Daniel Albino Mosca Rodrigues

ESSEF: Um *framework* para Modelamento de Requisitos de Segurança em Sistemas Embarcados

Itajubá-MG, Brasil

2024

Daniel Albino Mosca Rodrigues

ESSEF: Um *framework* para Modelamento de Requisitos de Segurança em Sistemas Embarcados

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologia da Computação.

Universidade Federal de Itajubá – UNIFEI

Orientador: Prof. Dr. João Paulo Reus Rodrigues Leite

Itajubá-MG, Brasil

2024

Agradecimentos

Os agradecimentos são divididos para todos os professores da formação técnica, na Escola Técnica de Eletrônica "FMC", que me motivaram desde cedo a perseguir a carreira acadêmica com muito empenho. Também aos professores da graduação no Instituto Nacional de Telecomunicações que me levaram a almejar ainda mais a carreira de professor e pesquisador, e contribuíram imensamente com toda bagagem de engenharia que hoje utilizo para exercer a função com muito carinho. Agradeço também especialmente ao meu orientador e professor João Paulo R. R. Leite que potencializou a ideia inicial do trabalho e me deu todas as diretrizes para que este fosse realizado com excelência. Também agradeço a todos os outros professores da Universidade Federal de Itajubá que contribuíram imensamente para minha formação de mestre.

Resumo

Sistemas embarcados são cruciais em diversos domínios da computação, desde eletrônicos de consumo até automação industrial, mas também apresentam desafios significativos relacionados à segurança cibernética. Este trabalho conduz uma análise do estado atual da engenharia de requisitos de segurança para sistemas embarcados e explora soluções de cibersegurança adaptadas para eles. Preocupações específicas de segurança em ambientes embarcados são abordadas, levando à proposta de um *framework* para modelagem e documentação da arquitetura de segurança de sistemas embarcados. Este *framework* tem como objetivo facilitar a integração de requisitos de segurança em projetos envolvendo *firmware* e *hardware*. Fatores considerados incluem o cumprimento de requisitos de segurança, disponibilidade de recursos de *hardware*, sobrecarga de desempenho, compatibilidade e facilidade de integração. Partindo de uma avaliação sistemática, este artigo tem como objetivo fornecer orientações práticas para pesquisadores e profissionais na seleção de soluções de cibersegurança eficazes para sistemas embarcados e na modelagem deles de forma abrangente para melhorar a qualidade do *software* por meio da documentação.

Palavras-chaves: cibersegurança. engenharia de requisitos. avaliação de *software*. sistemas embarcados. *hashing*. segurança de *firmware*.

Abstract

Embedded systems are crucial in various computing domains, from consumer electronics to industrial automation, but they also pose significant cybersecurity challenges. This work conducts an analysis of the current state of security requirements engineering for embedded systems and explores cybersecurity solutions tailored for them. Specific security concerns in embedded environments are addressed, leading to the proposal of a framework for modeling and documenting the security architecture of embedded systems. This framework aims to facilitate the integration of security requirements in projects involving firmware and hardware. Factors considered include security requirements fulfillment, hardware resource availability, performance overhead, compatibility, and ease of integration. Through a systematic evaluation, this article aims to provide practical guidance for researchers and practitioners in selecting effective cybersecurity solutions for embedded systems and modeling them comprehensively for improved software quality through documentation.

Keywords: cybersecurity. requirements engineering. software evaluation. embedded systems. hashing. firmware security.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AES	*Advanced Encryption Standard*
ACL	*Access Control List*
ECSS	*European Cooperation for Space Standardization*
ESSEF	*Embedded Systems Security Evaluation Framework*
I/O	*Input/Output*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
I2C	*Inter-Integrated Circuit*
ISO	*International Organization for Standardization*
IoHT	*Internet of Health Things*
MD5	*Message Digest Algorithm*
NFR	*Non-functional Requirement*
OTP	*One-time Password*
RAM	*Random Access Memory*
RSA	*Rivest Shamir Adleman*
SIG	*Soft-Goal Interdependency Graph*
SHA	*Secure Hashing Algorithm*
SPI	*Serial Peripheral Interface*
STM	*ST Microcontroller*
UART	*Universal Asynchronous Receiver/Transmitter*
WDT	*Watchdog*
JTAG	*Joint Test Action Group*
CRC	*Cyclic Redundancy Check*

Sumário

1 Introdução

A fase de estabelecimento de requisitos de segurança em um projeto de computação se destaca como um dos aspectos mais importantes no ciclo de desenvolvimento de *software*; no entanto, apesar de sua importância, os desenvolvedores frequentemente negligenciam essa etapa (SALINI; KANMANI, 2016). Mesmo quando reconhecidos, os requisitos de segurança geralmente são articulados em níveis excessivamente abstratos, apresentando desafios na sua visualização efetiva dentro da documentação de arquitetura do projeto (PAJA; DALPIAZ; GIORGINI, 2015). Uma revisão cuidadosa da literatura destaca a existência de uma lacuna em métodos para aprimorar requisitos de segurança, especialmente no domínio de sistemas embarcados.

Nesse contexto específico, dos sistemas embarcados, temos que o processo está intrinsecamente ligado aos recursos de *hardware* disponíveis e à arquitetura subjacente do sistema. Sistemas embarcados, por sua natureza, impõem limitações na implementação de soluções de segurança devido aos seus recursos computacionais mais restritos (RAVI et al., 2004). Hoje, este tipo de sistema encontra amplo uso em diversas aplicações, que vão desde automação residencial até equipamentos médicos críticos com exigentes requisitos de disponibilidade e confiabilidade. No entanto, a integração desses sistemas em uma infraestrutura gera preocupações sobre potenciais vulnerabilidades de segurança, enfatizando a importância de identificar requisitos funcionais e implementar medidas de segurança em tais projetos.

A escolha das soluções de cibersegurança em sistemas embarcados é um desafio crítico devido a várias considerações. Primeiramente, a natureza diversificada das aplicações dos sistemas embarcados implica uma variedade de ameaças e vulnerabilidades potenciais, cada uma exigindo abordagens específicas de segurança (RAVI et al., 2004). Por exemplo, um sistema embarcado em um dispositivo médico pode enfrentar ameaças relacionadas à privacidade dos dados do paciente, enquanto um sistema em um dispositivo de controle industrial pode estar mais vulnerável a ataques de sabotagem física. Portanto, a seleção de soluções de segurança deve ser altamente adaptável e personalizável para atender às necessidades específicas de cada aplicação.

Além disso, os recursos limitados de *hardware* e as restrições de energia dos sistemas embarcados acrescentam complexidade à escolha das soluções de segurança. Enquanto em sistemas de computação convencionais é possível implementar soluções de segurança robustas sem muitas preocupações com recursos, em sistemas embarcados cada bit de memória e cada ciclo de *CPU* são preciosos (RAVI et al., 2004). Assim, as soluções de segurança devem ser eficientes em termos de uso de recursos, evitando sobrecarregar o

sistema e comprometer seu desempenho ou funcionalidade. Este dimensionamento também impacta no custo final do projeto e do produto, visto que microcontroladores podem necessitar de mais recursos de memória e processamento para comportar uma determinada solução de cibersegurança.

Outro desafio importante é a necessidade de garantir a interoperabilidade das soluções escolhidas com o restante do sistema embarcado e com outros sistemas com os quais ele pode interagir, como memórias externas, sensores, microprocessadores. Em um ecossistema cada vez mais interconectado, os sistemas embarcados frequentemente precisam se comunicar com outros dispositivos e sistemas, exigindo que as soluções de segurança sejam compatíveis e integráveis com uma ampla gama de tecnologias e protocolos de comunicação(ALOSEEL et al., 2020).

Diante desses desafios, a escolha das soluções de cibersegurança em sistemas embarcados requer uma abordagem cuidadosa e multifacetada, considerando não apenas as ameaças específicas e os recursos disponíveis, mas também a interoperabilidade e a adaptabilidade das soluções escolhidas.

Este trabalho introduz o *Framework* de Avaliação de Segurança de Sistemas Embarcados (ESSEF, do inglês *Embedded Systems Security Evaluation Framework*) como uma proposta para aprimorar o refinamento de requisitos de segurança funcionais e modelar a arquitetura de segurança de sistemas embarcados. O ESSEF facilita a visualização abrangente dos requisitos de segurança do sistema, suas soluções correspondentes e os impactos dos recursos computacionais dessas implementações. O processo de modelagem utiliza o Grafo de Interdependência de Objetivos *Soft* (SIG), desenvolvido originalmente por Chung et al. (2012) e adaptado para o escopo específico deste trabalho.

1.1 Motivação

A cibersegurança e a engenharia de requisitos para sistemas embarcados têm recebido menos atenção em comparação com outras áreas da computação. Isso é preocupante, considerando o papel cada vez mais crucial que os sistemas embarcados desempenham em diversos setores, desde dispositivos médicos até automóveis inteligentes. A falta de foco nessas áreas pode expor esses sistemas a vulnerabilidades e ameaças de segurança, comprometendo não apenas a integridade dos dados, mas também a segurança das pessoas que interagem com esses dispositivos.

Avaliando os *frameworks* existentes que auxiliam nas fases de engenharia de requisitos e arquitetura, observou-se que nenhum deles é especificamente adaptado para sistemas embarcados. Isso reflete uma lacuna significativa no campo da engenharia de *software*, na qual a maioria das metodologias e ferramentas se concentra em sistemas de grande escala, negligenciando as peculiaridades e restrições dos sistemas embarcados (STROBEL

et al., 2014). Para preencher essa lacuna, o ESSEF foi desenvolvido, contribuindo para os domínios de engenharia de requisitos e arquitetura de projetos para sistemas embarcados. O ESSEF não apenas oferece um conjunto de diretrizes e práticas recomendadas, mas também fornece ferramentas específicas para auxiliar no processo de desenvolvimento de sistemas embarcados seguros e confiáveis.

A determinação das soluções de segurança empregadas em um projeto influencia significativamente as dimensões dos componentes dentro de um sistema embarcado, abrangendo fatores como memória, poder de processamento, consumo de energia e velocidade operacional. Consequentemente, pode impactar aspectos críticos como custo, tempo de desenvolvimento e confiabilidade. Notavelmente, uma proporção considerável dos microcontroladores atualmente utilizados se enquadra na categoria de baixo orçamento, destacando a necessidade de métodos de implementação apropriados no código. Abordando essa consideração, a modelagem de arquitetura proposta neste trabalho visa destacar as contribuições e os efeitos de cada solução de segurança nos recursos computacionais disponíveis no sistema embarcado. Ao fazer isso, os desenvolvedores podem tomar decisões informadas sobre a seleção e implementação de medidas de segurança, garantindo que os sistemas embarcados atendam aos requisitos de segurança e funcionem de maneira eficiente dentro das restrições de recursos disponíveis.

No entanto, além de garantir a segurança e o funcionamento eficiente, a documentação de *software* desempenha um papel fundamental na manutenção e longevidade do código em sistemas embarcados. Em um ambiente onde os recursos são limitados e as restrições são altas, uma documentação clara e abrangente pode facilitar a compreensão do código-fonte, permitindo atualizações e modificações sem comprometer a integridade do sistema.

A documentação adequada não apenas fornece uma referência para desenvolvedores atuais e futuros, mas também ajuda a mitigar o risco de erros e falhas durante o ciclo de vida do *software*. Em sistemas embarcados, onde atualizações de *software* podem ser desafiadoras devido a restrições de recursos e restrições de acesso físico, uma documentação detalhada pode economizar tempo e esforço, garantindo que as mudanças sejam implementadas de forma correta e eficiente.

Além disso, a documentação clara e abrangente pode ser crucial para conformidade regulatória e auditorias de segurança em setores altamente regulamentados, como a indústria médica e automotiva. Ao fornecer uma trilha de auditoria transparente e completa, a documentação de *software* ajuda a garantir a conformidade com padrões e regulamentos específicos, protegendo a reputação da empresa e evitando possíveis penalidades legais.

Portanto, a sistematização da escolha de soluções de cibersegurança em sistemas embarcados e a implementação de uma documentação de *software* robusta são essenciais para garantir não apenas a segurança e o funcionamento eficiente desses sistemas, mas

também sua manutenção e longevidade ao longo do tempo.

1.2 Estrutura

Este trabalho está organizado da seguinte forma: Na Seção 2, é fornecido um entendimento dos fundamentos subjacentes aos tópicos de cibersegurança, engenharia de requisitos e sistemas embarcados. Em seguida, a pergunta de pesquisa é apresentada (Seção 3), destacando a pergunta principal, que é o foco da fase inicial deste estudo, e as perguntas secundárias, que apoiaram a construção do *framework* ESSEF. Por meio de uma revisão sistemática da literatura, foram coletados artigos científicos relevantes relacionados aos pontos previamente definidos. No final do artigo, o *framework* é apresentado (Seção 4), delineando os passos desde a coleta de requisitos, considerando os requisitos fundamentais de segurança, até o refinamento deles em categorias de soluções seguras e, em seguida, modelando completamente cada solução presente em um banco de dados pré-selecionado concernente aos recursos computacionais impactados: memória, velocidade e consumo de energia.

2 Contexto

2.1 Árvore de Requisitos

Uma árvore de requisitos, também conhecida como árvore de requisitos de engenharia, é uma representação gráfica hierárquica que descreve os requisitos de um sistema de maneira estruturada (CHUNG et al., 2012). Nessa representação (Figura 1), os requisitos são organizados em níveis hierárquicos, sendo que cada nível representa um aspecto específico do sistema e os requisitos são refinados à medida que descemos na hierarquia. Essa técnica é comumente usada na engenharia de *software* para organizar e visualizar os requisitos do sistema de forma clara e compreensível.

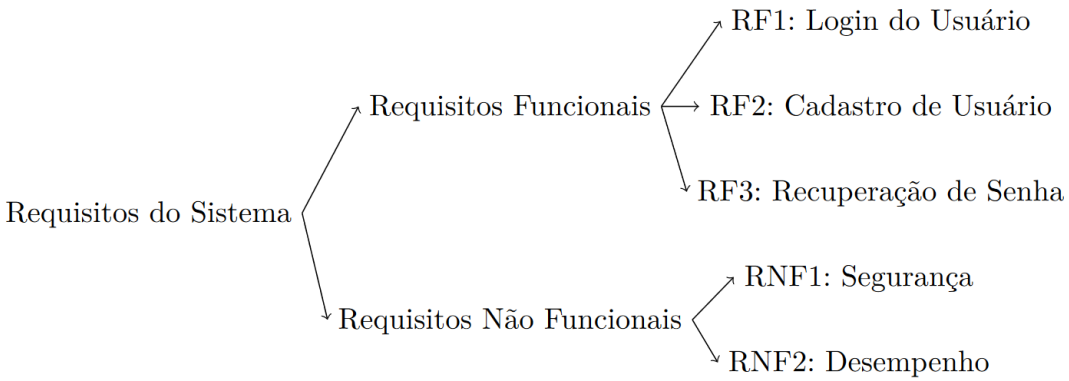

Figura 1 – Exemplo de Árvore de Requisitos

No contexto de sistemas embarcados, que são sistemas computacionais integrados em dispositivos ou produtos físicos, a árvore de requisitos desempenha um papel fundamental na especificação das necessidades e funcionalidades do sistema de forma detalhada e sistemática. Ela permite que os engenheiros de *software* e projetistas de sistemas compreendam os requisitos do sistema em um nível mais granular, identificando as relações entre os requisitos e garantindo que todas as necessidades do sistema sejam adequadamente capturadas. A estrutura hierárquica da árvore de requisitos permite uma abordagem sistemática para o desenvolvimento de sistemas embarcados, na qual os requisitos são decompostos em requisitos mais detalhados e específicos à medida que avançamos na hierarquia. Isso facilita o gerenciamento e a rastreabilidade dos requisitos ao longo do ciclo de vida do projeto, desde a análise e especificação até o design, implementação e testes.

Além disso, a árvore de requisitos também pode ser usada como uma ferramenta de comunicação eficaz entre os diferentes *stakeholders* do projeto, incluindo clientes, gerentes

de projeto, desenvolvedores e testadores. Ao fornecer uma visualização clara dos requisitos do sistema e suas inter-relações, a árvore de requisitos ajuda a garantir que todas as partes interessadas tenham uma compreensão comum dos objetivos e funcionalidades do sistema, facilitando assim o processo de desenvolvimento e garantindo a entrega de um produto de alta qualidade que atenda às expectativas do cliente.

Em resumo, a árvore de requisitos é uma ferramenta poderosa e versátil na engenharia de sistemas embarcados, fornecendo uma maneira estruturada e organizada de especificar e gerenciar os requisitos do sistema, garantindo assim o sucesso do projeto e a satisfação do cliente.

2.1.1 Passos para Aplicação em Sistemas Embarcados

1. **Identificação de Requisitos Gerais:** Identificar os requisitos gerais do sistema embarcado. Isso pode incluir funcionalidades principais, restrições de desempenho, interfaces com outros sistemas, entre outros.

2. **Decomposição Hierárquica:** Organizar os requisitos identificados em uma estrutura hierárquica. Os requisitos mais gerais e abrangentes são colocados no topo da árvore, enquanto os requisitos mais específicos e detalhados são colocados nas camadas inferiores.

3. **Detalhamento dos Requisitos:** Para cada nó na árvore, detalhar os requisitos. Isso envolve definir claramente condições, comportamentos e características desejadas.

4. **Relacionamentos entre Requisitos:** Estabelecer relacionamentos entre diferentes requisitos para mostrar dependências e interações. Isso ajuda a entender como as funcionalidades do sistema estão interconectadas.

5. **Validação e Verificação:** Garantir que os requisitos sejam claros, consistentes e verificáveis. Isso é crucial para garantir que o sistema atenda às expectativas e necessidades do usuário.

6. **Adaptação às Restrições do Ambiente Embarcado:** Considerar as restrições específicas dos sistemas embarcados, como limitações de recursos (energia, memória, processamento), tolerância a falhas, requisitos em tempo real e outros. A árvore de requisitos deve refletir essas considerações.

7. **Iteração e Atualização:** A árvore de requisitos é um artefato dinâmico que pode ser atualizado ao longo do ciclo de vida do projeto. À medida que novas informações são obtidas ou os requisitos são alterados, a árvore deve ser revisada e atualizada iterativamente.

2.2 Requisitos Funcionais

Os requisitos funcionais fornecem descrições específicas das funções que um sistema ou *software* deve executar para atender às necessidades do usuário e aos objetivos do projeto (MALAN; BREDEMEYER et al., 2001). Eles se concentram no 'o que' o sistema deve fazer, excluindo o 'como'. Aspectos-chave dos requisitos funcionais incluem:

1. **Descrição de Funcionalidades**: Os requisitos descrevem as ações, processos e informações que o sistema deve manipular.
2. **Especificidade**: Eles devem ser formulados de forma clara e não ambígua.
3. **Priorização**: É essencial definir prioridades para alocar os recursos de desenvolvimento de forma eficaz.
4. **Rastreabilidade**: Cada funcionalidade deve estar vinculada a uma necessidade do usuário ou objetivo de negócio.
5. **Testabilidade**: Critérios de aceitação devem ser definidos para verificar a implementação correta.
6. **Mudanças e Evolução**: Os requisitos podem mudar ao longo do projeto; é necessário documentar e gerenciar essas mudanças.
7. **Documentação**: Os requisitos são formalmente documentados, facilitando o acesso e a compreensão pela equipe.
8. **Iteração**: Conforme o desenvolvimento avança, os requisitos podem ser refinados com base em novas informações e necessidades.

Em resumo, os requisitos funcionais são essenciais para o desenvolvimento de *software* e sistemas, delineando o que o sistema deve fazer. Eles devem ser elaborados, priorizados, documentados e validados ao longo do projeto para garantir o sucesso.

2.2.1 Exemplos de Requisitos Funcionais na Área de Software:

Para melhor detalhar o uso do conceito de requisitos no projeto de um *software*, temos alguns exemplos em caráter geral e um exemplo na área de cibersegurança:

1. **Sistema de Gerenciamento de Biblioteca:**
    - RF1: O sistema deve permitir que os usuários façam login utilizando suas credenciais de acesso.

- RF2: O sistema deve permitir que os usuários pesquisem e solicitem livros disponíveis no catálogo.
- RF3: O sistema deve enviar notificações por e-mail aos usuários quando um livro reservado estiver disponível para retirada.

2. **Aplicativo de Compras Online:**

- RF1: O sistema deve permitir que os usuários adicionem itens ao carrinho de compras.
- RF2: O sistema deve calcular automaticamente o total da compra, incluindo impostos e custos de envio.
- RF3: O sistema deve fornecer opções de pagamento seguro, como cartão de crédito ou PayPal.

Exemplo de Requisito Funcional na Cibersegurança:

1. **Sistema de Gerenciamento de Acesso:**

- RF1: O sistema deve autenticar os usuários por meio de credenciais seguras, como senha única e autenticação de dois fatores.
- RF2: O sistema deve registrar e monitorar as atividades dos usuários, incluindo tentativas de login e acesso a recursos sensíveis.
- RF3: O sistema deve permitir que os administradores concedam e revoguem privilégios de acesso com base nas funções e responsabilidades dos usuários.

2.3 Requisitos Não Funcionais

Os requisitos não funcionais abrangem critérios que delineiam as características de desempenho, segurança, usabilidade e qualidade que um sistema ou *software* deve obedecer. Ao contrário dos requisitos funcionais, que definem 'o que' o sistema faz, os requisitos não funcionais se concentram em 'como' ele alcança essas funções (CHUNG et al., 2012). Aspectos-chave dos requisitos não funcionais incluem:

1. **Desempenho**: Descreve a capacidade do sistema de responder às demandas, incluindo tempo de resposta, taxa média de transferência e escalabilidade.

2. **Segurança**: Refere-se a medidas de proteção de dados, autenticação de usuários e prevenção de ameaças.

3. **Usabilidade**: Define a facilidade de uso e a experiência do usuário, incluindo interface do usuário e acessibilidade.

4. **Confiabilidade**: Descreve a disponibilidade do sistema e sua capacidade de lidar com falhas.

5. **Manutenibilidade**: Indica o quão facilmente o sistema pode ser mantido e atualizado.

6. **Compatibilidade**: Refere-se à capacidade do sistema de funcionar em diferentes plataformas e ambientes.

7. **Eficiência**: Diz respeito à otimização de recursos, como uso de processamento e memória.

8. **Legal e Regulatório**: Inclui requisitos relacionados à conformidade com leis e regulamentos específicos.

9. **Interoperabilidade**: Descreve a capacidade do sistema de interagir com outros sistemas e serviços.

10. **Documentação**: Os requisitos não funcionais podem incluir a necessidade de documentação adequada.

11. **Custo**: Define restrições orçamentárias e requisitos de custo.

Em resumo, os requisitos não funcionais são tão importantes quanto os funcionais, pois determinam como o sistema executa suas funções. Eles devem ser cuidadosamente identificados, documentados e considerados ao longo do ciclo de vida do projeto para garantir que o sistema atenda aos padrões desejados de qualidade e desempenho.

Exemplos de Requisitos Não Funcionais na Área de Software:

1. **Sistema de Gerenciamento de Biblioteca:**

   - RNF1: O sistema deve garantir uma resposta às solicitações de pesquisa de livros em menos de 2 segundos.
   - RNF2: O sistema deve ser compatível com diferentes navegadores da web, incluindo Chrome, Firefox e Safari.
   - RNF3: O sistema deve ter uma taxa de disponibilidade de pelo menos 99% durante o horário comercial.

2. **Aplicativo de Compras Online:**

- RNF1: O sistema deve ser capaz de lidar com um pico de carga de 1000 usuários simultâneos durante períodos de venda.
- RNF2: O sistema deve estar em conformidade com os padrões de segurança PCI-DSS para proteção de dados do cartão de crédito.
- RNF3: O sistema deve ser internacionalizado e suportar múltiplos idiomas, incluindo inglês, espanhol e francês.

Exemplo de Requisito Não Funcional na Cibersegurança:

1. **Sistema de Gerenciamento de Acesso:**

   - RNF1: O sistema deve garantir que todas as transações de login e acesso sejam registradas em um arquivo de log protegido contra modificação.
   - RNF2: O sistema deve ser capaz de detectar e bloquear tentativas de acesso não autorizadas em menos de 5 segundos.
   - RNF3: O sistema deve ser projetado para minimizar o consumo de recursos de hardware, garantindo um desempenho otimizado.

2.4 Objetivos *Soft*

Os objetivos *soft* referem-se a dimensões subjetivas e não técnicas de um sistema, frequentemente associadas à experiência do usuário, preferências estéticas e comportamento. Divergindo dos requisitos técnicos mais objetivos, os requisitos *soft* podem ser desafiadores de serem quantificados e medidos, mas têm uma influência significativa na melhoria da satisfação do usuário e na aceitação do sistema (SEN; JAIN, 2007). Embora os objetivos *soft* possam ser categorizadas dentro dos requisitos não funcionais, eles se destacam devido à sua natureza subjetiva, enfatizando a experiência do usuário e as preferências. Este estudo explora suas derivações, que são:

2.4.0.1 Objetivos *Soft* de Reivindicação

Esse tipo de objetivo representa declarações qualitativas sobre o desempenho desejado nos objetivos *soft*. Eles ajudam a comunicar claramente e de forma compreensível as expectativas em relação a esses requisitos.

2.4.0.2 Objetivos *Soft* Operacionalizáveis

A operacionalização dos objetivos envolve a transformação de declarações qualitativas em requisitos mensuráveis e específicos. Isso significa traduzir conceitos subjetivos em

critérios objetivos que podem ser testados e avaliados. Para a declaração anterior, a operacionalização pode incluir métricas de usabilidade específicas, como tempo de aprendizado do usuário ou taxa de conclusão de tarefas. A operacionalização torna possível avaliar tangivelmente a realização dos objetivos *soft*.

2.4.1 Exemplos de Objetivos Soft

2.4.1.1 Objetivos Soft

- Sensação de Confiança do Usuário;
- Facilidade de Uso da Interface de Segurança;
- Estética Visual do Sistema.

2.4.1.2 Objetivos Soft de Reivindicação

- Melhorar a Percepção de Segurança do Usuário;
- Aprimorar a Intuitividade da Interface de Segurança;
- Reforçar a Impressão Profissional da Estética Visual.

2.4.1.3 Objetivos Soft Operacionalizáveis

- Implementar mecanismos que adicionam confiabilidade nos componentes chave do sistema, evitando principalmente travamentos e demora nos tempos de reposta em eventos;
- Reduzir o tempo de treinamento necessário do usuário por meio de uma interface mais intuitiva;
- Implementar um design moderno e profissional, utilizando *frameworks* para melhor coesão entre os componentes gráficos.

2.5 Sistemas Embarcados

Sistemas embarcados são sistemas computacionais especializados projetados para tarefas específicas dentro de dispositivos eletrônicos e equipamentos, e são fundamentais para vários domínios, desde eletrodomésticos inteligentes até máquinas industriais (DUTT; JANTSCH; SARMA, 2016). Esses sistemas, intimamente integrados ao *hardware*, geralmente operam em tempo real e contribuem significativamente para a eficiência e conveniência na vida diária.

Um sistema embarcado geralmente é composto por uma unidade microcontrolada com *software* embarcado, frequentemente referido como *firmware*, e requer um circuito de referência para funcionar de acordo com as especificações do projeto. Este circuito pode incluir circuitos de memória, cristais de oscilação e componentes passivos para filtros de sinal e fontes de alimentação.

A implementação de segurança em sistemas embarcados apresenta desafios únicos devido a restrições de recursos como memória limitada, capacidade de processamento e requisitos de baixo consumo de energia (ALOSEEL et al., 2020). Interfaces de comunicação de diferentes protocolos são comuns, necessitando de medidas como algoritmos de criptografia para proteger o tráfego de dados (LÁZARO et al., 2011).

Além disso, a evolução dos sistemas embarcados trouxe consigo a necessidade de lidar com desafios de segurança cada vez mais complexos. Com a proliferação de dispositivos conectados à *Internet*, como *IoT* (Internet das Coisas, ou *Internet of Things*) e dispositivos médicos implantáveis, a segurança tornou-se uma preocupação crítica. Os sistemas embarcados agora enfrentam ameaças como ataques de negação de serviço (*DoS*, ou *Denial of Service*), invasões de rede e roubo de dados confidenciais. Portanto, a implementação de mecanismos de segurança robustos tornou-se uma prioridade para os desenvolvedores de sistemas embarcados.

Para enfrentar esses desafios, os engenheiros de sistemas embarcados estão adotando abordagens inovadoras, como a integração de técnicas de criptografia avançada diretamente nos dispositivos, o uso de *firewalls* de rede embutidos e a implementação de protocolos de autenticação rigorosos. Além disso, a conscientização sobre segurança em todas as fases do ciclo de vida do desenvolvimento de sistemas embarcados, desde a concepção até a implantação e manutenção, tornou-se essencial para garantir a integridade e a confiabilidade dos sistemas.

Portanto, enquanto os sistemas embarcados continuam a desempenhar um papel crucial em nossa vida cotidiana, a garantia da segurança desses sistemas é fundamental para proteger a privacidade, a segurança e o bem-estar dos usuários finais.

2.5.1 Microcontroladores

Microcontroladores, unidades de processamento com memória programável limitada (*FLASH*) e memória de acesso dinâmico (*RAM*), juntamente com periféricos como conversores analógico-digital e temporizadores, são componentes fundamentais de sistemas embarcados. Essas unidades são projetadas para tarefas específicas e encontram amplo uso em diversas aplicações. Um microcontrolador típico inclui um processador responsável por executar instruções, memória para armazenar dados e instruções, e periféricos para utilidades específicas, como entradas/saídas (I/O) e interfaces de comunicação.

A programação para microcontroladores, frequentemente realizada em linguagens como *C* ou *Assembly*, requer adesão a práticas de otimização para gerenciar efetivamente a memória do sistema (DUTT; JANTSCH; SARMA, 2016). Periféricos, configuráveis através de registradores, auxiliam na implementação de funcionalidades específicas, com temporizadores *watchdog* garantindo a disponibilidade do sistema.

2.6 Requisitos de Segurança

A segurança forma um aspecto muito importante na definição de sistemas computacionais confiáveis, sendo a seleção de soluções de segurança adequadas essencial durante a elicitação de requisitos funcionais (SALINI; KANMANI, 2016). Apesar dos desafios em auxiliar os desenvolvedores com a engenharia de requisitos de segurança, estudos ajudam a identificar e refinar os requisitos de segurança de forma eficaz (MERIDJI et al., 2019).

Os quatro pilares da segurança - disponibilidade, integridade, autenticidade e confidencialidade - são requisitos recorrentes em sistemas seguros (EL-HADARY; EL-KASSAS, 2014). Embora as fases iniciais de desenvolvimento possam carecer de requisitos de segurança específicos, abordar esses pilares garante uma arquitetura de software fundamentalmente segura. Além disso, é importante destacar que a segurança não é um requisito estático, mas sim um processo contínuo que evolui ao longo do ciclo de vida do desenvolvimento do sistema. À medida que novas ameaças emergem e as tecnologias evoluem, os requisitos de segurança também devem ser atualizados e adaptados para garantir a proteção contínua do sistema contra potenciais vulnerabilidades.

A abordagem para a elicitação e refinamento de requisitos de segurança pode variar de acordo com o contexto do sistema e os requisitos específicos do projeto. No entanto, algumas práticas comuns incluem a realização de análises de risco, a identificação de atores e cenários de ameaças, e a definição de controles de segurança adequados para mitigar os riscos identificados.

Além disso, a colaboração entre diferentes partes interessadas, incluindo desenvolvedores, especialistas em segurança e usuários finais, desempenha um papel fundamental no processo de engenharia de requisitos de segurança. Ao envolver todas as partes interessadas desde o início do projeto, é possível garantir que os requisitos de segurança sejam compreendidos e atendidos de maneira eficaz ao longo de todo o ciclo de vida do desenvolvimento do sistema.

2.7 Arquitetura de *Software* Segura

Esforços colaborativos na pesquisa e seleção de soluções de segurança são fundamentais para definir uma pilha de software que esteja alinhada com as necessidades do

projeto e parâmetros computacionais (RAVI et al., 2004). Medir os impactos dos requisitos de segurança durante as etapas de desenvolvimento de *software* é essencial, não apenas para avaliar a segurança do *software*, mas também para avaliar o quanto ele atende a cada requisito de segurança (LÁZARO et al., 2011).

3 Revisão Sistemática da Literatura

Este estudo emprega o método de revisão sistemática da literatura e revisão bibliográfica, envolvendo uma busca processual de trabalhos relacionados com base em critérios de inclusão e exclusão, juntamente com métricas para avaliar os resultados da pesquisa. O objetivo desta etapa é analisar a gama de estudos conduzidos na área de engenharia de requisitos de segurança, separando-os por área de concentração, e assim obter a percepção do progresso desses estudos no campo de sistemas embarcados. Para isso, foram selecionadas pesquisas que lidam com modelagem de requisitos e como podemos definir melhor soluções seguras para sistemas-alvo seguindo critérios de decisão bem selecionados e coerentes com conceitos e práticas de cibersegurança em todo o mundo (MERIDJI et al., 2019).

Na realização de uma revisão sistemática da literatura, é essencial formular uma questão principal e, às vezes, questões secundárias para direcionar o escopo da pesquisa e a análise dos resultados. A questão principal é a pergunta central que orienta toda a revisão. Ela define o objetivo principal da pesquisa e o foco da análise. Geralmente, a questão principal é ampla e abrangente, buscando uma compreensão holística do tema de estudo. As questões secundárias são perguntas mais específicas que ajudam a desdobrar a questão principal em áreas temáticas mais detalhadas. Elas são formuladas para investigar aspectos específicos do tema de pesquisa, fornecendo uma estrutura mais detalhada para a análise dos estudos selecionados.

3.1 Questão Principal

- **QP** - Qual é o processo para avaliar os requisitos de segurança no ciclo de desenvolvimento de *software*?

3.2 Questões Secundárias

- **QS1** - Qual é a direção dos estudos de segurança para sistemas embarcados?
- **QS2** - Quais requisitos funcionais são fundamentais para iniciar um design de segurança no projeto?
- **QS3** - Como selecionar requisitos de segurança com base na disponibilidade de recursos em um sistema embarcado?
- **QS4** - Como definir soluções seguras com base em requisitos não funcionais?

- **QS5** - Quais limitações do ambiente computacional devem ser consideradas na seleção de uma solução segura?

3.3 Foco e Escopo

A pesquisa se concentra na literatura que discute desafios e técnicas para identificar e traduzir requisitos de segurança em soluções prontas para o mercado e adequadas para implementação. Com base em estudos existentes, a pesquisa procede a analisar processos para refinar requisitos e propõe um *framework* para aplicar esses processos e modelagem dentro do domínio do *software* embarcado.

3.4 Métodos de Busca

Fontes digitais foram acessadas na *web* usando expressões de busca e palavras-chave predefinidas. Estudos foram escolhidos com base em referências de organizações de computação ou segurança, ou se fossem projetos exemplo de um provedor de soluções criptográficas para qualquer um dos ambientes computacionais analisados no estudo. O processo de seleção envolve dois passos:

1. Uma seleção inicial e catalogação dos dados coletados, que envolvem a aplicação da expressão de busca às fontes de dados escolhidas. Cada publicação é então catalogada em um banco de dados dedicado e armazenada em um repositório para análise subsequente.

2. Seleção de dados relevantes, pois a seleção preliminar não garante que todo o material coletado seja útil no contexto da pesquisa, já que a aplicação de expressões de busca é restrita ao aspecto sintático. Assim, após identificar as publicações por meio de mecanismos de busca, os resumos foram lidos e analisados seguindo os critérios de inclusão e exclusão.

3.4.1 Critérios de Exclusão

- **CE1-01** - Publicações em que as palavras-chave não estão presentes, e não há variações dessas palavras-chave (exceto plural), não serão selecionadas.

- **CE1-02** - Publicações em que as palavras-chave de busca não aparecem no título, resumo e/ou texto da publicação (excluindo o campo 'palavras-chave', seção de agradecimentos, biografias dos autores, referências bibliográficas e anexos) não serão selecionadas.

- **CE1-03** - Publicações que descrevem e/ou apresentam "palestras principais", tutoriais, cursos, *workshops* e similares não serão selecionadas.

3.4.2 Critérios de Inclusão

Apenas publicações do conjunto preliminar que atendam aos seguintes critérios podem ser incluídas:

- **CI1-01** - Publicações que mencionem a coleta de requisitos de segurança podem ser selecionadas.
- **CI1-02** - Publicações que analisem soluções de segurança específicas podem ser selecionadas.
- **CI1-03** - Publicações que discutam a aceitabilidade de requisitos de segurança em áreas de computação podem ser selecionadas.
- **CI1-04** - Publicações que descrevam evoluções (extensões) no campo da coleta de requisitos em Engenharia de Software podem ser selecionadas.

3.4.3 Critério de Seleção

Após limitar as publicações pelos critérios de inclusão e exclusão, outro critério de seleção mais específico deve ser aplicado para garantir que os estudos sejam consistentes com a proposta de avaliar requisitos de segurança como requisitos não funcionais:

CS2 -SecReq -NFR - Publicações que não consideram os conceitos não funcionais das soluções de segurança não devem ser selecionadas. Todas as referências encontradas na primeira rodada de busca serão submetidas ao 1º filtro e à aplicação dos critérios de inclusão e exclusão. Se passarem por isso e pelo critério de seleção definido como o 2º filtro, o artigo será catalogado em um banco de dados para a extração de dados de pesquisa ocorrer.

3.5 Extração de Dados

Para a primeira etapa da pesquisa, o tópico de interesse foi definido: requisitos de segurança na engenharia de *software*. O objetivo desta etapa é avaliar a relevância dos requisitos de segurança em projetos de *software* e entender as dificuldades atuais enfrentadas por equipes de desenvolvimento ou pesquisadores. Considerando os critérios de inclusão e exclusão, estudos que não consideram requisitos de segurança na fase não funcional, ou seja, quando o objetivo do artigo é avaliar soluções de segurança específicas, não foram avaliados no primeiro filtro.

Tabela 1 – Máquinas de busca acessadas

Máquina	**Link**	**Acessado em**
SCOPUS	<http://www.scopus.com/home.url>	OUT/2023
CAPES	<https://www-periodicos-capes-gov-br>	OUT/2023
IEEE	<http://www.ieee.org/portal/site>	OUT/2023

Com o tema de pesquisa definido, as palavras-chave e expressão de busca foram utilizadas em mecanismos de busca listados no Quadro 1. As palavras-chave selecionadas para a execução das buscas foram, em inglês, "*security requirements*" e seus sinônimos. A expressão de busca inicial foi definida como *("software security requirements" OU "security design") E ("security algorithms" OU "criptography")*. A escolha visa limitar os estudos que analisam a fase de coleta de requisitos de um projeto de software, não as medições realizadas após a implementação de soluções seguras.

Após a primeira rodada de buscas, foram obtidos resultados de estudos sobre requisitos de segurança em várias áreas da computação. Como não é um critério de exclusão que esses requisitos sejam discutidos em áreas e seções específicas do conhecimento, os estudos foram considerados independentemente da área de aplicação.

Os resultados foram catalogados. Nenhuma meta-análise foi realizada nesta fase, mas as soluções são analisadas e discutidas nos tópicos subsequentes. Os dados extraídos das publicações selecionadas foram armazenados em um banco de dados que inclui: título do artigo, autor(es), data e local de publicação.

Os objetivos desta fase da pesquisa são: i) definir o escopo do tema de interesse e ii) conduzir estudos preliminares sobre o assunto. Portanto, esses parâmetros atendem aos objetivos iniciais.

3.5.0.1 Primeira Rodada

A seguinte expressão de busca foi utilizada durante a primeira consulta aos periódicos: *("software security requirements" OU "security design") E ("secure algorithms" OU "criptography")*. Para compreender e justificar a concentração de estudos sobre segurança de software ao longo das décadas, a primeira rodada foi conduzida sem um corte temporal. No total, 844 artigos foram obtidos. O estudo mais antigo foi publicado em 1976, um artigo publicado em 1986 e 13 artigos publicados na década de 1990 foram obtidos. A Figura 2 mostra a distribuição dos trabalhos ao longo das décadas.

3.5.1 Segunda Rodada

Para a segunda rodada, a mesma sequência de busca foi utilizada, mas com o corte temporal previamente definido, a partir de 1º de junho de 2012. Esse corte foi considerado para abranger o período de aproximadamente 10 anos a partir da data do início desta

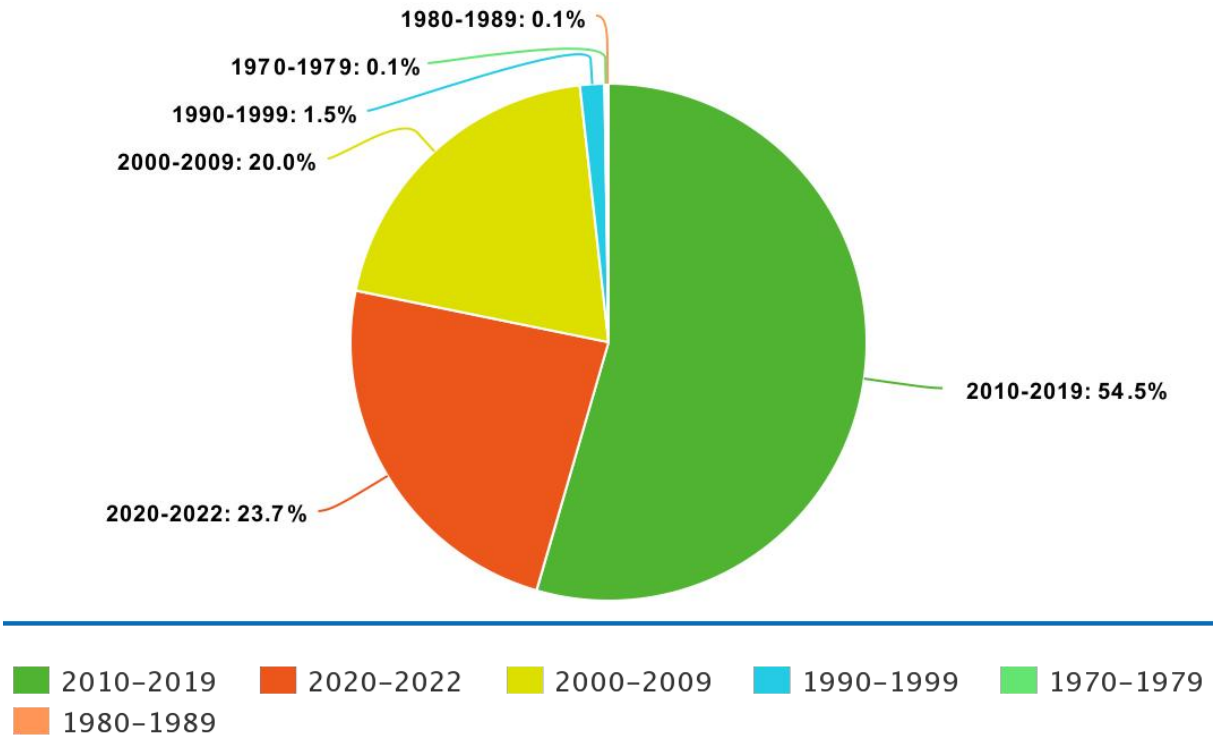

Figura 2 – Distribuição temporal dos artigos

pesquisa, presumindo que trabalhos dentro desse período sejam mais relevantes para o tópico estudado.

Foram obtidos 596 artigos, ou seja, 70,6% dos resultados da primeira sequência de busca estão contidos na última década. Destes, os 24 artigos científicos mais relacionados ao tópico de "requisitos de segurança não funcionais" (4% do total) foram selecionados manualmente com base na leitura em seus tópicos de resumo e com base nos critérios de inclusão listados em 3.4.2. Nesta seleção foram considerados os estudos que discutem a coleta de requisitos de segurança em projetos de *software*. Após aplicar os filtros e critérios de exclusão, esse número foi reduzido para 9 artigos. Estes foram listados no Quadro 2.

3.5.2 Terceira Rodada

Para a terceira rodada, a expressão de busca foi modificada. A intenção foi restringir os resultados a estudos que não apenas analisam requisitos de *software* não funcionais, mas também propõem ferramentas e métodos para convertê-los em requisitos funcionais ou medir a relevância dos requisitos não funcionais, conforme os critérios de inclusão em 3.4.2. Para alcançar esse objetivo, os termos *"requisitos não funcionais"* ou "NFR" foram adicionados com o intuito de captar estudos que incluem a análise desse tipo de requisito.

Além disso, para capturar estudos que incorporem pelo menos um dos requisitos não funcionais delineados nesta pesquisa, os termos "*criptografia de dados*", "*autenticidade*",

Tabela 2 – Artigos selecionados após segunda rodada

Year	**Title**	**Author**
2014	Security Requirements for Multimedia Archives (PARK, 2015)	Sang Bae Park
2019	System security requirements: A framework for early identification, specification and measurement of related software requirements (MERIDJI et al., 2019)	Meridji, Kenza et al.
2012	System Security Requirements Analysis: A Smart Grid Case Study (ZAFAR et al., 2014)	Zafar, Nauman et al.
2020	A Hybrid MCDM Approach of Selecting Lightweight Cryptographic Cipher Based on ISO and NIST Lightweight Cryptography Security Requirements for Internet of Things (NING et al., 2020)	Ning, Li. et al.
2014	Capturing security requirements for software systems (EL-HADARY; EL-KASSAS, 2014)	El-Hadary, Hassan et al.
2016	Data security and privacy preservation in cloud storage environments based on cryptographic mechanisms (KAANICHE; LAURENT, 2017)	Kaaniche, Nesrine et al.
2015	Effectiveness and performance analysis of model-oriented security requirements engineering to elicit security requirements: a systematic solution for developing secure software systems (SALINI; KANMANI, 2016)	Salini, S. Kanmani
2019	Quantifying Satisfaction of Security Requirements of Cloud Software Systems (NHLABATSI et al., 2021)	Nhlabatsi, Armstrong et al
2021	Stakeholder perspectives and requirements on cybersecurity in Europe (FISCHER-HÜBNER et al., 2021)	Fischer-Hubner, Simone et al

"*integridade*", "*disponibilidade*" ou "*confidencialidade*" foram incorporados. Adicionalmente, o termo "*modelagem de segurança*" foi incluído porque, ao revisar os resumos no Quadro 2, ficou evidente que esse termo também é utilizado para denotar a engenharia de requisitos de segurança. A expressão de busca finalizada é: *("Security software requirements" OR "security design" OR "security modeling") AND ("security algorithms" OR "cryptography" OR "data encryption" OR "authenticity" OR "integrity" OR "availability" OR "confidentiality") AND ("non-functional requirements" OR "NFR").*

Essa expressão de busca resultou em 50 publicações. Destas, foi realizada uma seleção manual de estudos que contenham propostas de ferramentas ou métodos aplicáveis à fase de coleta de requisitos, e 9 artigos foram extraídos (Quadro 3).

Tabela 3 – Publicações que Propõem Ferramentas ou Métodos Durante a Elicitação de Requisitos de Segurança

Ano	**Título**	**Autor**
2015	Describing early security requirements using use case maps (HASSINE; HAMOU-LHADJ, 2015)	Hassine, Jameleddine et al.
2015	Aspect-oriented security hardening of UML design models (MOUHEB et al., 2015)	Mouheb, Djedjiga et al.
2017	Selection security mechanisms in secure tropos (PAVLIDIS et al., 2017)	Michalis, Pavlidis et al.
2014	Capturing security requirements for software systems (EL-HADARY; EL-KASSAS, 2014)	El-Hadary, Hassan et al.
2017	A semi-automatic approach for eliciting cloud security and privacy requirements (ARGYROPOULOS et al., 2017)	Argyropoulos, Nikolaos et al.
2015	Effectiveness and performance analysis of model-oriented security requirements engineering to elicit security requirements: a systematic solution for developing secure software systems (SALINI; KANMANI, 2016)	Salini, S. Kanmani
2019	Quantifying Satisfaction of Security Requirements of Cloud Software Systems (NHLABATSI et al., 2021)	Nhlabatsi, Armstrong et al.
2019	A security modeling and verification method of embedded software based on Z and MARTE (HU; ZHUANG; ZHANG, 2020)	Hu, Xinwen et al.
2019	System security requirements: A framework for early identification, specification and measurement of related software requirements (MERIDJI et al., 2019)	Meridji, Kenza et al.

3.5.3 Resultados Parciais

A fusão das percepções das 2ª e 3ª rodadas resultou em um total de 18 artigos. Essas descobertas destacam uma tendência significativa em simplificar a elicitação de requisitos de segurança nas fases iniciais de projetos de *software*. Essa ênfase surge devido

à alta probabilidade de retrabalho quando tais requisitos são abordados apenas após o desenvolvimento.

Além disso, uma análise desses artigos revela um foco predominante na segurança de *software* em um contexto generalizado. No entanto, há uma notável falta de ênfase nos requisitos de *software* não funcionais. Em particular, há um interesse pronunciado nos requisitos de segurança da computação em nuvem, impulsionado pela acessibilidade distribuída e remota das soluções em nuvem, o que exige projetos de segurança robustos. Apesar de sistemas embarcados constituírem uma porção modesta dos estudos (11,8%), sua relevância no mercado atual destaca uma lacuna na análise de soluções específicas para esse fim. Contribuições nesse domínio são altamente significativas, dada a escassez de técnicas personalizadas para sistemas embarcados na literatura atual. Embora alguns métodos existentes sejam aplicáveis, muitos são generalizados, e alguns são orientados para aplicativos em nuvem. O *framework* ESSEF emerge como uma contribuição valiosa, abordando essa lacuna e aprimorando a análise de requisitos de segurança no nicho de sistemas embarcados.

3.5.4 Considerações Sobre os Requisitos

Segurança, como um elemento arquitetônico, deve ser inicialmente abordada em um nível mais alto (EL-HADARY; EL-KASSAS, 2014), sem adentrar em soluções criptográficas, técnicas de criptografia ou ferramentas. Esses detalhes mais finos são considerados durante o processo de refinamento de requisitos não funcionais em funcionais. Requisitos não funcionais e suas terminologias aderem a definições padronizadas encontradas em instituições como *IEEE*, *ECSS* e *ISO* (MERIDJI et al., 2019). Seguindo esses padrões e identificando termos repetidos por meio de uma interseção, os requisitos não funcionais de segurança podem ser enumerados da seguinte forma:

Utilização de técnicas de criptografia Registro ou histórico de dados do sistema Sistema modularizado, multifuncional Intercomunicação gerenciada Integridade de dados Privacidade de dados Disponibilidade de dados Confidencialidade de dados Autenticação de dados Controles de acesso e papéis de acesso Camadas de auditoria Gerenciamento de infraestrutura Dentro desses requisitos, os conceitos fundamentais de integridade, disponibilidade, confidencialidade e autenticidade servem como pilares para a modelagem de sistemas de segurança, formando a base para a maioria das ferramentas e métodos que propõem requisitos funcionais a partir de requisitos não funcionais. Aqui está uma descrição detalhada desses requisitos:

- **Confidencialidade:** A capacidade do sistema de identificar tentativas de acesso não autorizado e garantir que usuários não autenticados não possam acessar conteúdo protegido.

- **Disponibilidade:** A característica do sistema de manter redundância diante de escassez de recursos, como quedas de energia, corrupção de dados e problemas de fila de processamento. Um sistema com boa disponibilidade pode operar independentemente de falhas em uma ou mais de suas camadas.

- **Integridade:** Garantia de que o conteúdo protegido de um sistema permanecerá não corrompido (não deletado ou não modificado) mesmo no caso de acesso não autorizado.

- **Autenticidade:** A capacidade do sistema de autenticar usuários, diferenciar papéis de acesso e restringir recursos do sistema com base em um conjunto pré-definido de permissões para cada usuário.

A implementação de requisitos funcionais, seja em parte ou totalmente, é contingente ao ambiente computacional utilizado. A identificação precoce das demandas de segurança do projeto é imperativa. Por exemplo, um projeto que prioriza o armazenamento seguro de dados públicos deve priorizar a integridade para evitar a corrupção de dados em possíveis ataques. Por outro lado, a autenticidade, limitando o acesso a usuários específicos, pode ser menos crítica em tais cenários, dada a natureza pública inerente dos dados acessíveis sem credenciais. Além disso, um projeto que enfatiza a disponibilidade pode implementar um sistema de processamento redundante, garantindo operação contínua na ausência de qualquer recurso que prejudique a funcionalidade normal.

Considerar o ambiente computacional alvo é crucial durante a elicitação de requisitos de segurança, especialmente em sistemas embarcados como o *IoHT* (*Internet of Health Things*). Fatores como velocidade de autenticação e pegada de memória do sistema precisam ser considerados, adicionando uma fase extra na seleção de requisitos funcionais: avaliar a compatibilidade da solução de segurança com o sistema embarcado — um aspecto simplificado pelo *framework* ESSEF.

3.5.5 Resposta à Pergunta Principal

PQ) Qual é o processo para avaliar soluções de segurança no ciclo de vida do desenvolvimento de *software*?

R: Iniciar o processo envolve considerar os requisitos não funcionais da arquitetura do projeto. Diversas técnicas, métodos e ferramentas auxiliam no refinamento desses requisitos para estabelecer um modelo de requisitos funcionais para o sistema. Uma vez definidos, a equipe pode prosseguir para pesquisar soluções adequadas para cada requisito. No entanto, é crucial avaliar os recursos disponíveis no ambiente computacional alvo para garantir compatibilidade e eficiência na escolha de soluções seguras.

3.5.6 Respostas às Perguntas Secundárias

SQ1) Qual é o foco dos estudos de segurança para sistemas embarcados? ***R:*** Os sistemas embarcados recebem relativamente menos foco nos estudos de segurança dentro do período de tempo fornecido.

SQ2) Quais são os requisitos não funcionais essenciais para iniciar um projeto de design de segurança? ***R:*** Os requisitos padronizados incluem confidencialidade, autenticidade, integridade e disponibilidade. Esses parâmetros são definidos por instituições como *IEEE*, *ECSS* e *ISO*.

SQ3) Como selecionar requisitos de segurança com base na aplicação específica do sistema? ***R:*** A avaliação entre os requisitos mencionados é necessária para determinar a relevância para a aplicação. Por exemplo, a disponibilidade pode não ser crítica se a inoperância do sistema devido a violações de segurança ou falhas for aceitável.

SQ4) Como definir soluções seguras com base em requisitos não funcionais? ***R:*** Ferramentas e métodos auxiliam na seleção de soluções seguras para cumprir os requisitos funcionais correspondentes, como *frameworks* de conversão, diagramas *UML* e referências de implementação.

SQ5) Quais são as limitações do ambiente computacional que devem ser consideradas na seleção de uma solução segura? ***R:*** Fatores como velocidade de processamento, capacidade de memória e interfaces de comunicação precisam ser considerados. Algoritmos criptográficos, por exemplo, exigem processamento em tempo real e interfaces de interação.

3.6 Conclusão da Pesquisa

Durante a seleção de publicações e extração de dados, foi colocada uma ênfase notável na avaliação dos requisitos de segurança na etapa de design do sistema de software. Escolher requisitos não funcionais padronizados adaptados às necessidades do projeto evita gastos computacionais ou financeiros desnecessários durante o desenvolvimento.

Numerosos estudos destacaram ferramentas ou métodos que facilitam a derivação de requisitos funcionais a partir dos não funcionais. No entanto, é necessária uma exploração adicional para identificar aprimoramentos ou uma síntese desses métodos para aplicações específicas. Isso inclui o desenvolvimento de um *framework* para a coleta de requisitos funcionais em sistemas embarcados, destacando a importância do *framework* ESSEF.

4 Framework *ESSEF*

A estrutura de documentação científica fornece diretrizes para organizar e apresentar informações científicas, facilitando a comunicação padronizada. O *framework* ESSEF visa facilitar a seleção de soluções seguras em sistemas embarcados e documentar a arquitetura do projeto por meio de gráficos de interdependência. Isso aborda a lacuna observada na revisão sistemática. A estrutura operacional inclui:

- **Estágio 1** - Seleção de requisitos funcionais;
- **Estágio 2** - Refinamento de soluções por categorias;
- **Estágio 3** - Análise dos impactos nos recursos de hardware;
- **Estágio 4** - Seleção da solução de segurança;
- **Estágio 5** - Integração do modelo de árvore de requisitos.

O primeiro estágio envolve a escolha de requisitos funcionais de segurança e a construção de uma subárvore inicial usando um gráfico de interdependência. Estágios subsequentes refinam as soluções em categorias com base nos padrões ISO 25010, garantindo sistemas com boa segurança.

Durante a modelagem da árvore de requisitos, considera-se que uma categoria selecionada atende totalmente ao critério quando serve explicitamente ao propósito para o qual foi criada. Por exemplo, um algoritmo de *hashing* atende explicitamente ao requisito de integridade, pois seu objetivo é detectar qualquer mutação nos dados originais. Este caso de uso é representado na Figura 5 considerando as categorias que idealmente atendem a cada um dos requisitos de segurança.

Depois de gerar um submodelo para cada requisito funcional com categorias refinadas, a atenção se volta para o banco de dados de soluções para avaliar o impacto nos recursos de hardware necessários para implementar cada solução de segurança 3.5.6. Os recursos notáveis incluem:

- Percentual de ocupação de memória;
- Velocidade de processamento;
- Complexidade do código.

Esses fatores são amplamente observados no mercado de *software* atual e exercem uma influência significativa no dimensionamento dos componentes, impactando diretamente o custo total do projeto (RAVI et al., 2004).

Ao escolher as soluções, a subárvore final pode ser construída, delineando as contribuições das soluções para os recursos de *hardware*. Esta subárvore representa os efeitos adversos de cada solução nos recursos disponíveis no sistema embarcado. Por exemplo, optar por uma solução que consuma um percentual maior de memória em comparação com outras será representado na subárvore, conectado ao recurso de memória com uma indicação de contribuição negativa no gráfico.

No Estágio 4, a seleção final de uma das soluções é feita pelo usuário do *framework*, que pode optar pela solução com o melhor desempenho em cada um dos fatores mencionados.

Na fase final, todas as subárvores modeladas usando gráficos de interdependência podem ser consolidadas, formando um modelo abrangente da arquitetura de segurança do projeto. Este modelo integrado destaca o impacto de cada solução segura nos recursos de *hardware* relevantes para o sistema. Todo o processo é ilustrado no fluxo de execução mostrado na Figura 3.

4.1 Resultados

Os experimentos foram conduzidos utilizando o microcontrolador STM32, escolhido por sua popularidade no mercado, acessibilidade e adequação didática. Este microcontrolador é amplamente utilizado e possui uma vasta documentação, o que facilita a replicação e compreensão dos experimentos.

A avaliação foi restrita ao recurso de memória, através de uma análise estática. Esta escolha se deve ao objetivo principal do estudo, que é validar os métodos propostos pelo *ESSEF*, e não propor métodos eficazes para avaliação de recursos de hardware neste momento. A análise estática foi considerada suficiente para demonstrar a aplicabilidade do *framework* e identificar áreas de melhoria futura.

Para as medidas de velocidade e consumo, foram utilizadas as complexidades dos algoritmos em questão. Quanto mais complexo, e mais instruções necessárias para as operações, maior o tempo gasto para executar todo algoritmo. Consequentemente, quanto maior tempo gasto, maior o consumo, considerando que o tempo ativo do microcontrolador possui o maior consumo de memória.

A partir dos resultados obtidos, foi possível examinar as árvores geradas e tomar decisões informadas de maneira prática e rápida. As árvores de requisitos e as subárvores de impacto nos recursos de hardware permitiram uma visualização clara das implicações

de cada solução de segurança no uso da memória. Isso facilitou a seleção da solução mais adequada, considerando as restrições de hardware do sistema embarcado.

Além disso, destaca-se que outro pesquisador que utilize este *framework* teria o trabalho de avaliação da solução significativamente reduzido. O *framework* fornece uma estrutura organizada e padronizada para a seleção de soluções seguras, o que poupa tempo e esforço na fase de avaliação e decisão. Os gráficos de interdependência e as árvores de requisitos geradas pelo *framework* oferecem uma visão consolidada do impacto das soluções, permitindo que os pesquisadores se concentrem em outras etapas críticas do desenvolvimento do sistema embarcado.

Em conclusão, o uso do ESSEF demonstrou ser eficaz na seleção de soluções de segurança, fornecendo uma abordagem estruturada para avaliar e integrar essas soluções em sistemas embarcados. A validação do *framework* por meio do microcontrolador STM32 e a análise estática de memória confirmaram sua utilidade prática e potencial para futuras melhorias no campo de engenharia de requisitos.

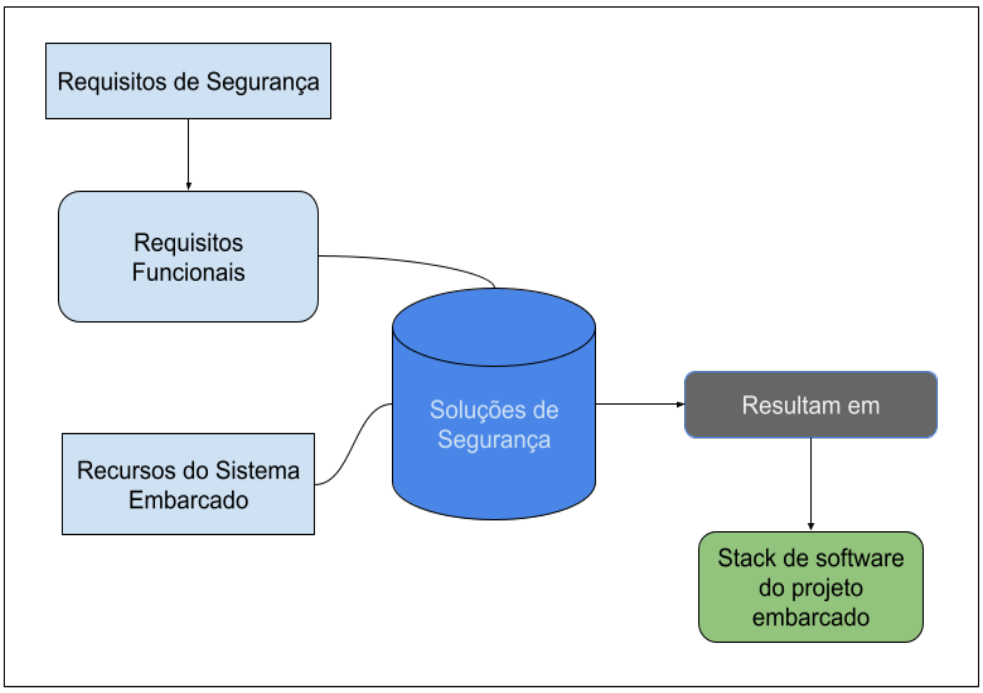

Figura 3 – Estágios do Framework ESSEF

4.2 Gráficos de Interdependência de Objetivos Soft (SIGs)

Originalmente introduzidos por Chung et al. (2012), os SIGs (*Soft Goal Interdependency Graphs*) servem como uma ferramenta valiosa para modelar e descrever requisitos não funcionais (NFR, de *Non-Functional Requirements*) e suas interdependências.

Esta abordagem envolve a decomposição de requisitos não funcionais em metas mais específicas, levando eventualmente a uma ou mais soluções que satisfaçam o *NFR*. Dentro dos tipos de objetivos representados nos gráficos *SIG*, temos:

1. Objetivo *Soft*: exibe o *NFR* a ser satisfeito pelo sistema.
2. Objetivo *Soft* Operacionalizável: representa soluções possíveis (operações, processos, representações de dados) ou alternativas de design que podem ajudar a satisfazer o *NFR*.
3. Objetivo de Reivindicação: mostra o refinamento entre objetivos *soft* ou a justificação relacionada a eles.

Os SIGs introduzem seis tipos de contribuições que influenciam a satisfação do objetivo *soft* pai positiva ou negativamente. No entanto, para o *framework* ESSEF, as contribuições foram reduzidas para simplificação do modelo. Neste *framework*, contribuições negativas são empregadas para modelar a sub-árvore de recursos de hardware, enquanto contribuições positivas são utilizadas para retratar o impacto de cada solução nos requisitos de segurança. Uma representação colorida foi adotada para distinguir entre diferentes tipos de objetivos, divergindo do método original que as diferenciava pela espessura do traço do elemento gráfico. Consequentemente, objetivos *soft* são representados em azul, objetivos de reivindicação em preto, e objetivos operacionalizáveis (representando soluções potenciais de segurança para implementação) são codificadas por cores em verde, para soluções com um impacto negativo menor nos recursos de hardware, e vermelho, para soluções com um impacto negativo mais significativo. A Figura 4 resume visualmente a notação descrita.

4.3 Passo 1 - Seleção de Requisitos Funcionais

Para iniciar o *framework*, o primeiro passo envolve a seleção de requisitos funcionais para o projeto alvo. A Figura 5 e as subseções a seguir fornecem exemplos práticos e orientações sobre como escolher cada requisito com base nos cenários de aplicação de sistemas embarcados. Compreender os conceitos dentro do escopo do projeto é crucial para identificar a necessidade desses requisitos.

4.3.0.1 **Integridade**

Em um sistema embarcado, garantir a integridade envolve proteger os dados contra adulteração em qualquer região de memória e em qualquer ponto durante a execução do código. Esse requisito é crucial para manter a imutabilidade dos dados, tanto durante o armazenamento quanto durante a transferência.

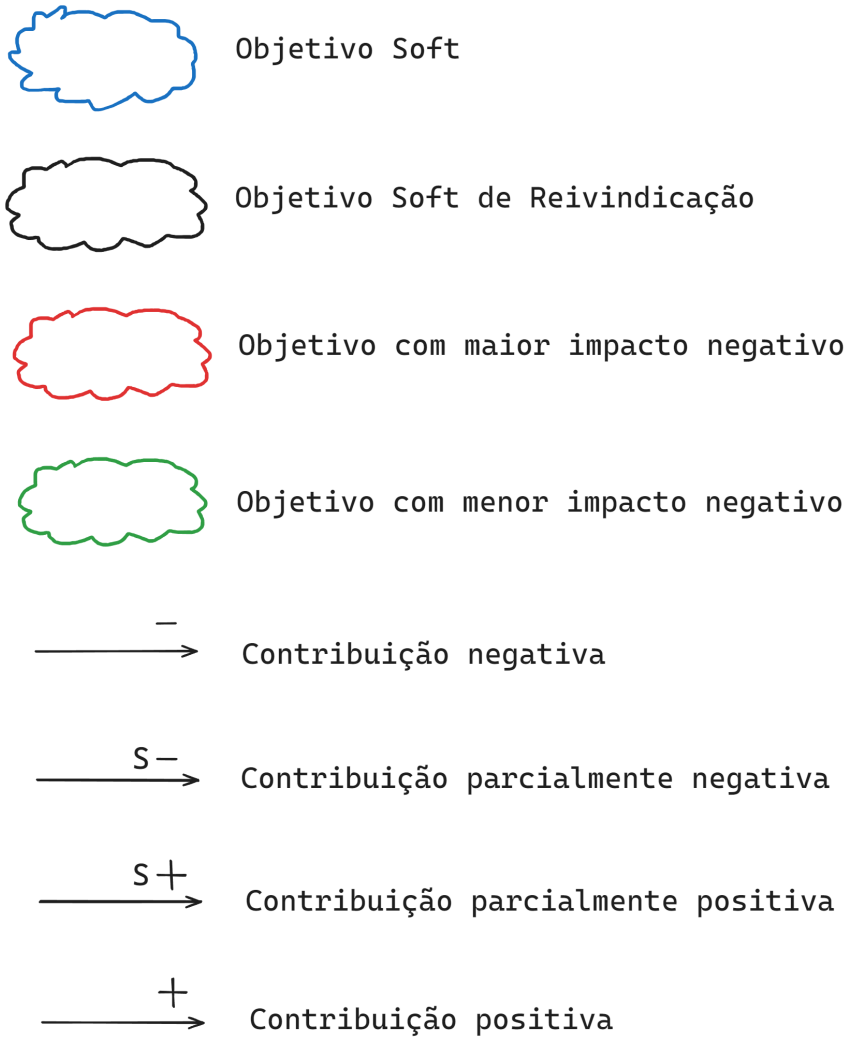

Figura 4 – Grafo de interdependência de objetivos (SIG)

Para dados armazenados, a imutabilidade pode ser garantida usando funções de proteção como *hash*, *Cyclic Redundancy Check* (CRC) ou métodos similares. Em termos práticos, soluções que abordam a integridade do projeto são implementadas sempre que a mutação de dados possa interromper a execução do *firmware*. Esse princípio também se aplica quando o *firmware* se comunica com sistemas externos por meio de interfaces como I2C, SPI, UART, *Bluetooth* ou *Wi-Fi* (RAVI et al., 2004). Nesse contexto, a integridade garante que os dados transmitidos permaneçam intactos e não adulterados.

Além disso, é importante considerar que a garantia da integridade dos dados não se limita apenas à prevenção de alterações não autorizadas, mas também à detecção e correção de quaisquer alterações acidentais ou maliciosas que possam ocorrer. Portanto, além de mecanismos de proteção robustos, é essencial incorporar técnicas de verificação e validação de dados para garantir a integridade do sistema como um todo.

Por fim, a integridade dos dados é um aspecto fundamental da segurança de sistemas embarcados, garantindo não apenas a confiabilidade das informações armazenadas

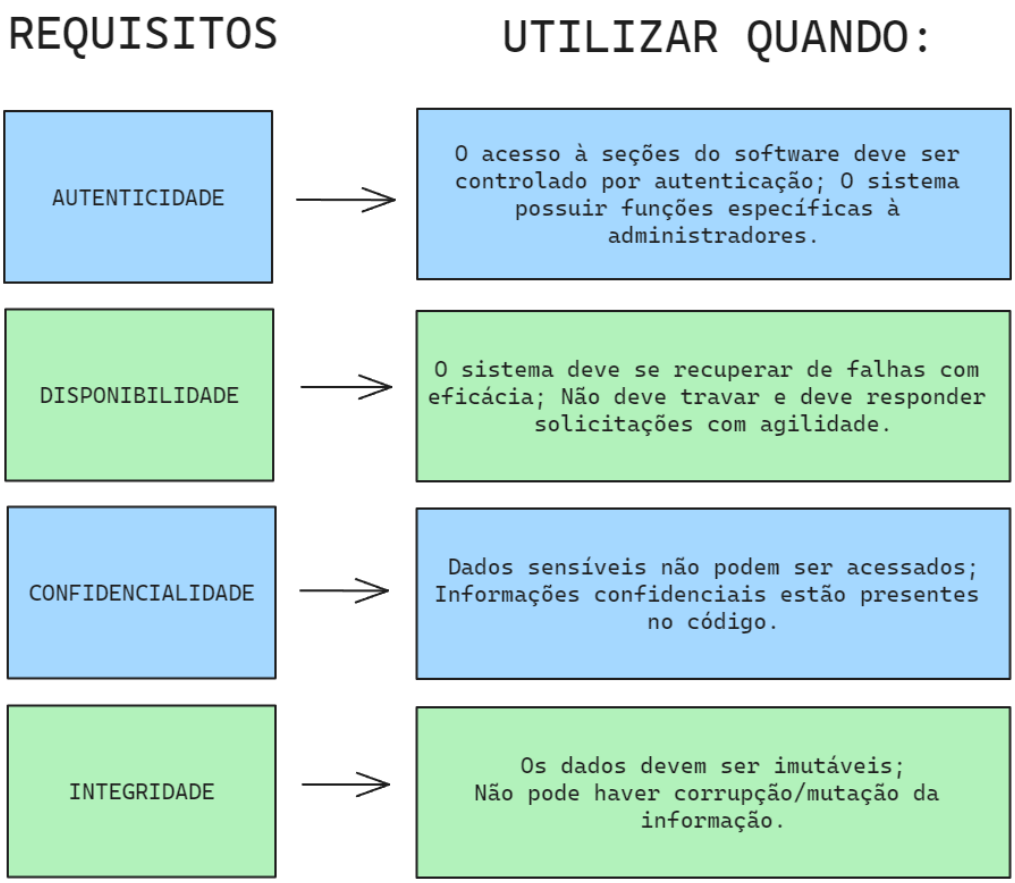

Figura 5 – Casos de uso de requisitos

e transmitidas, mas também a estabilidade e o funcionamento adequado do sistema em diferentes contextos e cenários de uso.

4.3.0.2 Confidencialidade

A confidencialidade visa proteger os dados do projeto contra interpretação, ou seja, seu conteúdo não deve ser compreensível mesmo se um usuário obtiver acesso a ele. Esse requisito pode ser aplicado tanto durante o armazenamento de dados quanto durante as transferências e geralmente é atendido por meio da implementação de funções de criptografia.

Ao criptografar os dados, pode-se garantir que, mesmo que um atacante consiga autenticar e obter os dados do domínio, ele não será capaz de interpretá-los e, consequentemente, não será possível usá-los como informações válidas. Nesse caso, a menos que o projeto também tenha funções que garantam a integridade, esses dados criptografados ainda podem ser modificados, potencialmente causando corrupção de dados e mau funcionamento do *firmware*.

Além disso, é importante ressaltar que a confidencialidade não se limita apenas à proteção de dados armazenados, mas também se estende às comunicações entre o sistema

embarcado e outros dispositivos ou sistemas. Ao garantir que todas as comunicações sejam criptografadas e protegidas adequadamente, pode-se evitar que informações sensíveis sejam interceptadas ou acessadas por partes não autorizadas, garantindo assim a privacidade e a segurança dos dados em trânsito.

Por fim, a confidencialidade desempenha um papel crucial na garantia da segurança global de sistemas embarcados, protegendo informações sensíveis contra acessos não autorizados e garantindo a integridade e a confiabilidade do sistema como um todo. Ao implementar medidas de segurança robustas e abordar adequadamente os requisitos de confidencialidade, os desenvolvedores podem garantir a proteção adequada dos dados e a confiança dos usuários finais no sistema.

4.3.0.3 **Disponibilidade**

O *firmware* é suscetível a vários tipos de ataques, muitos dos quais podem interromper a execução normal do programa (ALOSEEL et al., 2020). A disponibilidade, nesse contexto, refere-se à capacidade do sistema de se recuperar eficientemente de ataques ou falhas resultantes de incidentes de hardware ou *bugs*. Esse requisito torna-se crítico quando o sistema embarcado deve evitar tempo de inatividade, normalmente alinhando-se com soluções que garantem redundância de recursos ou que detectam e priorizam falhas e falhas internas.

Além disso, é fundamental considerar que a disponibilidade não se restringe apenas à capacidade de recuperação após falhas, mas também inclui medidas preventivas para evitar interrupções indesejadas no funcionamento do sistema. Isso pode envolver a implementação de técnicas de monitoramento proativo, como a detecção de anomalias e a prevenção de ataques conhecidos, para garantir que o sistema permaneça operacional mesmo diante de ameaças em potencial.

Em ambientes críticos em que a disponibilidade é de extrema importância, estratégias como a distribuição de carga e o balanceamento de carga podem ser adotadas para garantir que a capacidade do sistema seja adequadamente dimensionada e distribuída, evitando assim sobrecargas que possam comprometer sua capacidade de resposta.

Portanto, garantir a disponibilidade do *firmware* em sistemas embarcados é essencial para garantir a continuidade das operações e a confiabilidade do sistema como um todo. Ao implementar medidas robustas de recuperação e prevenção de falhas, os desenvolvedores podem garantir que o sistema esteja sempre disponível e pronto para responder às demandas do ambiente operacional.

4.3.0.4 Autenticidade

Este requisito é projetado para estabelecer distintos níveis de acesso para vários grupos de usuários dentro do sistema embarcado. Torna-se crucial em projetos com camadas de autenticação, como funcionalidades de login, garantindo que as informações permaneçam inacessíveis para usuários não autorizados.

Além disso, a autenticidade também desempenha um papel fundamental na garantia da integridade e da confiabilidade dos dados, assegurando que as informações recebidas ou transmitidas pelo sistema sejam originadas de fontes legítimas e não tenham sido adulteradas ou falsificadas durante o processo.

Para alcançar a autenticidade, podem ser implementadas várias medidas de segurança, como o uso de técnicas de criptografia e assinaturas digitais para verificar a autenticidade dos dados e a identidade dos remetentes. Além disso, sistemas de autenticação robustos, como a autenticação de dois fatores, podem ser empregados para garantir que apenas usuários autorizados tenham acesso aos recursos protegidos do sistema.

Em resumo, a autenticidade é essencial para garantir a segurança e a confiabilidade dos sistemas embarcados, protegendo contra acesso não autorizado e garantindo a integridade dos dados transmitidos e armazenados. Ao implementar medidas adequadas de autenticação e controle de acesso, os desenvolvedores podem garantir que seus sistemas permaneçam protegidos contra ameaças internas e externas.

4.3.0.5 Modelagem de Requisitos Funcionais

Após a seleção do conjunto de requisitos funcionais, podemos começar a construir a árvore de requisitos do projeto. Como exemplo, uma primeira sub-árvore foi criada considerando um sistema que deve implementar os quatro requisitos funcionais definidos. A sub-árvore está ilustrada na Figura 6.

Figura 6 – Sub-árvore de requisitos funcionais

4.4 Passo 2 - Refinamento das Soluções por Categorias

A segunda etapa do procedimento envolve o refinamento das soluções que atendem totalmente ou parcialmente aos requisitos de segurança identificados. Essas soluções são modeladas e inter-relacionadas com base em suas respectivas contribuições para cada requisito de segurança, seguindo o padrão do Grafo de Interdependência de Objetivos *Soft* (*SIG*), conforme representado na Figura 4.

Dentro do *framework* ESSEF, é empregado um banco de dados abrangente de soluções, estruturado de acordo com os padrões ISO 25010 (ISO, 2011) e categorizado em seções como:

- criptografia de dados;
- *hashing*;
- sistemas de autenticação; e
- redundância.

Para exemplificar a aplicação prática dessas soluções, casos de uso específicos são destacados, alinhando-os com os requisitos funcionais previamente selecionados, conforme demonstrado na Figura 7. Além desses casos ilustrativos, a categorização de soluções seguras é definida com base em seu quadro conceitual e aplicabilidade dentro do escopo do projeto designado.

4.4.1 Criptografia de Dados

A criptografia de dados em microcontroladores refere-se ao processo de proteção de informações sensíveis ou confidenciais armazenadas ou transmitidas por um microcontrolador por meio do uso de técnicas de criptografia. A criptografia é a ciência de transformar dados em texto criptografado que só pode ser descriptografado e compreendido por alguém que tenha a chave de descriptografia correta. Alguns fabricantes já incorporam esse tipo de tecnologia em suas arquiteturas de microcontroladores. Em 2014, a *Microchip* já apresentava em seu catálogo um módulo dedicado à criptografia de dados, juntamente com documentação simplificada que abstrai o usuário final do processo matemático complexo dessa técnica (WEEKLY, 2014).

Essa técnica é de extrema importância, pois mesmo que todas as portas de comunicação de um sistema embarcado estejam protegidas e o código tenha métodos de autenticação para controlar o acesso a informações privilegiadas, ainda é possível extrair informações de um microcontrolador por meio de outros processos, incluindo os bioquímicos (STROBEL et al., 2014).

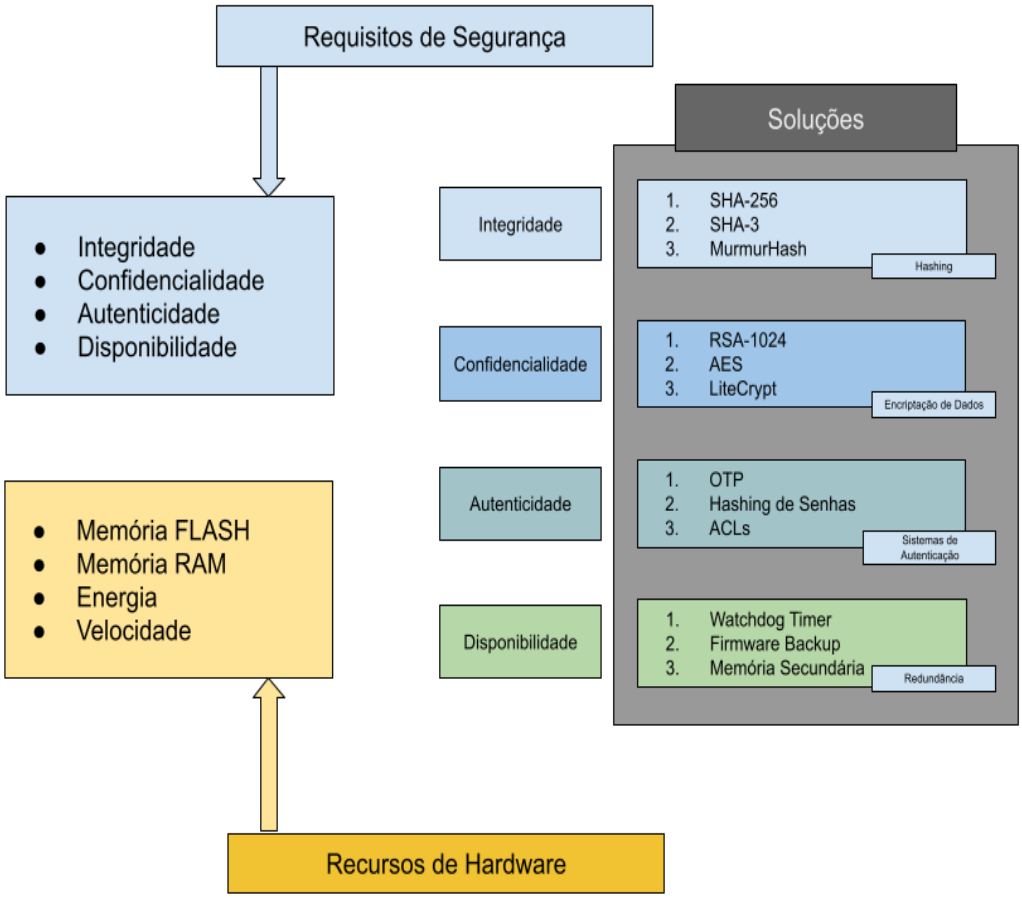

Figura 7 – Filtro de requisitos

Além disso, a criptografia de dados em microcontroladores favorece na garantia da segurança e privacidade das informações armazenadas e transmitidas pelo dispositivo. Ao proteger os dados com técnicas de criptografia robustas, os desenvolvedores podem impedir que informações sensíveis sejam interceptadas ou acessadas por partes não autorizadas, garantindo assim a confidencialidade e integridade dos dados em todas as etapas do processo de comunicação e armazenamento.

4.4.1.1 *Hashing*

A técnica de *hashing* é um processo de transformação de dados de entrada, geralmente de comprimento variável, em um valor de tamanho fixo chamado '*hash*' ou 'resumo'. Este *hash* é calculado de forma determinística usando uma função de *hash* geralmente bastante simples, como mostrado no código da Listagem 4.1.

Listing 4.1 – Exemplo de funçao simples de hashing em linguagem C

```
unsigned int hashFunction(const char *str) {
    unsigned int hash = 5381;
    int c;

    while ((c = *str++)) {
        hash = ((hash << 5) + hash) + c; // hash * 33 + c
    }

    return hash;
}
```

A função fornecida embaralha cada caractere dentro da *string* fornecida (*str*). Embora essa operação seja irreversível, ela serve como um método para validar a integridade ponta a ponta. Se o conteúdo de *str* for transmitido para outro componente do sistema, o componente receptor pode executar a mesma função de embaralhamento e comparar o resultado com os dados embaralhados originalmente calculados, esperando resultados idênticos.

As técnicas de *hashing* são amplamente utilizadas em segurança da informação, bancos de dados (para indexação eficiente), criptografia (para proteger senhas e dados confidenciais) e em muitas outras aplicações quando é necessário resumir ou verificar a integridade dos dados. Alguns exemplos de funções de *hash* bem conhecidas incluem *MD5*, *SHA-1*, *SHA-256* e *SHA-3*. A operação dos algoritmos *SHA* é feita por meio de criptografia de bloco, conforme descrito em (SELVAKUMAR; RATASTOGI, 2014).

Essa categoria abrange principalmente bibliotecas de *hashing* e funções que facilitam cálculos criptográficos. Tipicamente, essas técnicas são compatíveis com sistemas de baixos recursos, ajustando a complexidade do cálculo de *hashing* de acordo com os recursos de memória disponíveis do sistema alvo. No entanto, essa abordagem adaptativa pode levar a uma redução no fator de segurança da implementação.

4.4.2 Sistemas de Autenticação

A implementação de sistemas de autenticação desempenha um papel importante na proteção contra acesso não autorizado e na preservação da integridade dos dados. Em sistemas embarcados, devemos verificar a identidade dos usuários, dispositivos ou processos que interagem com o sistema. Em ambientes em que a segurança é primordial, como sistemas médicos, financeiros ou industriais, uma autenticação robusta evita violações potenciais.

Além de garantir a legitimidade das interações, os sistemas de autenticação em sistemas embarcados contribuem para a preservação da confidencialidade e integridade dos dados. Mecanismos de autenticação eficazes protegem contra ameaças como acesso não autorizado, adulteração de dados e interceptação de comunicação. Implementar métodos

de autenticação seguros, como senhas criptografadas, *tokens* ou biometria, fortalece a postura de segurança dos sistemas embarcados. Essas medidas devem ser consideradas em cenários onde a falha de autenticação pode resultar em danos para o sistema, como a perda de dados sensíveis ou comprometimento da operação.

Em resumo, incorporar uma autenticação robusta em sistemas embarcados é uma saída para mitigar riscos de segurança, preservar a integridade dos dados e garantir a operação confiável desses sistemas em ambientes desafiadores.

4.4.3 Redundância

Como os sistemas embarcados desempenham um papel crítico em aplicações críticas, a redundância e a disponibilidade surgem como elementos-chave para garantir a confiabilidade e o desempenho contínuo desses sistemas. A redundância, referindo-se à replicação de componentes críticos, destina-se à mitigação de falhas. Ao incorporar redundância em hardware ou *software*, os sistemas embarcados podem continuar operando mesmo se um de seus componentes falhar. Isso é particularmente crítico em ambientes em que falhas podem ter consequências graves, como na indústria automotiva ou em sistemas médicos.

Além disso, a disponibilidade é um indicador da eficácia de um sistema embarcado. Garantir alta disponibilidade significa minimizar o tempo durante o qual um sistema está inoperável. A introdução de mecanismos de redundância contribui para melhorar a disponibilidade, reduzindo o impacto de falhas imprevistas.

Conclusivamente, integrar estratégias de redundância e priorizar a disponibilidade são boas práticas na criação de sistemas embarcados resilientes e confiáveis. Essas medidas permitem que os engenheiros sustentem a continuidade operacional mesmo em cenários desafiadores, transmitindo maior confiança entre usuários e partes interessadas.

4.4.4 Modelando a subárvore após o refinamento

Nesta fase de modelagem, todas as categorias de soluções propostas pelo *framework* ESSEF são avaliadas de forma abrangente, considerando suas contribuições para cada requisito funcional, conforme representado na Figura 8. Além disso, esta etapa permite a identificação de contribuições entre requisitos refinados e outros requisitos funcionais, possibilitando a avaliação de suas interdependências e a seleção do elemento *SIG* correspondente.

4.5 Passo 3 - Análise dos impactos nos recursos de hardware

Para esta fase, determinadas soluções foram escolhidas como exemplos ilustrativos, levando em consideração sua popularidade, acessibilidade de código aberto e a

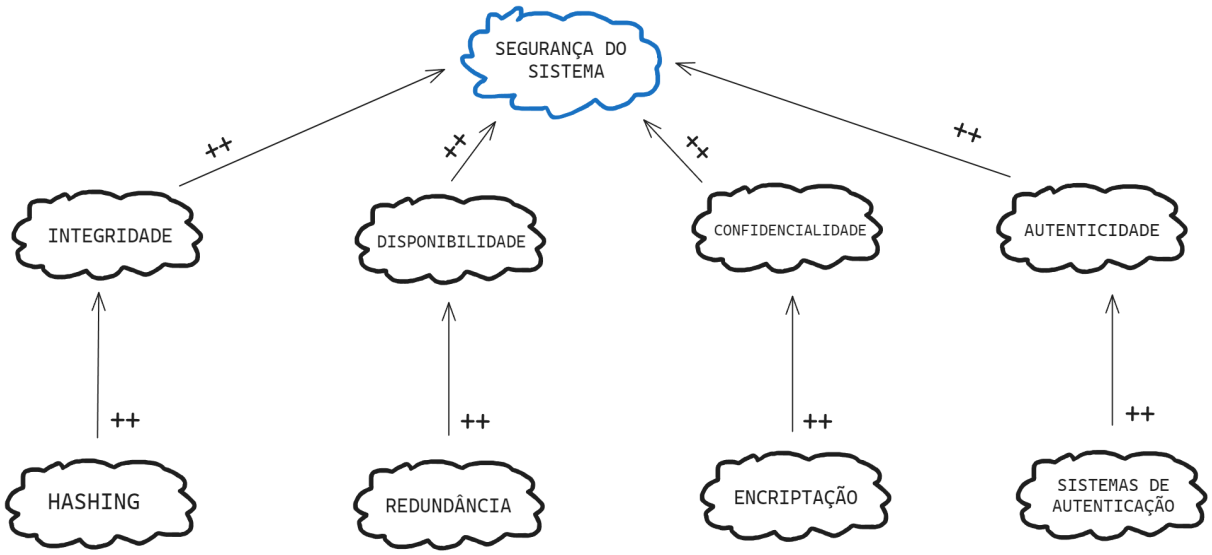

Figura 8 – Sub-árvore com recursos computacionais

existência de versões específicas adaptadas para sistemas embarcados (Seção 4.8). Essas soluções selecionadas são incluídas como ponto de referência neste estudo por meio de um repositório Git[1]. Todas as implementações foram conduzidas na plataforma de desenvolvimento *STElectronics*, especificamente a *STM32*, escolhida por sua ampla utilização e compatibilidade com os perfis de projetos para os quais o *framework* é recomendado (STMICROELECTRONICS, n.d.).

Nesta etapa, incorporamos recursos de *hardware* como metas suaves na arquitetura do projeto. Esses recursos, identificados por meio de estudos relevantes, demonstram exercer uma influência substancial na definição do sistema embarcado, abrangendo considerações tanto computacionais quanto financeiras (RAVI et al., 2004). Ao integrar na árvore de requisitos, a meta suave associada ao consumo de recursos computacionais é posicionada estrategicamente de baixo para cima. Este arranjo visa aprimorar a clareza e a compreensão da interação entre as metas suaves relacionadas à segurança e a meta suave referente ao consumo de recursos de *hardware*.

No caso de uso ilustrado na Figura 9, o requisito de integridade foi considerado para modelar a árvore de requisitos de segurança do projeto. Para a meta suave de recursos de *hardware*, foram aplicadas contribuições com base na comparação entre os possíveis algoritmos, que foram implementados em uma plataforma de desenvolvimento *STM32*. Nesta implementação, foi utilizado o STM32Cube IDE[2], com o qual foi possível extrair a porcentagem de memória ocupada no microcontrolador após a compilação (Quadros 4 a 6). Características fundamentais de cada um dos algoritmos também foram analisadas para extrair dados sobre complexidade de código e número de instruções para associar

[1] Disponível em: <https://github.com/essef-algorithms>
[2] https://www.st.com/en/development-tools/stm32cubeide.html

com a velocidade de execução.

Tabela 4 – Uso de memória do Murmur Hash

Região de Memória	Tamanho	Livre	Ocupado	Utilizaçao
RAM	20KB	18.45KB	1.55KB	7.73%
FLASH	64KB	59.97KB	4.03KB	6.29%

Tabela 5 – Uso de memória do SHA-256

Região de Memória	Tamanho	Livre	Ocupado	Utilizaçao
RAM	20KB	18.45KB	1.55KB	7.73%
FLASH	64KB	58.97KB	5.03KB	7.86%

Tabela 6 – Uso de memória do SHA-3

Região de Memória	Tamanho	Livre	Ocupado	Utilizaçao
RAM	20KB	18.45KB	1.55KB	7.73%
FLASH	64KB	56.93KB	7.07KB	11.05%

Esses dados sugerem que várias soluções podem utilizar diferentes níveis de recursos de *hardware*, como nos casos dos algoritmos analisados, nos quais o uso de memória FLASH é maior na função SHA-3 **??** do que no Murmur Hash 4. Portanto, os desenvolvedores são aconselhados a implementar cada nova biblioteca, função ou algoritmo em um código experimental dentro do sistema alvo. Idealmente, esse código experimental deve envolver exclusivamente a solução em consideração. Se código adicional for necessário para avaliação, certifique-se de que ele permaneça consistente em todas as comparações.

4.6 Passo 4 - Seleção da solução de segurança

Para atender ao requisito de integridade, a solução SHA-3 é destacada em vermelho na Figura 9, refletindo seu notável impacto negativo nos recursos de *hardware* (Quadro 6). Em contraste, a solução *Murmur Hash* apresenta o menor impacto negativo, marcado em verde (Quadro 4). Diante disso, os desenvolvedores podem optar por implementar o *Murmur Hash*, pois ele satisfaz efetivamente o objetivo de *Hashing*, atendendo assim ao requisito de integridade (meta suave) enquanto minimiza o consumo de recursos computacionais.

4.7 Passo 5 - Integração do Modelo da Árvore de Requisitos

Ao final do *framework*, os outros requisitos funcionais selecionados no primeiro passo (Seção 4.3) devem passar pelas etapas subsequentes (Seções 4.4, 4.5 e 4.6). O resultado

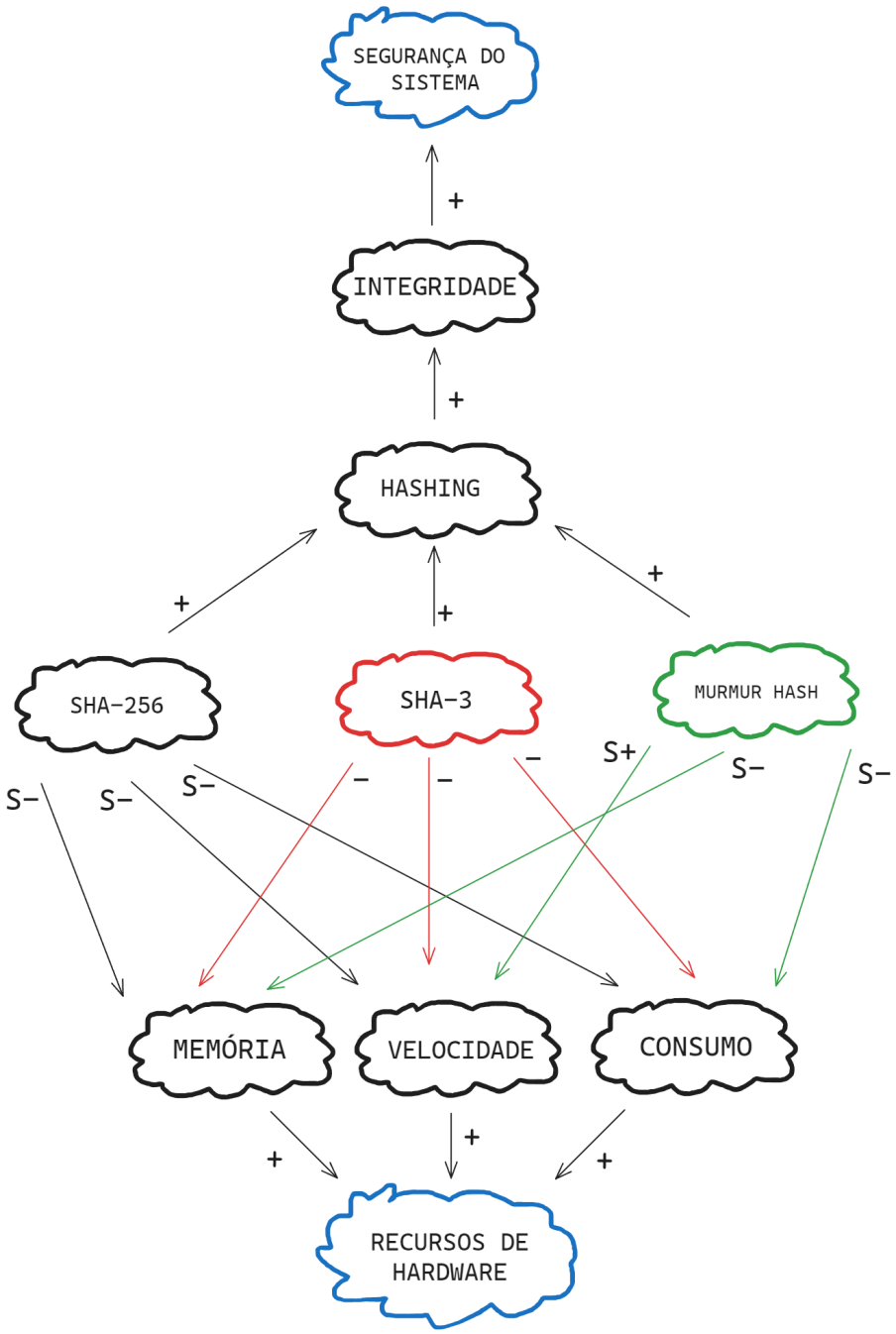

Figura 9 – Sub-árvore para integridade

final é a integração de todas as subárvores referentes a cada um dos requisitos selecionados, tornando-se um item consolidado da arquitetura do projeto. Considerando o caso de uso ilustrado na Figura 6, as subárvores representadas nas Figuras 10, 12 e 11 demonstram o resultado final da modelagem de todos os requisitos de segurança do projeto.

4.8 Base de Dados de Soluções e Repositório *Git*

As soluções abaixo foram selecionadas considerando sua popularidade no mercado, acesso aberto ao código e também por serem soluções já compatíveis com sistemas embarcados em termos de otimização e adequação à arquitetura. O código aberto para as soluções avaliadas no *framework* ESSEF está disponível no repositório Git e foi implementado na linguagem *C* para a plataforma *STM32* (MOSCA, 2023).

4.8.1 Soluções de Segurança para *Hashing*

- **Algoritmo de *Hash* Seguro de 256 bits (SHA-256)**
  - O *SHA-256* é um algoritmo de *hash* amplamente utilizado que produz uma mistura criptográfica dos dados de entrada, usando uma chave criptográfica e resultando em uma palavra de tamanho fixo de 256 *bits*. Oferece alta resistência a colisões e é adequado para sistemas embarcados com recursos moderados (GILBERT; HANDSCHUH, 2003).

- **Algoritmo de *Hash* Seguro 3 (SHA-3)**
  - O *SHA-3* é uma família de algoritmos de *hash* aprovada pelo Instituto Nacional de Padrões e Tecnologia (*NIST*) dos EUA que inclui várias versões com diferentes tamanhos de saída. Tem o mesmo princípio de funcionamento que outros algoritmos de *hash*, que é misturar os dados de entrada seguindo um fluxo de operações criptográficas predefinidas. O *SHA-3* é altamente seguro e pode ser implementado em sistemas embarcados ocupando relativamente menos recursos computacionais do que o *SHA-256* (DWORKIN, 2015).

- ***MurmurHash***
  - O *MurmurHash* é uma família de algoritmos de *hash* não criptográficos conhecidos por sua velocidade e simplicidade. Eles usam métodos simplificados, sem chave criptográfica, contendo apenas operações matemáticas complexas predefinidas. Devido a essa simplicidade, são adequados para sistemas embarcados, nos quais os recursos computacionais são escassos (WERLE, 2023).

4.8.2 Soluções de Segurança para Criptografia de Dados

- **Padrão de Criptografia Avançado (AES)**

  - O *AES* é um algoritmo de criptografia simétrica amplamente utilizado que oferece um alto nível de segurança. É eficiente o suficiente para ser implementado em sistemas embarcados e é usado para criptografar dados confidenciais (DAEMEN; RIJMEN, 1999).

- **Rivest–Shamir–Adleman (RSA)**
  - O *RSA* é outro algoritmo de criptografia de chave pública amplamente utilizado. Ainda é uma opção viável para sistemas embarcados, mas, dada sua complexidade e uso intensivo de recursos computacionais, requer mais recursos do sistema alvo (MILANOV, 2009).
  - O nível de segurança do *RSA* depende do tamanho da chave utilizada. Tamanhos maiores de chave oferecem maior segurança, mas requerem mais recursos computacionais e tempo para criptografia e descriptografia.

- **LiteCrypt**
  - *LiteCrypt* é um mecanismo de criptografia de *software* otimizado para sistemas embarcados. Ele é projetado para ser leve e eficiente em termos de recursos, tornando-o adequado para dispositivos com restrições de recursos (ASHGW, 2023).

4.8.3 Soluções de Autenticação

- **One-Time Password (OTP)**
  - A técnica de *One-Time Password* (OTP) é um método de autenticação que gera senhas únicas que só podem ser usadas uma vez. Essas senhas são válidas por um curto período de tempo, geralmente apenas para uma tentativa de login ou um intervalo curto de tempo, e são geradas dinamicamente.

    O uso de *OTP* em sistemas embarcados envolve um gerador que cria senhas únicas por meio de *hardware* ou *software* embarcado, com base em parâmetros predefinidos como tempo, semente inicial e algoritmo de *hash*. Essas senhas são distribuídas com segurança aos usuários e validadas em tempo real durante o login, tendo uma vida útil limitada, garantindo uma camada adicional de segurança. Para sistemas com recursos limitados, os algoritmos de geração de *OTP* podem ser otimizados, exigindo menos recursos sem comprometer a segurança.

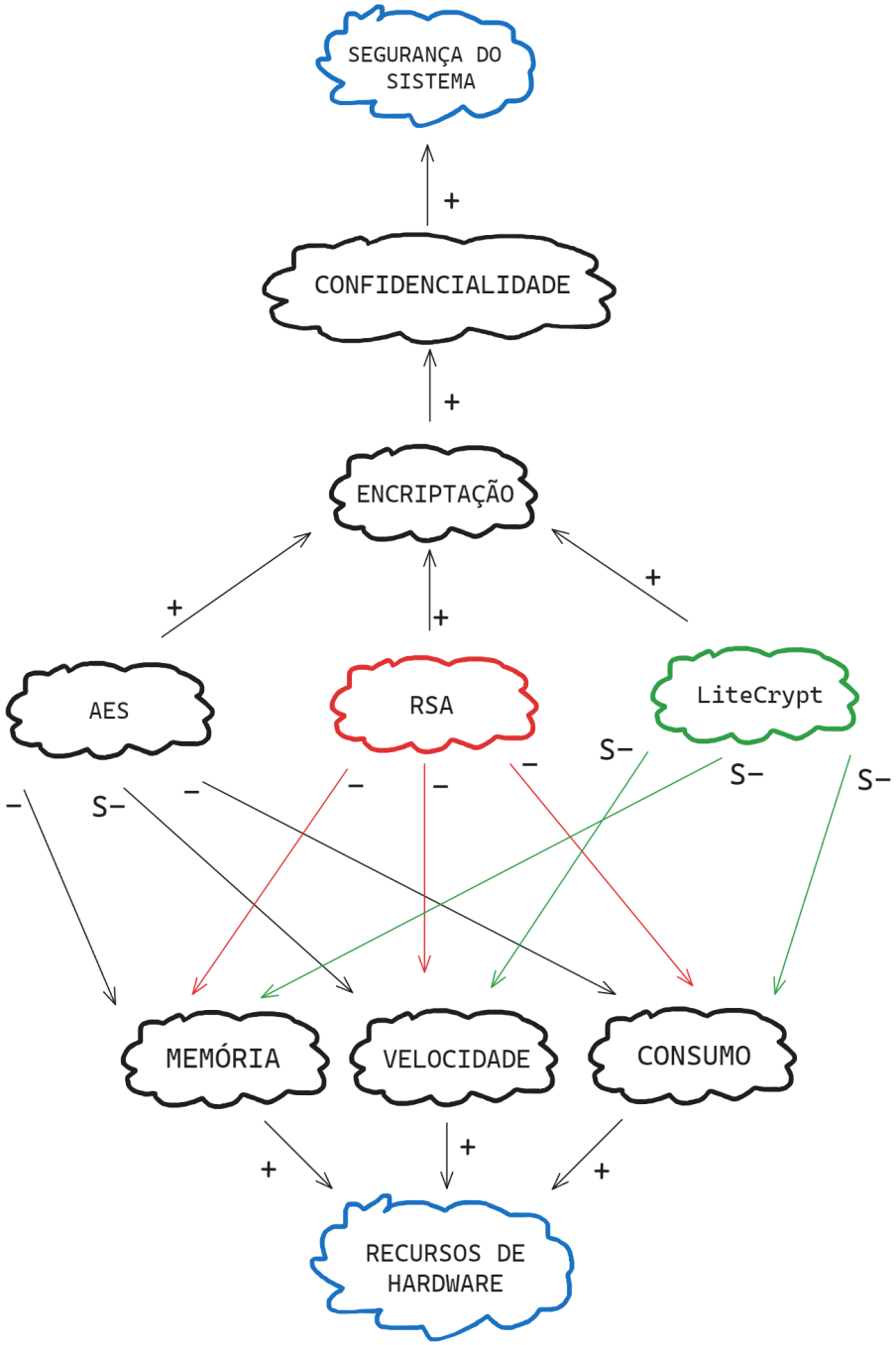

Figura 10 – Sub-árvore para confidencialidade

- ***Hashing* de Senhas**

  – Esta solução visa usar um dos algoritmos de *hash* disponíveis e aplicá-lo para armazenar senhas em estruturas de dados dentro do sistema embarcado. No momento do registro de uma credencial, a senha lida não é armazenada, mas sim o valor de seu *hash*, garantindo assim confidencialidade, integridade e posterior uso para autenticidade.

  No teste de uma senha, sempre que um usuário tenta acessar uma parte do

sistema, o valor inserido também passa por uma função de *hash* e depois é comparado com os valores armazenados para a credencial deste usuário.

- **Listas de Controle de Acesso (ACLs)**
  - *ACLs* são listas de permissões que determinam quem tem acesso a recursos específicos. Elas são usadas em sistemas embarcados para controlar o acesso a dados e funções.

4.8.4 Soluções de Disponibilidade

- ***Timers* do *Watchdog***
  - O *Timer* do *Watchdog* (WDT) é uma característica crítica em microcontroladores, projetada para monitorar e garantir a operação estável do sistema. Ele opera como um temporizador de contagem regressiva que requer redefinições periódicas para evitar reinicializações indesejadas ou falhas do sistema. Se o software do sistema falhar ou não redefinir o *WDT* dentro de um intervalo predefinido, o temporizador expira e causa uma reinicialização do microcontrolador. Essa característica protege contra falhas de software, travamentos ou *loops* infinitos, garantindo a integridade e confiabilidade do sistema, especialmente em aplicações críticas nas quais a falha não é aceitável. A configuração adequada do *WDT* é essencial para manter a operação estável do microcontrolador. Estudos considerados nesta pesquisa avaliam a importância do *Watchdog* em sistemas embarcados com recursos limitados (PONT; ONG, 2002).

- **Backup de *Firmware***
  - A técnica de backup de *firmware* em sistemas embarcados é um método de segurança que envolve duplicar o *firmware* ativo do dispositivo em uma localização segura. Isso é geralmente feito antes de uma atualização ou modificação do *firmware*. O *firmware* existente é copiado para uma área de backup protegida, permitindo, em caso de falha na atualização ou corrupção do novo *firmware*, que o sistema retorne ao estado funcional anterior. Esse *backup* é vital para garantir a continuidade operacional e minimizar os riscos associados a atualizações mal sucedidas, contribuindo para a confiabilidade dos sistemas embarcados.

- **Bloqueio de JTAG e Depuração**

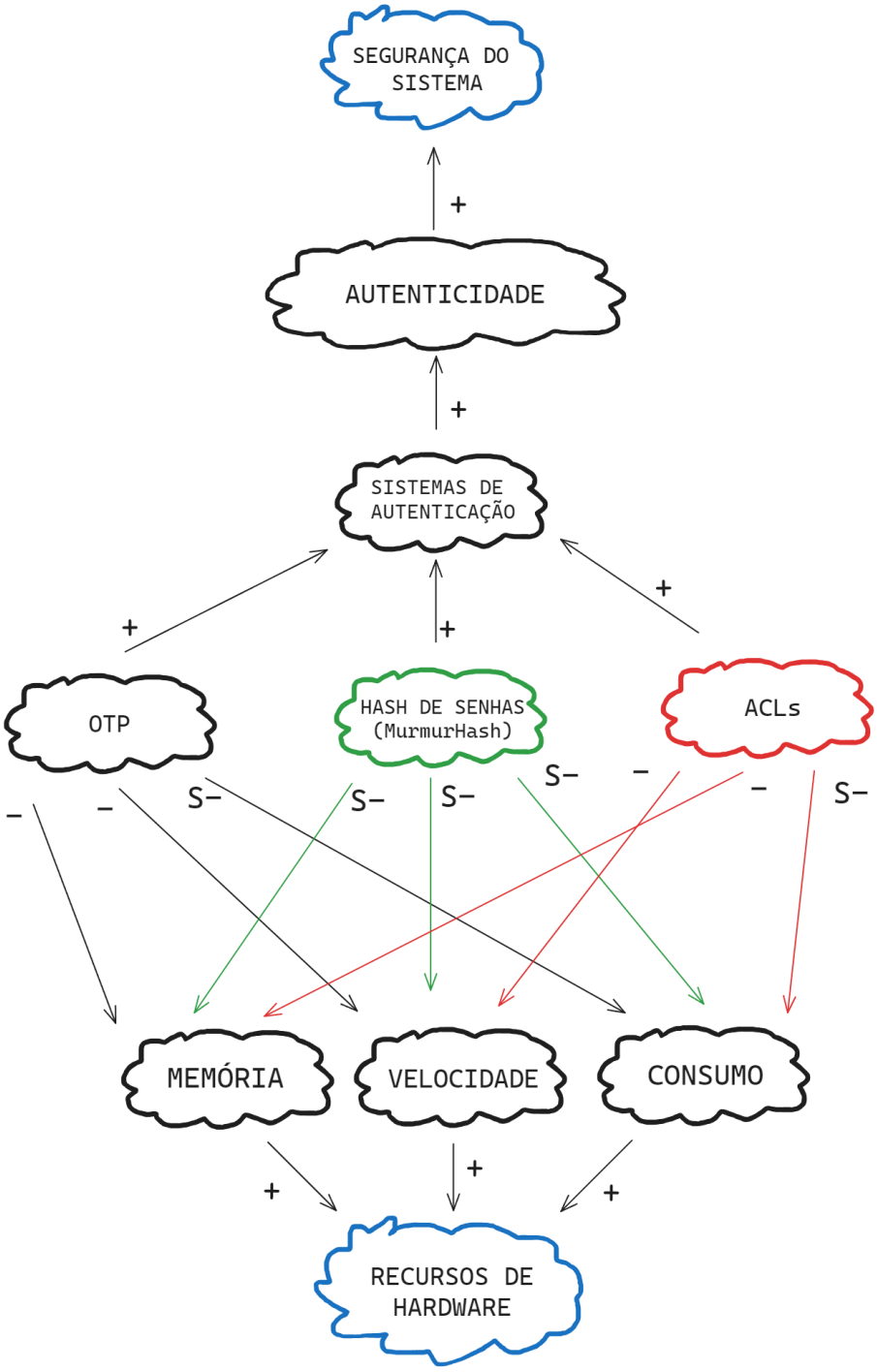

Figura 11 – Sub-árvore para autenticidade

– Bloquear o acesso através da porta *Joint Test Action Group* (JTAG) e recursos de depuração impede que os invasores acessem o microcontrolador durante a inicialização e, portanto, interrompam sua operação maliciosamente. Isso também favorece a confidencialidade do *firmware*, uma vez que o invasor não obtém as informações do código que está sendo executado no sistema embarcado (SGUIGNA, 2019).

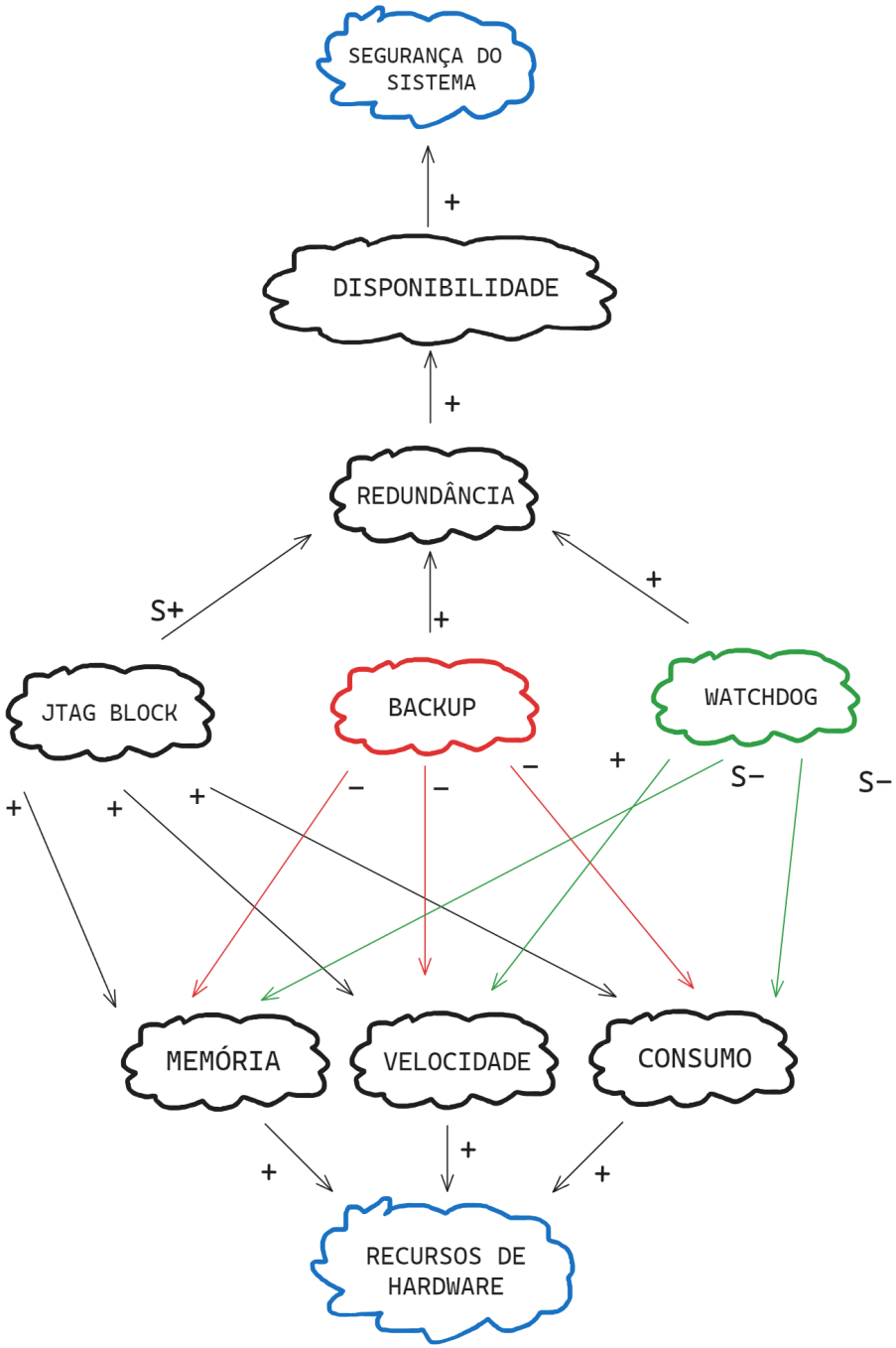

Figura 12 – Sub-árvore para disponibilidade

5 Conclusão

O *Framework* de Avaliação de Segurança de Sistemas Embarcados (ESSEF), proposto neste estudo, tem como objetivo fornecer suporte para equipes de engenharia implementarem soluções de segurança de forma fácil e sistemática em sistemas embarcados. Este trabalho destina-se a ser aplicado em documentação e arquitetura de software, oferecendo aos desenvolvedores uma abordagem estruturada para refinar requisitos funcionais de segurança e modelar a arquitetura de segurança.

O ESSEF serve como uma solução prática para o gerenciamento e documentação de projetos de software nesse contexto. Ele aborda questões como a identificação e análise de requisitos de segurança, a modelagem de arquitetura segura e a implementação de mecanismos de proteção adequados. Além disso, o *framework* é projetado para ser flexível e adaptável às necessidades específicas de cada projeto, permitindo que os desenvolvedores personalizem e estendam suas funcionalidades conforme necessário.

Aproveitando o *Soft Goal Interdependency Graph* (SIG), o *framework* oferece uma visualização clara e abrangente de todo o espectro de requisitos de segurança dentro do sistema, mapeando intrincadamente as soluções implementadas para abordar cada um desses requisitos. Esta abordagem inovadora não apenas melhora a compreensão e comunicação das medidas de segurança implementadas, mas também facilita uma análise aprofundada de seu impacto nos recursos computacionais.

A aplicação prática do ESSEF também desempenha um papel fundamental na priorização das considerações de segurança. Ele fornece aos desenvolvedores uma visão panorâmica inestimável da arquitetura de segurança, capacitando-os a tomar decisões bem fundamentadas ao longo das etapas de desenvolvimento de sistemas embarcados. Em essência, o ESSEF surge como um aliado poderoso na busca por segurança aprimorada e tomada de decisões estratégicas no intricado cenário do desenvolvimento de sistemas embarcados.

Para validar o escopo do *framework*, são essenciais mais testes em diversas arquiteturas de microcontroladores, indo além do *STM32* inicialmente usado. Essa expansão verificará a adaptabilidade do ESSEF a diversos contextos de sistemas embarcados. Considerando a natureza evolutiva das ameaças de segurança, recomenda-se a atualização contínua do banco de dados de soluções de segurança (Seção 4.8) para incorporar novas práticas e contramedidas.

O desenvolvimento de métricas específicas para avaliar a eficácia das soluções implementadas e a extensão do ESSEF para incluir automação na aplicação e teste de soluções apresentam áreas de pesquisa promissoras. Essas iniciativas contribuirão para

fortalecer a utilidade e aplicabilidade do ESSEF no dinâmico campo da segurança de sistemas embarcados. Em conclusão, o ESSEF não apenas serve como um ativo valioso no desenvolvimento atual de sistemas embarcados, mas também possui potencial para refinamento e aplicabilidade adicionais em resposta ao cenário de segurança em constante evolução.

5.1 Trabalhos Futuros

O *Framework* de Avaliação de Segurança de Sistemas Embarcados (ESSEF), proposto neste estudo, tem como objetivo fornecer suporte para futuras equipes de engenharia implementarem soluções de segurança de forma mais fácil e sistemática em sistemas embarcados. Este trabalho destina-se a ser aplicado na documentação e arquitetura de software, oferecendo aos desenvolvedores uma abordagem estruturada para refinar requisitos funcionais de segurança e modelar a arquitetura de segurança.

Um ponto de melhoria imediato identificado é a utilização de um banco de dados que considere a vulnerabilidade das soluções avaliadas. O nível de vulnerabilidade pode então ser levado em conta no momento de atribuir os níveis de contribuição de satisfação dos requisitos pais de cada solução.

Pesquisas e desenvolvimentos adicionais nesta área podem levar a um *framework* mais robusto com dados e experimentos adicionais. Especificamente, os trabalhos futuros podem se concentrar em:

- **Validação em Diversas Arquiteturas de Microcontroladores:** Realizar testes em uma variedade de arquiteturas de microcontroladores além do *STM32* inicialmente utilizado para verificar a adaptabilidade do ESSEF a diversos contextos de sistemas embarcados.

- **Atualização Contínua do Banco de Dados de Soluções de Segurança:** Atualizar regularmente o banco de dados de soluções de segurança para incorporar novas práticas e contramedidas em resposta às ameaças de segurança em constante evolução.

- **Desenvolvimento de Métricas de Avaliação Específicas:** Criar métricas específicas para avaliar a eficácia das soluções implementadas, fornecendo medidas quantitativas para avaliar as medidas de segurança.

- **Extensão do ESSEF para Automação:** Estender o ESSEF para incluir automação na aplicação e teste de soluções de segurança, agilizando o processo de implementação e aumentando a eficiência.

Essas iniciativas contribuirão para fortalecer a utilidade e aplicabilidade do ESSEF no campo dinâmico da segurança de sistemas embarcados, capacitando os desenvolvedores a tomarem decisões bem fundamentadas e criarem sistemas mais seguros.

Referências

ALOSEEL, A. et al. Analytical review of cybersecurity for embedded systems. *IEEE Access*, IEEE, v. 9, p. 961–982, 2020. 18, 28, 47

ARGYROPOULOS, N. et al. A semi-automatic approach for eliciting cloud security and privacy requirements. 2017. 37

ASHGW. *LiteCrypt: A lightweight cryptography library*. 2023. <https://github.com/ashgw/litecrypt>. Accessed: 2023-12-09. 57

CHUNG, L. et al. *Non-functional requirements in software engineering*. [S.l.]: Springer Science & Business Media, 2012. v. 5. 18, 21, 24, 43

DAEMEN, J.; RIJMEN, V. Aes proposal: Rijndael. Gaithersburg, MD, USA, 1999. 57

DUTT, N.; JANTSCH, A.; SARMA, S. Toward smart embedded systems: A self-aware system-on-chip (soc) perspective. *ACM Transactions on Embedded Computing Systems (TECS)*, ACM New York, NY, USA, v. 15, n. 2, p. 1–27, 2016. 27, 29

DWORKIN, M. J. Sha-3 standard: Permutation-based hash and extendable-output functions. Morris J. Dworkin, 2015. 56

EL-HADARY, H.; EL-KASSAS, S. Capturing security requirements for software systems. *Journal of advanced research*, Elsevier, v. 5, n. 4, p. 463–472, 2014. 29, 36, 37, 38

FISCHER-HÜBNER, S. et al. Stakeholder perspectives and requirements on cybersecurity in europe. *Journal of information security and applications*, Elsevier, v. 61, p. 102916, 2021. 36

GILBERT, H.; HANDSCHUH, H. Security analysis of sha-256 and sisters. In: SPRINGER. *International workshop on selected areas in cryptography*. [S.l.], 2003. p. 175–193. 56

HASSINE, J.; HAMOU-LHADJ, A. Describing early security requirements using use case maps. In: SPRINGER. *SDL 2015: Model-Driven Engineering for Smart Cities: 17th International SDL Forum, Berlin, Germany, October 12-14, 2015, Proceedings 17*. [S.l.], 2015. p. 202–217. 37

HU, X.; ZHUANG, Y.; ZHANG, F. A security modeling and verification method of embedded software based on z and marte. *Computers & Security*, Elsevier, v. 88, p. 101615, 2020. 37

ISO, I. Iec25010: 2011 systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models. *International Organization for Standardization*, v. 34, n. 2910, p. 208, 2011. 49

KAANICHE, N.; LAURENT, M. Data security and privacy preservation in cloud storage environments based on cryptographic mechanisms. *Computer Communications*, Elsevier, v. 111, p. 120–141, 2017. 36

LÁZARO, J. et al. I2csec: A secure serial chip-to-chip communication protocol. *Journal of Systems Architecture*, Elsevier, v. 57, n. 2, p. 206–213, 2011. 28, 30

MALAN, R.; BREDEMEYER, D. et al. Functional requirements and use cases. *Bredemeyer Consulting*, Citeseer, 2001. 23

MERIDJI, K. et al. System security requirements: A framework for early identification, specification and measurement of related software requirements. *Computer Standards & Interfaces*, Elsevier, v. 66, p. 103346, 2019. 29, 31, 36, 37, 38

MILANOV, E. The rsa algorithm. *RSA laboratories*, p. 1–11, 2009. 57

MOSCA, D. *Essef Algorithms Repository*. 2023. Accessed: 2023-12-02. Disponível em: <https://github.com/daniellmosca/essef-algorithms>. 56

MOUHEB, D. et al. *Aspect-oriented security hardening of UML design models*. [S.l.]: Springer, 2015. v. 2105. 37

NHLABATSI, A. M. et al. Quantifying satisfaction of security requirements of cloud software systems. *IEEE Transactions on Cloud Computing*, IEEE, 2021. 36, 37

NING, L. et al. A hybrid mcdm approach of selecting lightweight cryptographic cipher based on iso and nist lightweight cryptography security requirements for internet of health things. *IEEE Access*, IEEE, v. 8, p. 220165–220187, 2020. 36

PAJA, E.; DALPIAZ, F.; GIORGINI, P. Modelling and reasoning about security requirements in socio-technical systems. *Data & Knowledge Engineering*, Elsevier, v. 98, p. 123–143, 2015. 17

PARK, S. B. Security requirements for multimedia archives. *Advances in Multimedia*, Hindawi Limited London, UK, United Kingdom, v. 2015, p. 9–9, 2015. 36

PAVLIDIS, M. et al. Selecting security mechanisms in secure tropos. In: SPRINGER. *Trust, Privacy and Security in Digital Business: 14th International Conference, TrustBus 2017, Lyon, France, August 30-31, 2017, Proceedings 14*. [S.l.], 2017. p. 99–114. 37

PONT, M. J.; ONG, R. H. Using watchdog timers to improve the reliability of single-processor embedded systems: Seven new patterns and a case study. In: CITESEER. *Proceedings of the First Nordic Conference on Pattern Languages of Programs*. [S.l.], 2002. p. 159–200. 59

RAVI, S. et al. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)*, ACM New York, NY, USA, v. 3, n. 3, p. 461–491, 2004. 17, 30, 42, 45, 53

SALINI, P.; KANMANI, S. Effectiveness and performance analysis of model-oriented security requirements engineering to elicit security requirements: a systematic solution for developing secure software systems. *International Journal of Information Security*, Springer, v. 15, p. 319–334, 2016. 17, 29, 36, 37

SELVAKUMAR, A.; RATASTOGI, R. Study the function of building blocks in sha family. *arXiv preprint arXiv:1402.1314*, 2014. 51

SEN, A.; JAIN, S. An agile technique for agent based goal refinement to elicit soft goals in goal oriented requirements engineering. In: IEEE. *15th International Conference on Advanced Computing and Communications (ADCOM 2007)*. [S.l.], 2007. p. 41–47. 26

SGUIGNA, A. *Securing the JTAG Interface*. 2019. Accessed: 2023-09-12. Disponível em: <https://www.asset-intertech.com/resources/blog/2019/07/securing-the-jtag-interface/>. 60

STMICROELECTRONICS. *STM32 Evaluation Tools Portfolio*. n.d. Accessed: 2023-12-02. Disponível em: <https://www.st.com/resource/en/product_presentation/stm32_eval-tools_portfolio.pdf>. 53

STROBEL, D. et al. Microcontrollers as (in) security devices for pervasive computing applications. *Proceedings of the IEEE*, IEEE, v. 102, n. 8, p. 1157–1173, 2014. 19, 49

WEEKLY, E. Product of the week: Microchip adds data encryption to pic microcontrollers. *Electronics Weekly*, v. 2594, p. 18, 2014. Disponível em: <http://www.electronicsweekly.com>. 49

WERLE, J. *MurmurHash.c Repository*. 2023. Accessed: 2023-12-09. Disponível em: <https://github.com/jwerle/murmurhash.c>. 56

ZAFAR, N. et al. System security requirements analysis: A smart grid case study. *Systems Engineering*, Wiley Online Library, v. 17, n. 1, p. 77–88, 2014. 36