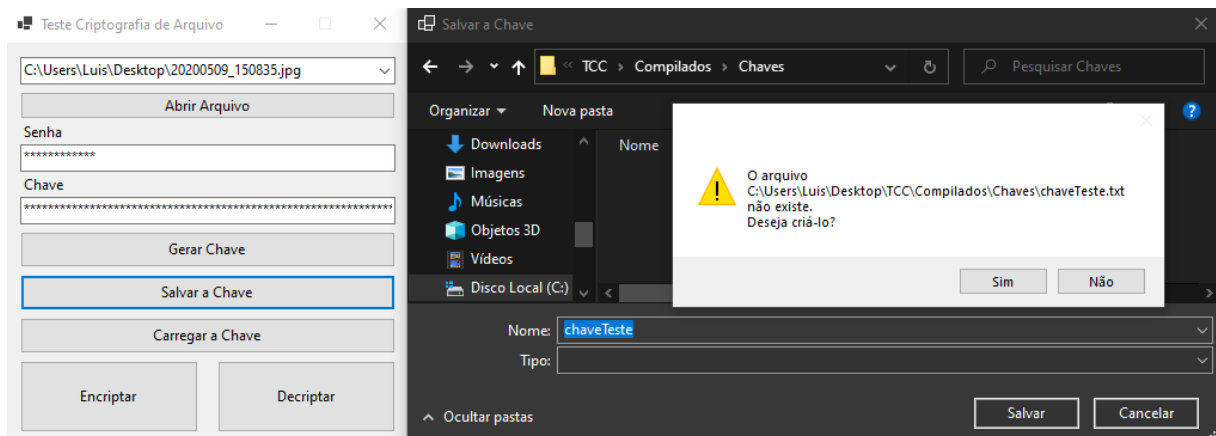
Figura 16 – Chave Gerada



Fonte: Elaborado pelo Autor.

Concluída a geração da chave, o usuário deve salvá-la. Este passo pode ser observado na Figura 17.

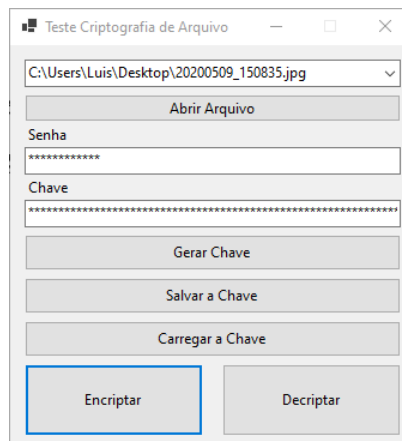
Figura 17 – Salvar a chave



Fonte: Elaborado pelo Autor.

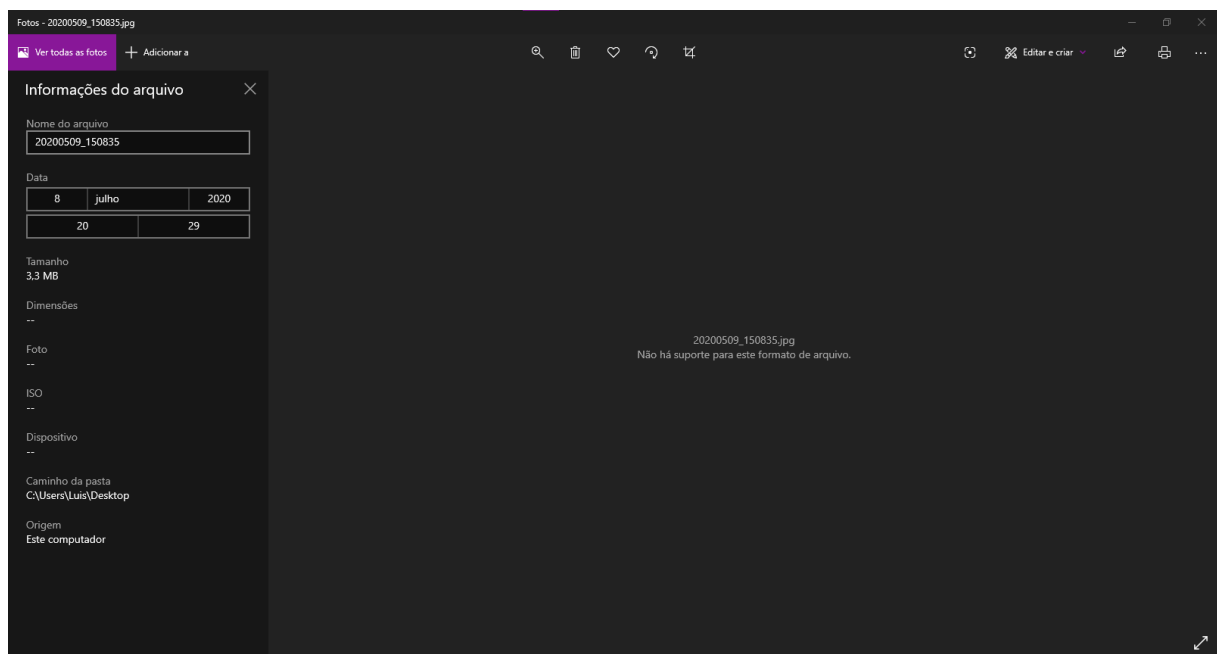
O próximo passo a ser seguido, é encriptar a imagem escolhida ao apertar-se o botão “Encriptar”, como pode ser visto na Figura 18 e o resultado na Figura 19.

Figura 18 – Encriptando o arquivo



Fonte: Elaborado pelo Autor.

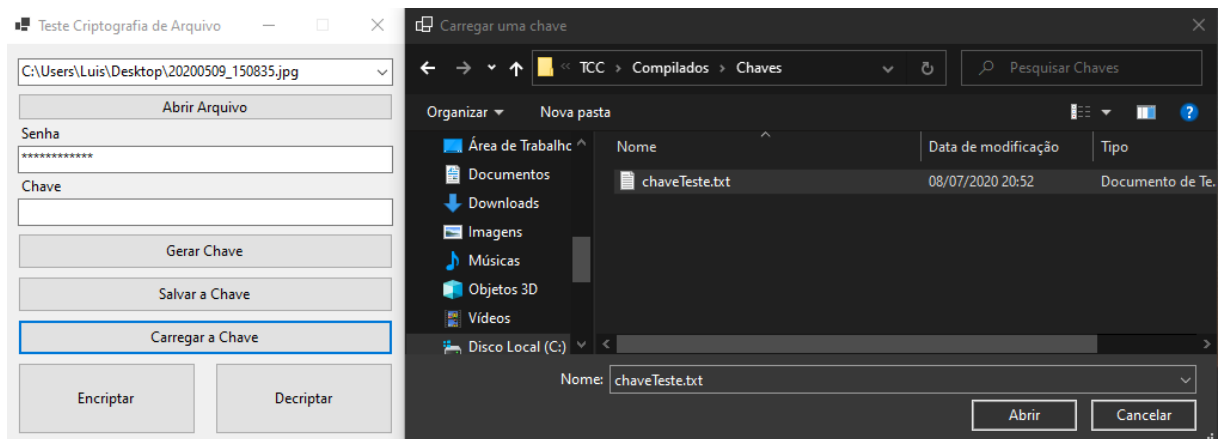
Figura 19 – Arquivo encriptado



Fonte: Elaborado pelo Autor.

Para reverter-se o processo de criptografia, o usuário deve informar a senha e a chave utilizadas anteriormente. Apenas a chave é armazenada, como visto na Figura 19. Desta forma, para recuperá-la o usuário deve abrir o arquivo da chave, como visto na Figura 20.

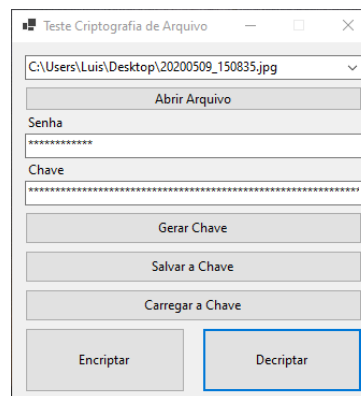
Figura 20 – Carregamento da Chave



Fonte: Elaborado pelo Autor.

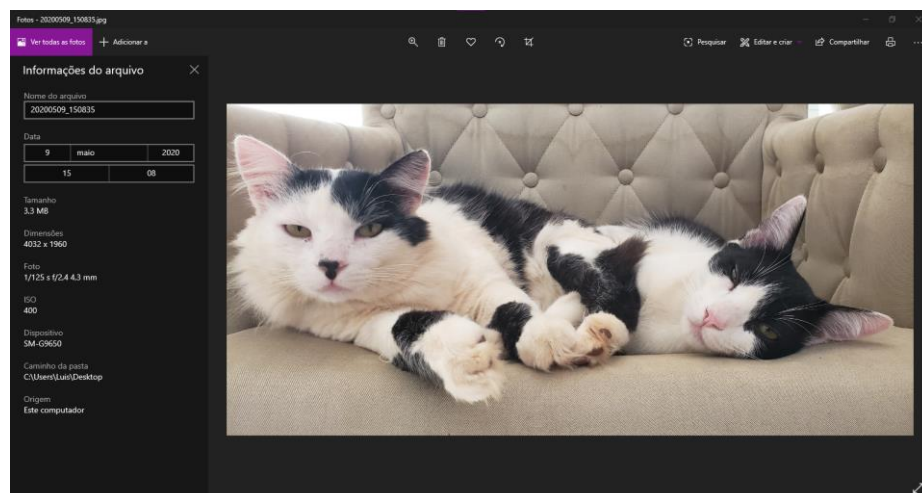
Para finalizar o processo de decryptografia, basta apertar o botão “Decryptar” como visto nas Figuras 21 e 22.

Figura 21 – Decryptar arquivo



Fonte: Elaborado pelo Autor.

Figura 22- Arquivo decryptado



Fonte: Elaborado pelo Autor..

4.7 Autorização de Usuário

Baseando-se no *pattern Authorization* de Roser (2012) criou-se o seguinte padrão:

Nome: Autorização de Usuário

Intenção: Criar um módulo de autorização de acesso à recursos e/ou dados do sistema.

Contexto: Deve ser utilizado na necessidade individualizar o acesso a dados e subsistemas.

Problema: Cada usuário possui uma função no sistema informático. Assim sendo, este deve acessar apenas os módulos, informações e subsistemas pertinentes a sua função. (Privilégio Mínimo)

Forças: isolamento de módulos e dados do sistema; mecanismo de autorização independente;

Solução: Atrelar às contas de usuário, os direitos de acesso à módulos e dados específicos.

Exemplo: Um Sistema de gerenciamento de uma empresa com diversos setores e filiais.

Os usuário do setor financeiro devem acesso restrito ou bloqueado às informações do setor R.H. Adicionalmente, estes devem ter acesso apenas aos dados de suas filiais e não os da empresa inteira.

Consequência:

- Garantia de isolamento de dados e subsistemas
- Aumento da complexidade de administração

Implementação:

O código-fonte deste padrão encontra-se no Apêndice F

4.7.1 Exemplo de Uso

Efetuando-se um login bem-sucedido, como visto em 4.2.1, o sistema passará para a fase de autorização, bloqueando ou liberando acesso aos recursos que convém ao usuário como visto na Figura 23.

Figura 23 – Janela de interação do usuário autorizado

Fonte: Elaborado pelo Autor.

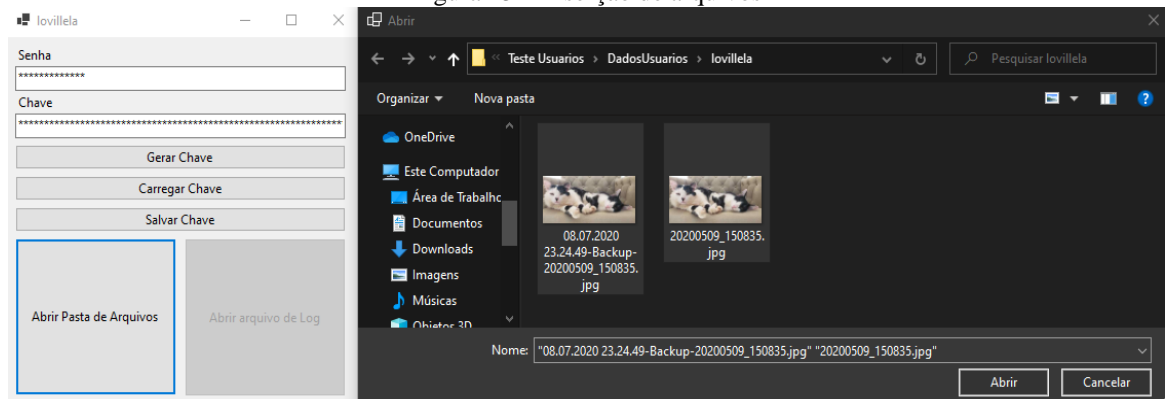
Deve-se notar que o botão “Abrir arquivo de *Log*” está desativado, pois este é de uso exclusivo do auditor. Logo, a única função permitida ao usuário é abrir a pasta pessoal de arquivos, informando a senha desejada e gerando a chave, como visto na Figura 24 abaixo.

Figura 24 – *Logs* desativados

Fonte Elaborado pelo Autor.

Em seguida, o usuário abre sua pasta pessoal de arquivos, e insere os arquivos desejados como visto na Figura 25.

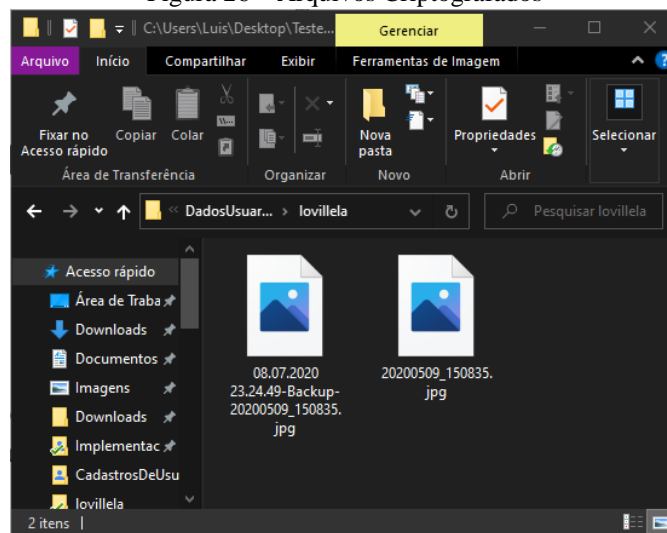
Figura 25 – Inserção de arquivos



Fonte: Elaborado pelo Autor.

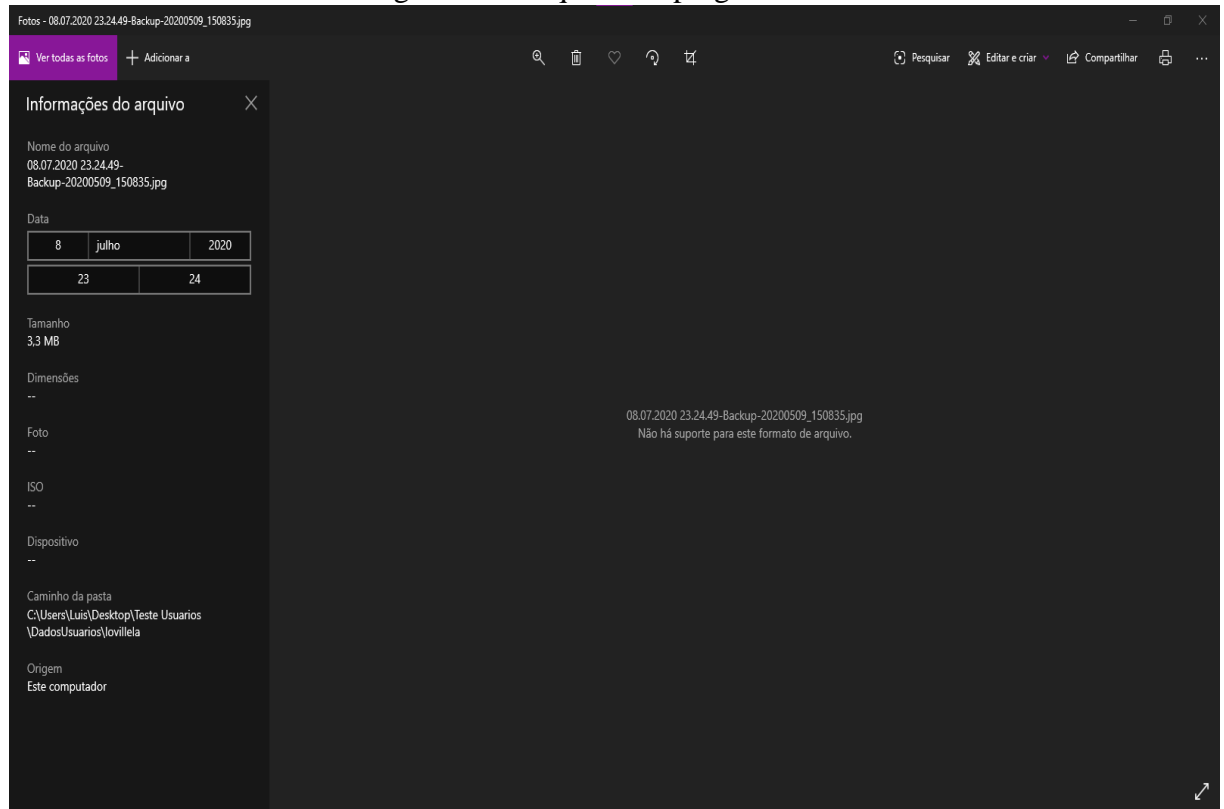
Ao pressionar o botão abrir, os arquivos da pasta são criptografados. Este resultado é observado nas Figuras 26 e 27.

Figura 26 – Arquivos Criptografados



Fonte: Elaborado pelo Autor.

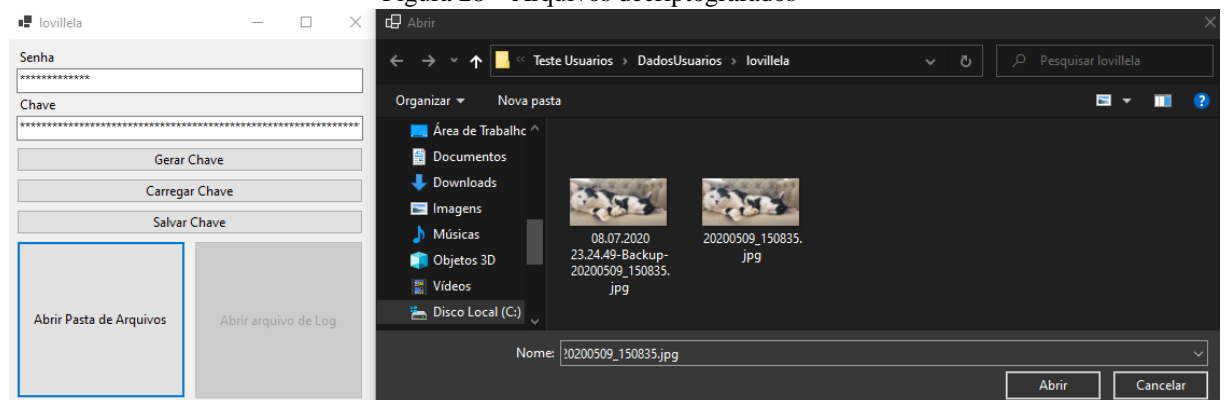
Figura 27 - Arquivo Criptografado



Fonte: Elaborado pelo Autor.

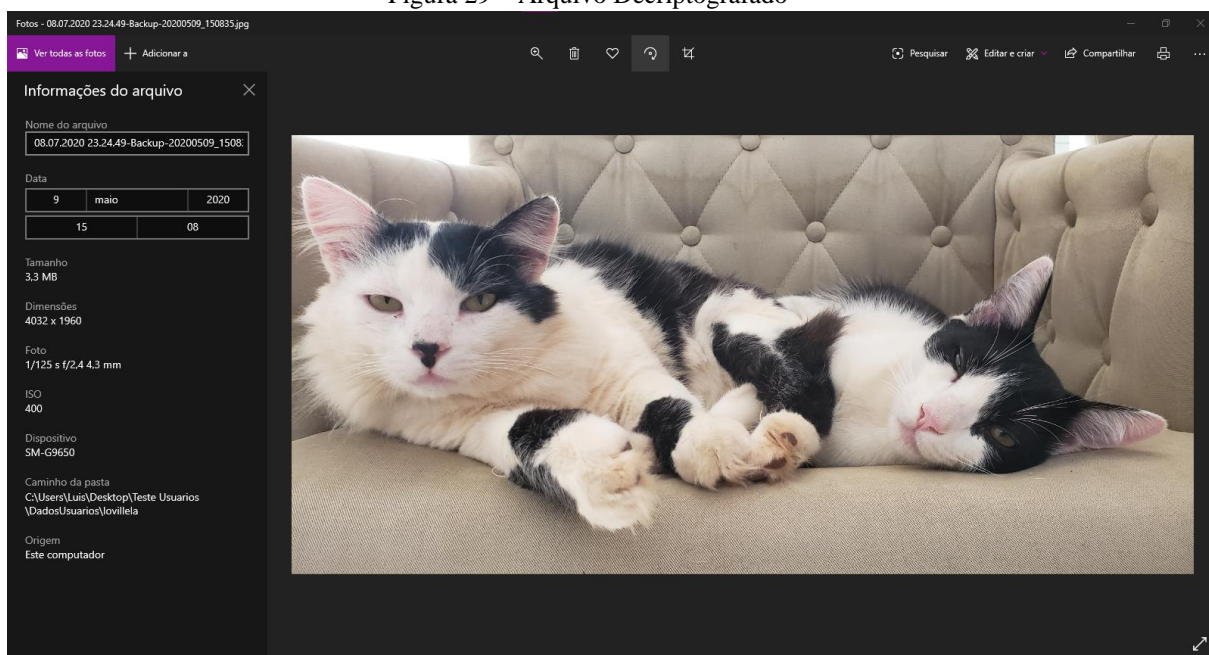
Para descriptografar os arquivos de sua pasta pessoal, basta ao usuário informar novamente a senha e chave utilizadas, como visto nas Figuras 28 e 29.

Figura 28 – Arquivos descriptografados



Fonte: Elaborado pelo Autor.

Figura 29 – Arquivo Decriptografado



Fonte: Elaborado pelo Autor.

4.8 Gerenciamento de Chaves Criptográficas

Baseando-se no pattern *Encryption with user-managed keys* do catálogo *Privacy Patterns* do campus de Berkley da Universidade da Califórnia criou-se o padrão abaixo:

Nome: Gerenciamento de Chaves Criptográficas

Intenção: Este *pattern* propõe como gerenciar a troca, armazenamento e destruição das chaves criptográficas.

Contexto: Sistemas informáticos que armazenem chaves criptográficas, e consequentemente informações sigilosas, devem utilizar este padrão.

Problema: As chaves criptográficas devem permanecer secretas.

Forças: chaves fracas; algoritmos antigos; não rotativo; armazenamento seguro; não destruição acidental; destruição de chaves;

Solução: Implementar um módulo de gerenciamento de chaves criptográficas.

Exemplo:

- A chave é requisitada
- O Gerenciamento de Chaves Criptográficas envia a chave

Consequência: Módulo exclusivo irá gerenciar o controle das chaves criptográficas.

Implementação:

A implementação deste padrão encontra-se no Apêndice G.

4.8.1 Exemplo de Uso:

- A senha informada pelo usuário, e a chave gerada são concatenadas e alimentadas para a função Hash SHA256.
 - Logo, a chave utilizada pelo algoritmo AES é Hash256(Senha + Chave Gerada).

- A chave gerada é alimentada isoladamente para a função Hash SHA256.
 - Logo, o Vetor Inicialização utilizado é pelo algoritmo AES é Hash256(Chave).

É importante salientar que nesta implementação a chave de criptografia nunca é armazenada, esta é calculada no momento da encriptação e decríptação. O vetor inicialização fica em posse do usuário.

4.9 Timeout de Sessão

Baseando-se nos conceitos de Tela Lima e Mesa Limpa, criou-se o padrão abaixo:

Nome: *Timeout* de sessão

Intenção: Este *pattern* propõe como gerenciar o *logout* automático de usuários inativos no sistema.

Contexto: Computadores em espaços compartilhados podem ser acessados por indivíduos não autorizados.

Problema: Muitos usuários esquecem de desconectar suas contas ao terminar de usá-las ou a se absterem por um curto período.

Esta brecha permite que agentes maliciosos com acesso físico ao computador aproveitem-se desta situação.

Forças: inatividade; acesso físico; mesa limpa; tela limpa

Solução: Depois de entrar no sistema, um cronômetro inicia uma contagem regressiva. Quando tal contagem se encerrar, o usuário será desconectado do sistema automaticamente. Ademais, esta mesma solução deve ser aplicada ao sistema operacional.

Em adição, é interessante monitorar o ambiente de trabalho através de câmeras de segurança.

Exemplo:

- O administrador levanta-se de sua estação de trabalho para almoçar
- Ao perceber que o computador estava desprotegido, a secretária aproveita-se da situação, causando suas ações maliciosas.
- Dias depois, vários dados críticos foram destruídos, incluindo os *backups*.
- O administrador perde o seu posto, pois é impossível provar que ele não executou o ato.

Consequência:

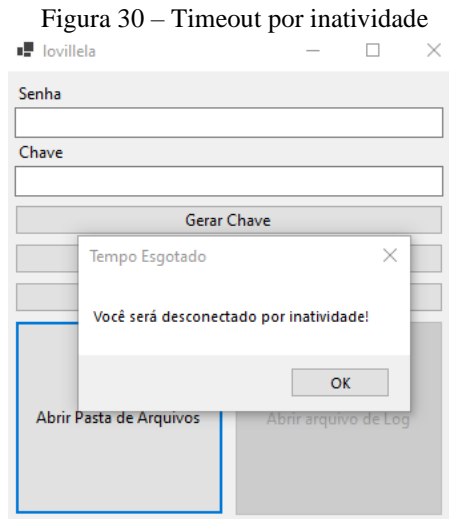
- Aumento da confiabilidade da segurança do sistema.
- Medir o tempo correto para o timeout é um grande desafio.
- É interessante mudar as políticas de usuário do administrador.
Exemplo: apenas dois ou mais administradores podem deletar arquivos de *backup*.

Implementação:

A implementação deste padrão encontra-se no Apêndice H.

4.9.1 Exemplo de Uso

O exemplo de uso deste padrão dá-se como visto na Figura 30.



Fonte: Elaborado pelo Autor.

Após efetuado o *login* bem sucedido, e a janela de interação é mostrada ao usuário, um contador interno é ligado. Após este chegar a zero, devido a inatividade do usuário, um aviso será mostrado, e o usuário desconectado.

5 CONCLUSÃO

Foram apresentados, ao total, 09 Padrões de Segurança da Informação, que visam garantir os conceitos fundamentais da Segurança da Informação (Confidencialidade, Integridade e Disponibilidade).

Por meio deste catálogo de padrões, permite-se garantir a autenticidade de acesso à uma conta em um sistema informático; a criação de senha fortes e gerenciamento de armazenamento das mesma; a catalogação dos registros de acesso, bem como um módulo de auditoria de tais registros; o gerenciamento de liberação ou bloqueio de recursos à usuários; a criptografia de dados de propriedade de terceiros, juntamente com o gerenciamento de chaves criptográficas utilizadas; e adicionalmente, a desconexão automática de usuários inativos.

O catálogo de padrões apresentado foi implementado e testado em projetos de *software* que se encontram em projetos reais, e durante o desenvolvimento e aplicação dos padrões, apresentou-se um bom ganho de produtividade.

Um ponto interessante é que sua utilização não é restrita a uma determinada linguagem de programação ou tecnologia, podendo ser aplicados em conjunto com diversas linguagens existentes no mercado.

Foi percebido também que sua utilização pode contribuir para a organização e manutenção de projetos relacionados à segurança de um sistema de informação, já que esses padrões se baseiam em baixo acoplamento e alta coesão entre as classes e padronização do código.

Além disso, com a padronização dos termos, as discussões técnicas são facilitadas, pois reduz a complexidade apresentando apenas o nome de um padrão de segurança em vez de ter que explicar todo o seu comportamento.

Também pode-se afirmar que a quantidade de erros é reduzida consideravelmente. Os padrões acabam sendo intuitivos, já que seu potencial permite a alta coesão entre os padrões, consequentemente há redução de custos com desenvolvimento de correções.

Como trabalhos futuros há a pretensão de expandir o número de padrões e criar uma taxonomia mais ampla para facilitar a utilização destes de forma mais expandida no que se refere a arquitetura dos sistemas de informação não limitando apenas ao *software*

Enfim, aplicar padrões, como já foi comprovado em diversas áreas da computação, e também através deste estudo, traz uma série de benefícios técnicos e analíticos aonde forem aplicados.

REFERÊNCIAS

ALEXANDER, Christopher; ISHIKAWA, Sara; SILVERSTEIN, Murray. **A Pattern Language**. Oxford University Press, 1979.

BASIN, D. **Applied Information Security**. Springer, 2011.

BARCALOW, Jeffrey; YODER, Joseph. **Architectural Patterns for Enabling Application Security**. Disponível em: <<https://www.idi.ntnu.no/emner/tdt4237/2007/yoder.pdf>>. Acesso em 29/04/2019

BLAKLEY, Bob; HEATH, Craig. **Security Designs Patterns**. Reading, Berkshire, UK. The Open Group, 2004. Disponível em: <https://pubs.opengroup.org/onlinepubs/9299969899/toc.pdf>. Acesso em: 10 mai. 2019

BUSCHMANN, Frank; MEUNIER, Regine; ROHNERT, Hans; SOMMERLAND, Peter; STAL, Michael. **Pattern- Oriented Software Architecture – A System of Patterns**. Vol. 1. John Wiley & Sons, 1996

BRAGA, Rosana V.; MALDONADO, José C.; GERMANO, Fernão S.; MASIERO, Paulo C.. **Padrões e Frameworks de Software**. Universidade de São Paulo. Disponível em: <http://conteudo.icmc.usp.br/pessoas/rtvb/apostila.pdf>. Acesso em: 08/05/2019.

COPLIEN, James O. *Software Design Patterns: Common Questions and Answers*. AT&T Bell Laboratories, 1998. Disponível em: https://pdfs.semanticscholar.org/9544/fccfd09a9315b29ced5bc1e69f572114b7ec.pdf?_ga=2.139690166.460569948.1592764695-468705456.1592764695

CRIPTOGRAFIA. **Dicionário Online de Língua Portuguesa**. Disponível em: <http://michaelis.uol.com.br/busca?r=0&f=0&t=0&palavra=criptografia> Acesso em: 05 mai 2019.

DANGLER, Jeremiah Y. Categorization of Security Design Patterns (2013). *Electronic Theses and Dissertations. Paper* 1119. Disponível em: <<https://dc.etsu.edu/etd/1119>>. Acesso em: 01 mai 2019.

DEVANBU, Premkumar T.; STUBBLEBINE, Stuart. *Software Engineering for Security: a Roadmap*. 2000. Disponível em: <<http://www0.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finaldevanbu.pdf>>. Acesso em 20 de mar 2019.

DHILLON, Gurpreet. *Principles of Information Systems Security: Text And Cases*. 1 ed. John Wiley & Sons, 2006.

DOBLE, Jim; MESZAROS, Gerard.. *In: BUSCHMAN, et al. A Pattern Language for Pattern Writing*. Boston: Addison-Wesley Longman Publishing Co., 1997.

DOUGHERTY, Chad; SAYRE, Kirk; SEACORD, Robert C.; SVOBODA, David, TOGASHI, Kazuya. **Secure Design Patterns**. [S. l.: s. n.], 2009. Disponível em: <<https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9115>>. Acesso em: 29 abr. 2019.

ENCRYPTION with user name-managed keys. Berkley University. Disponível em: <https://privacypatterns.org/patterns/Encryption-user-managed-keys>. Acesso em: 20 mai. 2019.

KIENZLE, Darrell M.; ELDER, Matthew C.; TYREE, David; EDWARDS-HEWITT, James. **Security Patterns Repository**. 2002. Disponível em: <http://www.scrypt.net/~celer/securitypatterns/repository.pdf>. Acesso em: 15 mai. 2019

GYSEN, Alan van. **How to clean desk and clear screen reduces your cyber risk**. 2018. Disponível em: <https://www.itweb.co.za/content/GxwQDq1A6j3qLPVo>. Acesso em: 02 mai. 2019

HAYRO. **The Three Goals of Cyber Security-CIA Triad Defined**. Disponível em: <<https://www.prefereditgroup.com/2019/08/27/the-three-goals-of-cyber-security-cia-triad-defined/>>. Acesso em: 29 ago. 2019

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 27000: Information technology — Security techniques — Information security management systems** — Overview and vocabulary. Genebra, 2018

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 27001: Information technology — Security techniques — Information security management systems** — Requirements. Genebra, 2013.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 27002: Information technology — Security techniques — Code of practice for information security controls**. Genebra, 2013

LAWRIE, Brown. STALLINGS, William. **Segurança de Computadores - Princípios e Práticas** - 2ª Ed. GEN LTC, 2014

LARMAN, Craig. **Utilizando UML e Padrões**. 3 ed. Bookman, 2007

LYNETT, Mike. **A History of Information Security From Past to Present**. Disponível em: <<https://blog.mesltd.ca/a-history-of-information-security-from-past-to-present>>. 2015. Acesso em 20 mar. 2019

NIELES, Michael; DEMPSEY, Kelley; PILLITTERI, Victoria. **An Introduction to Information Security: sp 800-12 rev. 1**. Information Technology Laboratory (Itl): Computer Security Division (CSD). Gaithersburg, Jun/2017. Applied Cybersecurity Division, Seção 800-12, Disponível em: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf>. Acesso em: 02 mai. 2019.

OLSON, Ryan; HOWARD, Richard; GRAHAM, James. **Cyber Security Essentials**. 1 ed. Routledge, 2010.

PELTIER, Thomas R.; PELTIER, Justin; BLACKLEY, John. **Information Security Fundamentals**. 1 ed. Auerbach Publications, 2005.

PONEMON INSTITUTE LLC. 2018 Cost of a Data Breach Study: Global Overview. 2018. Disponível em: <https://www.ibm.com/downloads/cas/861MNWN2>>. Acesso em 18 mar 2019

ROMANOSKY, Sasha. **Security Design Patterns** Part 1. Disponível em: <https://www.cgisecurity.com/lib/securityDesignPatterns.html>>. Acesso em 29 abr. 2019

ROSER, Florian. **Security Design Patterns in Software Engineering**. 2012. Disponível em: https://www.researchgate.net/publication/259561515_Security_Design_Patterns_in_Software_Engineering_-_Overview_and_Example. Acesso em: 10 mai 2019

SCHUMACHER, Markus; FERNANDEZ-BUGLIONI, Eduardo; HYBERTSON, Duane; BUSCHMAN, Frank; SOMMERLAND, Peter. **Security Patterns Integrating Security and Systems Engineering**. 1 ed. John Wiley & Sons, 2005.

SHIREY, R. **Internet Security Glossary**. 2000. Disponível em: <https://www.ietf.org/rfc/rfc2828.txt>>. Acesso em: 04 mai.2019

SOUPPAYA, Murugiah; SCARFONE, Karen. **Guide to Malware Incident Prevention and Handling for Desktops and Laptops**. National Institute of Standards and Technology. 2013. Disponível em: <https://csrc.nist.gov/publications/detail/sp/800-83/rev-1/final>. Acesso em: 02 mai 2019

STAMP, Mark. **Information Security: Principles and Practice**. 2 ed. Wiley-Blackwell, 2011.

ZETTER, Kim. **Diginotar Files For Bankruptcy In Wake Of Devastating Hack**. 2011. Disponível em:<<https://www.wired.com/2011/09/diginotar-bankruptcy/>>. Acesso em 20 mar 2019

YOSHIOKA, Nobukazu ; WASHIZAKI, Hironori; MARUYAMA Katsuhisa. **A survey on security patterns. Special issue: The future of software engineering for security and privacy**. Progress in Informatics, No. 5, pp.35–47, (2008). DOI: 10.2201/NiiPi.2008.5.5

APÊNDICE A – Senha Segura

Neste apêndice A, encontra-se o Código Fonte do Padrão Senha Segura na linguagem C#.

```
using Biblioteca_Implementacoes.Utilitarios;
using System;
using System.Security;

namespace Biblioteca_Implementacoes
{
    internal static class SenhaSegura
    {
        private static int ComprimentoMinimo =
        Int32.Parse(Properties.Resources.comprimentoMinimo);
        private static readonly string CaracteresEspeciais =
        Properties.Resources.caracteresEspeciais;
        private static readonly string senhaPequena = "\nSenha MENOR do que o comprimento
        mínimo! ( " + ComprimentoMinimo + " )";
        private static readonly string senhaSemEspeciais = "\nSenha não contém caracteres
        especiais ( " + CaracteresEspeciais + " )";
        private static readonly string senhaSemMaiusculo = "\nSenha não contém caracteres
        maiúsculos!";
        private static readonly string sucesso = "\nSenha válida!";

        internal static (int, string) CriarSenha(ref string senhaInformada)
        {
            bool possuiMaiusculo = false;
            bool possuiEspecial = false;

            if (senhaInformada.Length < ComprimentoMinimo)
            {
                senhaInformada = string.Empty;
                return (1, senhaPequena); //vê o tamanho da senha
            }

            foreach (var caractere in senhaInformada)
            {
                foreach (var caractereEspecial in CaracteresEspeciais)
                {
                    if (caractere == caractereEspecial) //conta os caracteres especiais
                    {
                        possuiEspecial = true; //se encontrar 1, dá break!
                        break;
                    }
                }
            }

            if (possuiEspecial)
            {
                break;
            }
        }
    }
}
```

```

    }
}

if (!possuiEspecial)
{
    senhaInformada = string.Empty;
    return (2, senhaSemEspeciais);
}

foreach (var caractere in senhaInformada)
{
    if (char.IsLetter(caractere) && char.IsUpper(caractere)) //verifica se é letra e se é
    maiúscula
    {
        possuiMaiusculo = true; //se for, encerra o laço
        break;
    }
}

if (!possuiMaiusculo)
{
    senhaInformada = string.Empty;
    return (3, senhaSemMaiusculo);
}

return (0, sucesso);
}
}
}

```

APÊNDICE B – Autenticação

Neste apêndice B, encontra-se o Código Fonte do Padrão Autenticação na linguagem C#.

```
using System;
using System.Security;
using Biblioteca_Implementacoes.Utilitarios;

namespace Biblioteca_Implementacoes
{
    public static class Autenticacao
    {
        private static string timeStamp { get; set; } //hora do login, para ser armazenado no log!
        private static string endIP { get; set; } //Endereço I.P. da conexão, para ser armazenado
        no log!
        private static string endMAC { get; set; } //Endereço MAC da conexão, para ser
        armazenado no log!

        public static void AutenticarUsuario(string entradaUsuario, ref string entradaSenha,
            out bool res, out string tipoUsuario)
        {
            timeStamp = DateTime.Now.ToString();
            endIP = CaptaDadosConexao.CaptaIP();
            endMAC = CaptaDadosConexao.CaptaMAC();

            res = Logar(entradaUsuario, ref entradaSenha, timeStamp, endIP, endMAC);

            if (res)
            {
                tipoUsuario = Autorizacao.AutorizaUsuario(entradaUsuario);
            }
            else
            {
                tipoUsuario = "";
            }
        }

        private static bool Logar(string usuario, ref string senha, string timeStamp, string endIP,
            string endMAC)
        {
            return RegistroDeOcorrencia.RegistroLoginExcel(usuario, ref senha, timeStamp,
            endIP, endMAC);
        }
    }
}
```

APÊNDICE C – Registro De Ocorrência

Neste apêndice C, encontra-se o Código Fonte do Padrão Autenticação na linguagem C#.

```
using Biblioteca_Implementacoes.Utilitarios;
using ClosedXML.Excel;
using System;
using System.IO;

namespace Biblioteca_Implementacoes
{
    public static class RegistroDeOcorrencia
    {
        private static readonly string pastaRaiz = AppDomain.CurrentDomain.BaseDirectory;
        private static readonly string pastaCadastroUsuarios =
Properties.Resources.pastaCadastroUsuarios;
        private static readonly string arquivoUsuarios = @"\Usuarios.xlsx";
        private static readonly string tipoUsuarios = "Usuarios";
        private static readonly string arquivoLogin = @"\LoginsUsuarios.xlsx";
        private static readonly string tipoLogin = "LoginsUsuarios";

        internal static bool RegistroCadastroDeUsuarioExcel(string usuario, string senha, string
sal, string tipoDeUsuario)
        {
            if (!File.Exists(pastaRaiz + pastaCadastroUsuarios + arquivoUsuarios))
            {
                //cria o arquivo se não existir...
                using (var arquivo = new XLWorkbook())
                {
                    arquivo.AddWorksheet(tipoUsuarios);
                    arquivo.SaveAs(pastaRaiz + pastaCadastroUsuarios + arquivoUsuarios);
                    arquivo.Dispose();
                }
                //salva na primeira linha, já que não existe registro...
                using (var arquivo = new XLWorkbook(pastaRaiz + pastaCadastroUsuarios +
arquivoUsuarios))
                {
                    var pagina = arquivo.Worksheet(tipoUsuarios);
                    pagina.Cell(1, 1).Value = usuario;
                    pagina.Cell(1, 2).Value = senha;
                    pagina.Cell(1, 3).Value = sal;
                    pagina.Cell(1, 4).Value = tipoDeUsuario;
                    arquivo.Save();
                    arquivo.Dispose();
                }
                return true;
            }
            else
            {

```

```

        using (var arquivo = new XLWorkbook(pastaRaiz + pastaCadastroUsuarios +
arquivoUsuarios))
        {
            var pagina = arquivo.Worksheet(tipoUsuarios);
            var quantidadeDeEntradas = pagina.LastRowUsed().RowNumber();//retorna a
última linha utilizada

            for (int i = 1; i <= quantidadeDeEntradas; i++)
            {
                if (pagina.Cell(i, 1).Value.ToString().Equals(usuario))//se o usuário existir,
retorna falso
                {
                    return false;
                }
            }
            pagina.Cell(quantidadeDeEntradas + 1, 1).Value = usuario;//Cell(linha, coluna)
            pagina.Cell(quantidadeDeEntradas + 1, 2).Value = senha;
            pagina.Cell(quantidadeDeEntradas + 1, 3).Value = sal;
            pagina.Cell(quantidadeDeEntradas + 1, 4).Value = tipoDeUsuario;
            arquivo.Save();
            arquivo.Dispose();
        }
        return true;
    }
}

internal static bool RegistroLoginExcel(string usuario, ref string senha, string
timeStamp, string endIP, string endMAC)
{
    bool res = false;

    //verifica a existência do usuário
    using (var arquivo = new XLWorkbook(pastaRaiz + pastaCadastroUsuarios +
arquivoUsuarios))
    {
        var pagina = arquivo.Worksheet(tipoUsuarios);
        var quantidadeDeEntradas = pagina.LastRowUsed().RowNumber();

        //(x,1) ->usuário
        //(x, 2) -> senha
        //(x, 3) -> sal

        for (int i = 1; i <= quantidadeDeEntradas; i++)
        {
            if (pagina.Cell(i, 1).Value.ToString().Equals(usuario))
            {
                var sal = pagina.Cell(i, 3).Value.ToString(); //pega o sal armazenado
                var senhaHash = HashDeSenhas.CifrarSenha512(senha, sal); //gera um hash
com a senha dada pelo usuário
            }
        }
    }
}

```

```

        res = pagina.Cell(i, 2).Value.ToString().Equals(senhaHash); //confirma se são
iguais
        break;
    }
}
arquivo.Dispose();
}

if (!File.Exists(pastaRaiz + pastaCadastroUsuarios + arquivoLogin))
{
    //cria o arquivo se não existir...
    using (var arquivo = new XLWorkbook())
    {
        arquivo.AddWorksheet(tipoLogin);
        arquivo.SaveAs(pastaRaiz + pastaCadastroUsuarios + arquivoLogin);
        arquivo.Dispose();
    }
    //salva na primeira linha, já que não existe registro...
    using (var arquivo = new XLWorkbook(pastaRaiz + pastaCadastroUsuarios +
arquivoLogin))
    {
        var pagina = arquivo.Worksheet(tipoLogin);
        pagina.Cell(1, 1).Value = usuario;
        pagina.Cell(1, 2).Value = timeStamp;
        pagina.Cell(1, 3).Value = endIP;
        pagina.Cell(1, 4).Value = endMAC;
        pagina.Cell(1, 5).Value = res;
        arquivo.Save();
        arquivo.Dispose();
    }
    return res;
}
else
{
    using (var arquivo = new XLWorkbook(pastaRaiz + pastaCadastroUsuarios +
arquivoLogin))
    {
        var pagina = arquivo.Worksheet(tipoLogin);
        var quantidadeDeEntradas = pagina.LastRowUsed().RowNumber();

        pagina.Cell(quantidadeDeEntradas + 1, 1).Value = usuario; //Cell(linha, coluna)
        pagina.Cell(quantidadeDeEntradas + 1, 2).Value = timeStamp;
        pagina.Cell(quantidadeDeEntradas + 1, 3).Value = endIP;
        pagina.Cell(quantidadeDeEntradas + 1, 4).Value = endMAC;
        pagina.Cell(quantidadeDeEntradas + 1, 5).Value = res;
        arquivo.Save();
        arquivo.Dispose();
    }
    return res;
}

```



```

    }

    internal static string BuscaTipoUsuario(string usuario)
    {
        using (var arquivo = new XLWorkbook(pastaRaiz + pastaCadastroUsuarios +
arquivoUsuarios))
        {
            var pagina = arquivo.Worksheet(tipoUsuarios);
            var quantidadeDeEntradas = pagina.LastRowUsed().RowNumber();
            string res="";

            for (int i = 1; i <= quantidadeDeEntradas; i++)
            {
                if (pagina.Cell(i, 1).Value.ToString().Equals(usuario))
                {
                    res = pagina.Cell(i, 4).Value.ToString();
                    break;
                }
            }
            arquivo.Dispose();

            return res;
        }
    }
}

```

APÊNDICE D – Replicação de Dados

Neste apêndice D, encontra-se o Código Fonte do Padrão Replicação de Dados na linguagem C#.

```
using System;
using System.IO;

namespace Biblioteca_Implementacoes
{
    internal static class ReplicacaoDeDados
    {
        private static readonly string erroArquivoNaoExiste = "Arquivo não existe!";
        private static readonly string sucessoBackup = "Backup criado com sucesso!";
        private static readonly int codErroArquivoNaoExiste = 1;
        private static readonly int codSucesso = 0;
        private static readonly string arquivoUsuarios = "Usuários-.txt";

        internal static (int, string) BackupManual(string localDoArquivoSerCopiado, string destino)
        {
            string nomeDoArquivo = DateTime.Now.ToString("dd.MM.yyyy HH.mm.ss") + "-Backup-" + arquivoUsuarios;

            if (File.Exists(localDoArquivoSerCopiado))
            {
                if (!Directory.Exists(destino))
                {
                    Directory.CreateDirectory(destino);
                }

                File.Copy(localDoArquivoSerCopiado, destino + nomeDoArquivo);

                return (codSucesso, sucessoBackup);
            }
            else
            {
                return (codErroArquivoNaoExiste, erroArquivoNaoExiste);
            }
        }

        internal static (int, string) BackupManual(string localDoArquivoSerCopiado, string destino, string nomeDoArquivo)
        {
            string nomeDoArquivoBackup = @"\" + DateTime.Now.ToString("dd.MM.yyyy HH.mm.ss") + "-Backup-" + nomeDoArquivo;
```

```
File.Copy(localDoArquivoSerCopiado, destino + nomeDoArquivoBackup);  
return (codSucesso, sucessoBackup);  
}  
}  
}
```

APÊNDICE E – Proteção de Dados

Neste apêndice E, encontra-se o Código Fonte do Padrão Proteção de Dados na linguagem C#.

```
using System;
using System.Security.Cryptography;
using Biblioteca_Implementacoes.Utilitarios;
using System.IO;

namespace Biblioteca_Implementacoes
{
    internal static class ProtecaoDeDados
    {
        internal static string EncriptaTexto(ref string textoParaEncriptar, ref byte[] chaveSegura,
ref byte[] IV)
        {
            byte[] bytesEcriptados;

            using (var aes = Aes.Create())
            {
                aes.Key = chaveSegura;
                aes.IV = IV;
                aes.Padding = PaddingMode.PKCS7;
                aes.Mode = CipherMode.CBC;

                var encriptador = aes.CreateEncryptor(aes.Key, aes.IV);

                //Cria um stream dentro da memória
                using (var streamMemoria = new MemoryStream())
                {
                    //Cria um stream de criptografia, como alvo o stram de memória
                    //utilizando o encriptador em modo de escrita
                    using (var streamCripto = new CryptoStream(streamMemoria,
                        encriptador, CryptoStreamMode.Write))
                    {
                        //Cria um escritor para o streamcripto
                        using (var escritorStream = new StreamWriter(streamCripto))
                        {
                            escritorStream.Write(textoParaEncriptar);
                            escritorStream.Flush();
                            escritorStream.Dispose();
                        }

                        bytesEcriptados = streamMemoria.ToArray();
                        streamCripto.Dispose();
                    }

                    streamMemoria.Dispose();
                }
            }
        }
    }
}
```

```

        aes.Clear();
        aes.Dispose();
    }
    //limpa as strings
    textoParaEncriptar = string.Empty;
    Array.Clear(chaveSegura, 0, chaveSegura.Length);
    Array.Clear(IV, 0, IV.Length);

    return Convert.ToBase64String(bytesEcriptados);
}

internal static string DecriptaTexto(ref string textoParaDecriptar, ref byte[] chaveSegura,
ref byte[] IV)
{
    string textoDecriptado;
    byte[] textoEncriptado = Convert.FromBase64String(textoParaDecriptar);

    using (var aes = Aes.Create())
    {
        aes.Key = chaveSegura;
        aes.IV = IV;
        aes.Padding = PaddingMode.PKCS7;
        aes.Mode = CipherMode.CBC;

        var decriptador = aes.CreateDecryptor(aes.Key, aes.IV);

        using (var streamMemoria = new MemoryStream(textoEncriptado))
        {
            using (var streamDecripto = new CryptoStream(streamMemoria,
                decriptador, CryptoStreamMode.Read))
            {
                using (var leitorStream = new StreamReader(streamDecripto))
                {
                    textoDecriptado = leitorStream.ReadToEnd();
                    leitorStream.Dispose();
                }

                streamDecripto.Dispose();
            }

            streamMemoria.Dispose();
        }
        aes.Clear();
        aes.Dispose();
    }

    Array.Clear(chaveSegura, 0, chaveSegura.Length);
    Array.Clear(IV, 0, IV.Length);
}

```

```

        return textoDecriptado;
    }

    internal static void EncriptaArquivo(string pathArquivo, ref byte[] chaveSegura, ref
byte[] IV)
    {
        using (var aes = Aes.Create())
        {
            aes.Key = chaveSegura;
            aes.IV = IV;
            aes.Padding = PaddingMode.PKCS7;
            aes.Mode = CipherMode.CBC;

            var encriptador = aes.CreateEncryptor(aes.Key, aes.IV);

            using(var streamMemoria = new MemoryStream(File.ReadAllBytes(pathArquivo)))
            {
                using (var streamArquivo = File.Create(pathArquivo))
                {
                    using (var streamCripto = new CryptoStream(streamArquivo,
                                                                encriptador, CryptoStreamMode.Write))
                    {
                        streamCripto.Write(streamMemoria.ToArray(), 0,
streamMemoria.ToArray().Length);
                        streamCripto.Dispose();
                    }
                }
            }

            encriptador.Dispose();
            aes.Clear();
            aes.Dispose();
        }

        Array.Clear(chaveSegura, 0, chaveSegura.Length);
        Array.Clear(IV, 0, IV.Length);
    }

    internal static void DecriptarArquivo(ref string pathArquivo, ref byte[] chaveSegura, ref
byte[] IV)
    {
        using (var aes = Aes.Create())
        {
            aes.Key = chaveSegura;
            aes.IV = IV;
            aes.Padding = PaddingMode.PKCS7;
            aes.Mode = CipherMode.CBC;

            var decriptador = aes.CreateDecryptor(aes.Key, aes.IV);

```

```

        using (var streamMemoria = new
MemoryStream(File.ReadAllBytes(pathArquivo)))
        {
            using (var streamArquivo = File.Create(pathArquivo))
            {
                using (var streamCripto = new CryptoStream(streamArquivo,
                    decryptador, CryptoStreamMode.Write))
                {
                    streamCripto.Write(streamMemoria.ToArray(), 0,
streamMemoria.ToArray().Length);
                    streamCripto.Dispose();
                }
            }

            decryptador.Dispose();
            aes.Clear();
            aes.Dispose();
        }

        Array.Clear(chaveSegura, 0, chaveSegura.Length);
        Array.Clear(IV, 0, IV.Length);
    }
}

```

APÊNDICE F – Autorização

Neste apêndice F, encontra-se o Código Fonte do Padrão Autorização na linguagem C#.

```
using System;
using System.Collections.Generic;
using System.Text;
using ClosedXML.Excel;

namespace Biblioteca_Implementacoes
{
    internal static class Autorizacao
    {
        private static readonly string pastaRaiz = AppDomain.CurrentDomain.BaseDirectory;
        private static readonly string pastaCadastroUsuarios =
Properties.Resources.pastaCadastroUsuarios;
        private static readonly string arquivoUsuarios = @"\\Usuarios.xlsx";
        private static readonly string tipoUsuarios = "Usuarios";

        internal static string AutorizaUsuario(string usuario)
        {
            //busca o tipo de usuário
            return BuscaTipoUsuario(usuario);
        }

        internal static string BuscaTipoUsuario(string usuario)
        {
            using (var arquivo = new XLWorkbook(pastaRaiz + pastaCadastroUsuarios +
arquivoUsuarios))
            {
                var pagina = arquivo.Worksheet(tipoUsuarios);
                var quantidadeDeEntradas = pagina.LastRowUsed().RowNumber();
                string res = "";

                for (int i = 1; i <= quantidadeDeEntradas; i++)
                {
                    if (pagina.Cell(i, 1).Value.ToString().Equals(usuario))
                    {
                        res = pagina.Cell(i, 4).Value.ToString();
                        break;
                    }
                }
                arquivo.Dispose();

                return res;
            }
        }
    }
}
```


APÊNDICE G – Gerenciamento de Chaves Criptográficas

Neste apêndice G, encontra-se o Código Fonte do Padrão Gerenciamento de Chaves Criptográficas na linguagem C#.

```
using System.Security.Cryptography;
using Biblioteca_Implementacoes.Utilitarios;

namespace Biblioteca_Implementacoes
{
    internal static class GerenciamentoDeChaves
    {
        internal static (byte[], byte[]) gerarChave(string chaveDigitada, string IVDigitado)
        {
            var chave = HashDeSenhas.CifrarSenhaSHA256(chaveDigitada, IVDigitado);

            var IVTemp = HashDeSenhas.CifrarSenhaSHA256(IVDigitado);
            var IV = new byte[16];

            for (int i = 0; i < IVTemp.Length; i++)
            {
                IV[i] = IVTemp[i * 2];
            }

            return (chave, IV);
        }
    }
}
```

APÊNDICE H – *Timeout* de Sessão

Neste apêndice H, encontra-se o Código Fonte do Padrão *Timeout* de Sessão na linguagem C#.

```
using System;
using System.Threading;

namespace Biblioteca_Implementacoes
{
    public class TimeoutSessao
    {
        public bool ocorreuTimeout { get; private set; }
        private TimeSpan tempoRestante;
        private DateTime horarioLimite;
        public void ContagemRegressiva(int segundos)
        {
            this.ocorreuTimeout = false;

            this.horarioLimite = DateTime.Now.AddSeconds(Convert.ToDouble(segundos));

            Timer contador = new Timer(this.AtualizaTempoRestante, null, 0, 1000);
        }

        private void AtualizaTempoRestante(object objeto)
        {
            this.tempoRestante = horarioLimite - DateTime.Now;

            if (tempoRestante.TotalSeconds <= 0)
            {
                this.ocorreuTimeout = true;
            }
        }
    }
}
```

APÊNDICE I – Captar dados de Conexão

Neste apêndice I, encontra-se o Código Fonte da classe de captação de dados de conexão na linguagem C#.

```
using System;
using System.Net;
using System.Net.NetworkInformation;

namespace Biblioteca_Implementacoes.Utilitarios
{
    internal static class CaptaDadosConexao
    {
        internal static string CaptaIP()
        {
            var IPcliente = new
WebClient().DownloadString(Properties.Resources.siteCapturaIP).TrimEnd();
//site que mostra o ip público

            return IPcliente;
        }

        internal static string CaptaMAC()
        {
            string MAC_cliente="";

            var interfaceLocal = NetworkInterface.GetAllNetworkInterfaces(); //gera lista de
adptadores de rede

            foreach (var adaptador in interfaceLocal) //itera através doss adptadores de rede
            {
                MAC_cliente = adaptador.GetPhysicalAddress().ToString();
                break;
            }

            return MAC_cliente;
        }
    }
}
```

APÊNDICE J – Gerador de *Salt*

Neste apêndice J, encontra-se o Código Fonte da classe de Geração de *Salt* na linguagem C#.

```
using System;
using System.Security.Cryptography;
using System.Text;

namespace Biblioteca_Implementacoes.Utilitarios
{
    public static class GeradorSalt
    {
        //pega a lista de caracteres que formam o salt
        private static readonly char[] listaChars =
        Properties.Resources.listaDeCaracteres.ToCharArray();
        public static string GeraSalt()
        {
            int tamanho = Convert.ToInt32(Properties.Resources.tamanhoSalt);//140
            var saltLocal = new StringBuilder(tamanho); //string de caracteres
            byte[] dados = new byte[4 * tamanho];

            using (var cripto = new RNGCryptoServiceProvider()) //para garantir randomização
            {
                cripto.GetBytes(dados); //enche o vetor com números aleatórios
            }

            for (int i = 0; i < tamanho; i++)
            {
                var intAleatorio = BitConverter.ToUInt32(dados, i*4); //pega um valor(gerado
                //acima) da posicao do vetor
                var indice = intAleatorio % listaChars.Length;

                saltLocal.Append(listaChars[indice]);
            }

            return saltLocal.ToString(); //converte para string
        }
    }
}
```

APÊNDICE K – *Hash* de Senhas

Neste apêndice K, encontra-se o Código Fonte da classe de *Hash* de Senhas na linguagem C#.

```
using System;
using System.Security.Cryptography;
using System.Text;

namespace Biblioteca_Implementacoes.Utilitarios
{
    public static class HashDeSenhas
    {
        public static byte[] CifrarSenhaSHA256(string senhaInformada, string salt)
        {
            //Cifra a senha com SHA256 (implementação padrão do framework)
            //"salt" -> termo em inglês para um texto que é adicionado à senha
            string senhaComSalt; //senhaInformada + salt
            byte[] dados, hash;

            senhaComSalt = senhaInformada + salt; //adiciona o salt à senha

            dados = Encoding.UTF8.GetBytes(senhaComSalt);
            hash = SHA256.Create().ComputeHash(dados);

            return (hash);
        }

        public static byte[] CifrarSenhaSHA256(string senhaInformada)
        {
            //Cifra a senha com SHA256 (implementação padrão do framework)
            byte[] dados, hash;

            dados = Encoding.UTF8.GetBytes(senhaInformada);
            hash = SHA256.Create().ComputeHash(dados);

            return (hash);
        }

        public static string CifrarSenha512(string senhaInformada, string salt)
        {
            //Cifra a senha com SHA256 (implementação padrão do framework)
            //"salt" -> termo em inglês para um texto que é adicionado à senha
            string senhaCifradaLocal, //armazena o hash da senha, convertida para string
                senhaComSalt; //senhaInformada + salt
            byte[] dados, hash;

            senhaComSalt = senhaInformada + salt; //adiciona o salt à senha
            senhaInformada = string.Empty;
        }
    }
}
```

```
dados = Encoding.UTF8.GetBytes(senhaComSalt); //converte os caracteres para bytes  
hash = SHA512.Create().ComputeHash(dados);  
senhaCifradaLocal = Convert.ToBase64String(hash);  
return (senhaCifradaLocal);  
}  
}  
}
```