Alex dos Santos Xavier

PROVENDO INTEGRIDADE MULTI-NÍVEL A MEDIDORES INTELIGENTES DE ENERGIA ELÉTRICA

Curitiba - PR, Brasil

Agosto de 2017

Alex dos Santos Xavier

PROVENDO INTEGRIDADE MULTI-NÍVEL A MEDIDORES INTELIGENTES DE ENERGIA ELÉTRICA

Dissertação de mestrado apresentado ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial à obtenção do título de Mestre.

Pontifícia Universidade Católica do Paraná - PUCPR

Programa de Pós-Graduação em Informática - PPGIa

Orientador: Prof. Dr. Altair Olivo Santin

Curitiba - PR, Brasil

Agosto de 2017

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central

X3p
2017

Xavier, Alex dos Santos
Provendo integridade multi-nível a medidores inteligentes de energia elétrica / Alex dos Santos Xavier ; orientador: Altair Olivo Santin. – 2017.
97, [1] f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2017
Bibliografia: f. 79-82

1. Medidores elétricos – Medidas de segurança. 2. Redes elétricas inteligentes. 3. Processamento de dados em tempo real. I. Santin, Altair Olivo. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 22. ed. – 621.31

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

PUCPR
GRUPO MARISTA

ATA DE SESSÃO PÙBLICA

DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 03/2017

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA – PPGIa
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ - PUCPR

Em sessão pública realizada às 14h00 de 28 de Agosto de 2017, no Sala 220 – Escola de Negócios, ocorreu a defesa da dissertação de mestrado intitulada **"Provendo Integridade Multi-nível a Medidores Inteligentes de Energia Elétrica"** apresentada pelo aluno **Alex dos Santos Xavier**, como requisito parcial para a obtenção do título de **Mestre em Informática**, na área de concentração **Ciência da Computação,** perante a banca examinadora composta pelos seguintes membros:

Prof. Dr. Altair Olivo Santin (Orientador)- PUCPR

Prof. Dr. Marcelo Eduardo Pellenz – PUCPR

Prof. Dr. Fabio Kurt Schneider – UTFPR

Prof. Dr. Arlindo Luis Marcon Junior - IFPR

Após a apresentação da dissertação pelo aluno e correspondente arguição, a banca examinadora emitiu o seguinte parecer sobre a tese:

Membro	Parecer	
Prof. Dr. Altair Olivo Santin	(X) Aprovada	( ) Reprovada
Prof. Dr. Marcelo Eduardo Pellenz	(X) Aprovada	( ) Reprovada
Prof. Dr. Fabio Kurt Schneider	(X) Aprovada	( ) Reprovada
Prof. Dr. Arlindo Luis Marcon Junior	(X) Aprovada	( ) Reprovada

Portanto, conforme as normas regimentais do PPGIa e da PUCPR, a tese foi considerada:

(X) **APROVADO**

(aprovação condicionada ao atendimento integral das correções e melhorias recomendadas pela banca examinadora, conforme anexo, dentro do prazo regimental)

( ) **REPROVADO**

E, para constar, lavrou-se a presente ata que vai assinada por todos os membros da banca examinadora.
Curitiba, 28 de Agosto de 2017.

Prof. Dr. Altair Olivo Santin

Prof. Dr. Marcelo Eduardo Pellenz

Prof. Dr. Fabio Kurt Schneider

Prof. Dr. Arlindo Luis Marcon Junior

Este trabalho é dedicado a minha família, amigos e professores, que tiveram paciência e me ajudaram nesta trajetória.

AGRADECIMENTOS

Agradeço ao Vilmar Abreu e ao Alison Lando por me ajudarem neste trabalho, disciplinas e ser um ponto de referência. Agradeço ao meu orientador Altair Santin pela sua paciência e ensinamentos e finalmente a minha família e amigos por me apoiarem e me darem força para continuar e aprender com os meus próprios erros.

"Criamos a época da velocidade, mas nos sentimos enclausurados dentro dela. A máquina, que produz abundância, tem-nos deixado em penúria. Nossos conhecimentos fizeram-nos céticos; nossa inteligência, empedernidos e cruéis. Pensamos em demasia e sentimos bem pouco. Mais do que de máquinas, precisamos de humanidade. Mais do que de inteligência, precisamos de afeição e doçura. Sem essas virtudes, a vida será de violência e tudo será perdido.

(Charles Chaplin)

RESUMO

O *Smart Meter* é um dispositivo fundamental para as companhias de distribuição de energia elétrica, porém este componente está exposto a ameaças físicas e lógicas, sendo que não há uma solução integrada que o proteja. O objetivo deste trabalho é apresentar uma solução que proteja o *Smart Meter*, utilizando técnicas de detecção de fraude multi-sensores e proteção lógica para microcontroladores, baseados na arquitetura 8051, aplicados em sistemas de medição elétrica. A implementação foi realizada utilizando um sistema operacional de tempo real e apresentando 95,93% de melhora no desempenho para detecção de violações de *tampering*, apesar de provocar um aumento de 324,52% em relação a leitura de memória.

Palavras-chave: Segurança Física e Lógica. Integridade Multi-nível. Detecção de Fraude. *Smart Meter*. Sistema de Tempo Real

ABSTRACT

The Smart Meter is a key component for power distribution companies, but this component is exposed to physical and logical threats and there is no integrated solution to protect it. The objective of this work is to present a solution that protects the Smart Meter using multi-sensor fraud detection and logic protection techniques for microcontrollers, based on the 8051 architecture, applies in electrical measurement systems. The implementation was performed using a real-time operating system and exhibiting 95.93% improvement in performance for detecting tampering violations, despite causing a 324.52% increase in memory reading

Keywords: Physical and Logical Security. Multilevel Integrity. Tampering Detection. Smart Meter. Real-time Operating System.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABRADEE	Associação Brasileira de Distribuidores de Energia Elétrica
ADC	Analog-to-Digital Converter
AMI	Advanced Measure Infrastructure
AMR	Automatic Meter Reading
CAK	Cumulative Attestation Kernel
CE	Computer Engine
DAC	Discretionary Access Control
DDoS	Distributed Denial of Service
GPL	General Public License
INMETRO	Instituto Nacional de Metrologia, Qualidade e Tecnologia
IoTMAC	Internet of Thing Mandatory Access Control
ISP	In-System-Programming
ISR	Interrupt Service Routine
LOMAC	Low Water-Mark Mandatory Access Control
MAC	Mandatory Access Control
MCU	Microcontroller Unit
MPU	Microprocessor Unit
MPU	Memory Protection Unit
MIPS	Millions of Instructions Per Second

MMU	Memory Management Unit
NAN	Neighborhood Area Network
NIST	National Institute of Standards and Technology
RAM	Random Access Memory
ROM	Read Only Memory
RTOS	Real Time Operation System
SCADA	Supervisory Control and Data Acquisition
SDCC	Small Device C Compiler
SFR	Special function register
SoC	System on Chip
TyTAN	Tiny Trust Anchor for Tiny Devices

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Ashton[1] cita que é preciso capacitar os computadores com os seus próprios meios de coleta de informação, para que possam "ver", "ouvir"e "sentir cheiros"por si próprios. Neste modo compreendendo o mundo sem a limitação dos dados introduzidos pelo homem.

O medidor de energia elétrica é um instrumento que tem por finalidade medir a energia consumida por uma casa ou indústria e gravar os registros desse consumo. Os medidores inteligentes de energia, os *Smart Meters*, são dispositivos eletrônicos que, além de realizar a medição de energia, são capazes de enviar os dados de consumo para uma central e contribuir com a manutenção da distribuição de energia, desligando a energia fornecida pela companhia elétrica ou enviando alarmes de possíveis ataques ou adulterações do medidor.

A interconexão entre os *Smart Meters*, a coleta de informações de maneira autônoma e constante podem criar novas vulnerabilidades e riscos a esses sistemas e a privacidade dos usuários. Os ataques mais comuns a esses dispositivos são: a interceptação da comunicação entre os sensores e o ataque direto ao dispositivo, uma vez que, geralmente, esses dispositivos não possuem proteção contra os ataques físicos. Além disto, a parte lógica em nível de sistema pode ser atacada, por ser constituída de componentes baseados em sistemas.

1.2 Motivação

Os dados obtidos pelo *Smart Meter* são relevantes para a concessionária de energia elétrica, pois estes dados são utilizados para calcular o faturamento da empresa. Sem os registros precisos de consumo obtidos pelo *Smart Meter* não há como a concessionária realizar a cobrança correta referente a despesa do usuário.

Os *Smart Meters* também coletam informações sobre a qualidade da energia fornecida para os usuários, como: queda, elevação e interferências de tensão na rede elétrica. Estes dados contribuem para que a distribuidora mantenha a qualidade de seus serviços. Por fim os *Smart Meters* possuem sensores e algoritmos para detectar possíveis furtos e fraudes na rede.

No Brasil, de acordo com o presidente da Associação Brasileira de Distribuidores de Energia Elétrica, Nelson Fonseca Leite[2], no ano de 2012, as perdas na distribuição no Brasil ficaram em 16,5%. Considerando a produção nacional, a energia perdida foi de aproximadamente 25TWh, o suficiente para suprir o estado do Paraná por um ano, por exemplo.

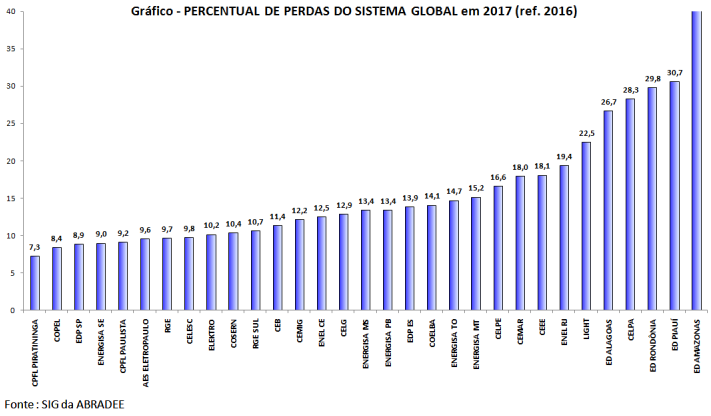
A perda de energia pode ocorrer por motivos técnicos pelo aquecimento dos fios condutores, ocasionados pela própria corrente elétrica, ou por perdas comerciais. Há duas modalidades de perdas comerciais: fraude e furto de energia.

É considerado furto o desvio de energia elétrica para o consumo ilegal, fraude é a adulteração da fiação elétrica ou dos dispositivos de medição de energia para que seja contabilizado apenas uma fração do consumo real de energia. A Figura 1 elaborada pela ABRADEE ilustra as perdas de energias globais e realiza uma comparação das perdas técnicas e comerciais.

Mesmo em países que já utilizam *Smart Meters* a fraude e furto de energia são problemas conhecidos. Um relatório emitido pelo FBI em 2012 citando fontes confidências[4] relata que, em Puerto Rico, os *Smart Meters* foram adulterados pelos funcionários da própria empresa de energia elétrica, sendo que era cobrado um valor entre 300 e 1.000 dólares para modificar os medidores residenciais e cerca de 3.000 dólares para os comerciais.

A medida que as informações fornecidas pelo *Smart Meter* são utilizadas para a tomada de decisões, visando a manutenção do sistema, é importante garantir a integridade e confiabilidade dos dados recebidos e enviados pelo *Smart Meter*. Inria Li *et al*[5] relatam como o ataque ao *Smart Meter* pode ser o ponto inicial para corromper toda a rede de distribuição de energia elétrica.

Como todo *software*, o *firmware* do medidor está sujeito a vulnerabilidades e, logo que detectada, a vulnerabilidade deve ser corrigida e as correções



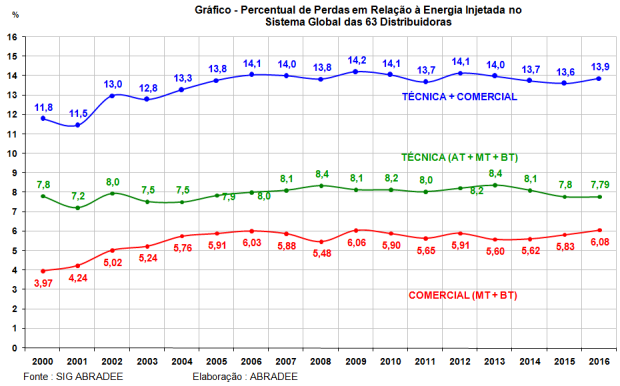


Figura 1 – Perdas de energia durante a transmissão de energia elétrica, extraído de [3]

disponibilizadas no menor tempo possível. Porém diferente da atualização de um *firmware* de um computador pessoal, os *firmwares* dos *Smart Meters* devem ser submetidos a certificação de agências reguladoras competentes.

No Brasil o INMETRO é a entidade responsável pela certificação do *software* dos *Smart Meters*, a agência garante que a metrologia do sistema está correta, ou seja, que a medição e os valores detectados pelo *Smart Meter* são exatos.

Nos Estados Unidos da América a agência competente para a certificação é o NIST. Em palestras com pessoas vinculadas ao NIST foi relatado que o menor tempo para a certificação de um *firmware* pelo NIST é de aproximadamente 45 dias, esse tempo é bastante elástico, mantendo em risco um dos principais dispositivos

do Smart Grid das companhias elétricas.

Os trabalhos encontrados na literatura possuem soluções de segurança que não englobam todas as ameaças que envolvem o *Smart Meter*, pois há pouca integração entre o *hardware* e *software* embarcado. Mohammad *et al.* [6] em seu trabalho apresentam técnicas para detectar algumas fraudes referente ao furto de energia e a abertura da tampa do medidor. Porém essas técnicas não são utilizadas de forma integrada e após a detecção da fraude o sistema não possui contramedidas para diminuir ou mitigar o impacto.

Tangsunantham *et al.* [7] sugerem utilizar técnicas de detecção de fraude utilizando a corrente das fases, porém se for utilizado a fase de uma residência vizinha, por exemplo, a fraude não é detectada, pois a solução não considera a corrente de todas as fases do medidor para a análise da fraude.

Para que seja possível a detecção das diversas ameaças e fraudes conhecidas, o medidor deve executar um sistema operacional preemptivo, ou seja, logo que ocorrer uma interrupção de *hardware* provocada por um sensor de fraude (como a abertura de tampa) o sistema deve atender a interrupção rapidamente e retornar a tarefa que estava executando antes da interrupção.

O uso de preempção busca diminuir o tempo de resposta à ameaça sem afetar as demais atividades e fornecer um nível de prioridade entre as fraudes. Por exemplo durante um ataque DDoS (*Distribuited Denial of Service*) a um *Smart Meter*, um sistema convencional ficaria lento e tratando exclusivamente as mensagens recebidas pela rede, tornando o dispositivo vulnerável a ataques físicos que ocorram durante este período. Com a utilização da preempção mesmo durante um ataque via rede o *Smart Meter* é capaz de responder pontualmente as ameaças físicas.

Alguns SoC de medição de energia, como o MAX71637 da Maxim Integrated [8], possuem uma arquitetura de *hardware* que oferece a proteção de leitura e escrita em espaços determinados em memória, mas não são todas as arquiteturas que oferecem essa proteção de baixo nível. Os circuitos integrados de medição de energia desenvolvidos pela Teridian, atualmente produzidos pela Silergy, são um exemplo de SoC para *Smart Meter* disponíveis do mercado e que não possuem essa proteção

de memória.

A utilização de um sistema de tempo real (RTOS, *Real Time Operating System*) com mecanismo de proteção de integridade multi-nível, busca prover a segurança de acesso e modificação em partes da memória considerados críticos para o funcionamento e a medição do *Smart Meter* em arquiteturas que não oferecem proteção de memória via *hardware*. Ao garantir a proteção de memória em porções críticas o sistema se torna mais resistente a fraudes, evitando que usuários adulterem a metrologia do sistema e provoquem prejuízo financeiro a distribuidora de energia elétrica.

1.3 Objetivos

O objetivo desse trabalho é demonstrar como um *Smart Meter* de baixo poder de processamento pode ter sua integridade preservada considerando os ataques conhecidos atualmente. A proposta é composta por um conjunto de sensores para detectar a possível violação física (*tampering*) do dispositivo, conta com a implementação de *software* para a detecção de vários tipos de fraude conhecidas e um sistema que oferece proteção a espaços de memória com informações críticas.

A proposta pretende oferecer proteção a um *Smart Meter* com o núcleo de processamento 80515, processador com arquitetura herdada no 8051. Para o 80515 foi criado o *Port* de um sistema operacional de tempo real (RTOS), que irá gerenciar suas tarefas de modo preemptivo e fornecer proteção a regiões críticas de memória em tempo real.

Além disto, tem o intuito de prover proteção lógica aos acessos a memória com base no modelo de integridade multi-nível Biba[9] para sistemas embarcados com restrições de armazenamento e processamento. Como o modelo de integridade utilizado consiste em regras de um controle de acesso mandatório, assunto principal desta dissertação, este será denominado IoTMAC (*Internet of Things Mandatory Access Control*).

A seguir são listados os objetivos específicos e as principais atividades que compreendem este trabalho:

1. Portar o código do RTOS para a arquitetura 80515;
2. Implementar a detecção de múltiplas fraudes;
3. Implementação de segurança multínivel (IoTMAC);
4. Testar e avaliar as implementações.

1.4 Contribuições Esperadas

Demonstrar que um microcontrolador (*e.g.*, o 80515) com baixa capacidade de processamento possui a capacidade necessária para executar sistemas operacionais de tempo real e com proteção lógica de memória.

Prover uma solução integrada de *software* e *hardware* para proteger o *Smart Meter*, mesmo que este esteja comprometido.

1.5 Organização

O restante deste documento está organizado da seguinte maneira. O Capítulo 2 contempla o referencial teórico. O Capítulo 3 aborda os trabalhos relacionados. O Capítulo 4 apresenta a elaboração da proposta. O Capítulo 5 apresenta a avaliação da proposta a partir dos dados coletados. O Capítulo 6 apresentando as conclusões e perspectivas para pesquisas futuras.

2 REFERENCIAL TEÓRICO

Neste capítulo será discutido o referencial teórico sobre os temas da dissertação, será realizada uma breve explicação sobre o Smart Grid e a importância dos *Smart Meters* para a rede. Também serão discutido as características e funcionalidades de um *Smart Meter*, os modelos de fraudes e furtos conhecidos atualmente, as demandas do mercado e a justificativa para a escolha de um SoC de medição de energia para o projeto.

2.1 Sistema Elétrico (*Grid*)

Grid, como explicado por Xi Fang, *et al.*[10], se refere a um sistema elétrico que suporta uma ou mais dessas quatro funcionalidades: geração, transmissão, distribuição e controle da eletricidade. A *Grid* tradicional possui apenas o fluxo de energia em uma única direção, da geração até o consumidor, como pode ser visto na Figura 2.

O controle e monitoramento da *Grid* tradiconal normalmente é feito através de funções de comando e controle. Um sistema típico utilizado pelas empresas de energia é o controle de supervisão e aquisição de dados (SCADA)[11], porém tais sistemas oferecem um monitoramento limitado da *Grid*, com alcance até a rede de transmissão e não possibilitam um monitoramento em tempo real.

Por esses motivos o monitoramento e controle de uma *Grid* convencional não é padronizado. Estima-se que nos Estados Unidos, considerado um dos sistemas de energia elétrica mais avançados do mundo, menos de um quarto da rede de distribuição esteja equipado com sistemas de informação e comunicação[12].

Levando em consideração que cerca de 90% das interrupções de energia ocorrem na rede de distribuição[11] é compreensível que a evolução da *Grid* aconteça onde existam as maiores dificuldades de controle, ou seja, na rede de distribuição.

Os primeiros projetos para melhorar o controle e monitoramento da rede de

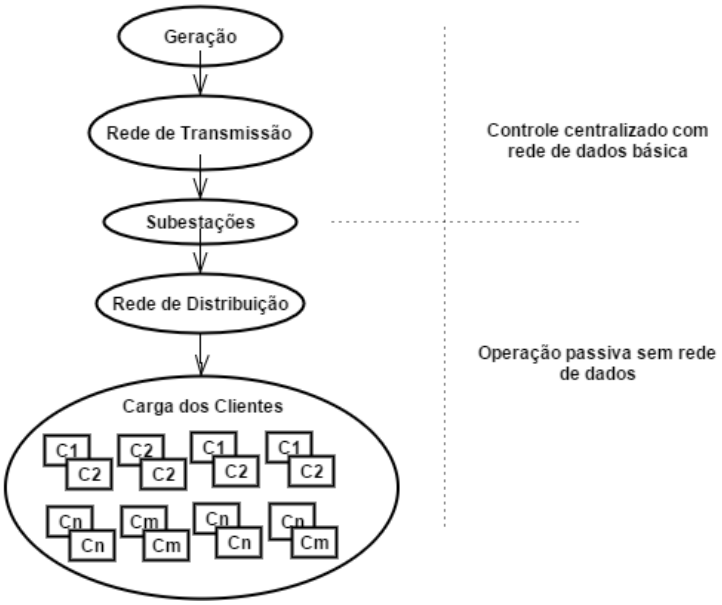


Figura 2 – *Grid* convencional, adaptado de [11]

distribuição utilizaram o sistema de leitura automática de medidor (AMR), que são medidores eletrônicos de energia que possibilitavam a leitura remota dos dados coletados.

2.2 *Smart Meters*

A evolução dos medidores eletromecânicos para os medidores eletrônicos permitiu a leitura remota e a geração automática de faturas para o consumidor. Apesar da automatização do processo de faturamento, o fluxo de energia e de dados não se modificou. A Figura 3 ilustra que para ambos os processos, manual e leitura automática, a única vantagem foi a automatização do processo de faturamento da energia consumida, sem alterar qualquer interação com a *Grid*. Este foi o início da utilização dos medidores eletrônicos.

O medidor eletrônico representado na Figura 3 não pode ser considerado um *Smart Meter*, pois sua única característica é o envio de informações. Um *Smart Meter* possui funções de análise, a verificação da qualidade de energia fornecida pela distribuidora e a detecção de fraudes.

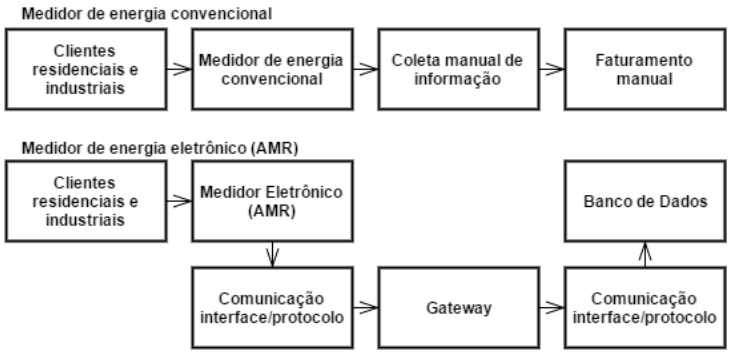


Figura 3 – Arquitetura dos medidores convencionais e AMR, adaptado de [11]

O aumento da cogeração de energia, utilizando painéis solares, levou as concessionárias de energia elétrica a investirem em medidores com novas funcionalidades, permitindo um melhor controle do fluxo de energia pela rede. A geração de energia deixa de ser, exclusivamente, da concessionária para o consumidor, mas a energia também pode ser gerada no consumidor. Com a alteração do fluxo convencional de produção de energia da *Grid*, houve a necessidade da evolução dos medidores de energia elétrica, ou seja, a criação de equipamentos capazes de medir o fluxo de energia consumida e fornecida pelo consumidor.

Além da mudança no fluxo de energia da *Grid* convencional, se iniciaram os estudos de novas formas de cobrança para o consumidor como, por exemplo, a energia "pré-paga", os postos-tarifários baseados no horário de consumo e como o cliente pode acessar o medidor e utilizar essas informações.

A integração de novas funcionalidades para o medidor, a mudança no fluxo de energia e a comunicação em um duplo sentindo, ou seja, da central de distribuição para o medidor e vice-versa, surgiu uma nova categoria de medidores, a Infraestrutura de Medição Avançada (AMI).

A Figura 4 demonstra a evolução dos medidores e suas funcionalidades. Atualmente as principais características de um *Smart Meter* são[13]:

1. Leitura remota;

2. Carregar dados de perfil;

3. Fornecer dados de consumo para o cliente e terceiros autorizados;
4. Opção de tarifas variáveis por tempo de uso;
5. Gerenciamento remoto do medidor;
6. Redução remota da demanda;
7. Conexão/Desconexão remota;
8. Qualidade da energia;
9. Fornecer o preço da tarifa para o consumidor;
10. Tarifa dinâmica e resposta a demanda;
11. Detecção de falha de energia;
12. Detecção de uso de energia em relação à demanda específica;
13. Detecção e correção de fraude e furtos;

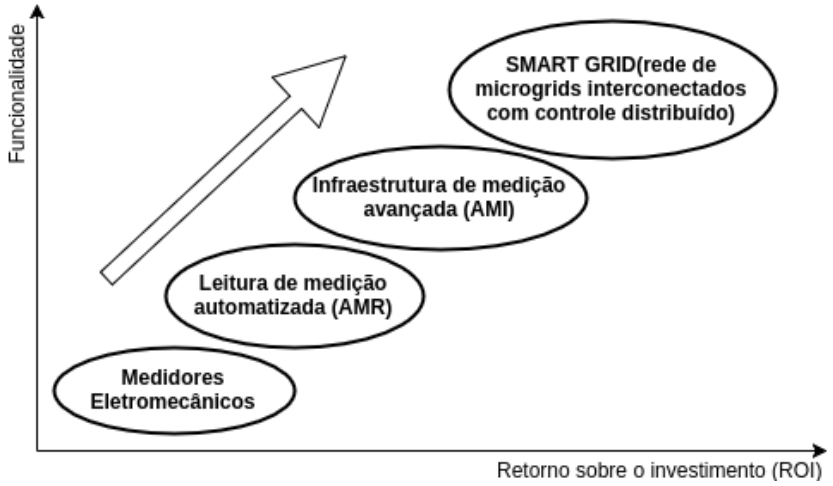


Figura 4 – Evolução do Smart *Grid*, adaptado de [11]

2.3 Fundamentos em segurança de sistemas operacionais

Segurança é uma preocupação fundamental em vários sistemas operacionais, sendo que o sistema operacional corresponde a uma camada intermediária entre as aplicações do usuário e o acesso a recursos de *hardware*, como memória e processador.

Silberschatz[14] descreve um sistema de computador seguro como um conjunto de objetos, estes podem ser o *hardware* (como acesso a memória e periféricos) ou o *software* (arquivos, programas ou semáforos). Cada objeto deve possuir um identificador único, sendo que o acesso a este dever ser feito por políticas bem definidas, autorizando operações de um processo (programa) sobre o objeto.

Alguns objetos podem executar apenas determinas ações, o processador, por exemplo, possui apenas a operação de execução, enquanto segmentos de memória podem ser lidos e escritos. Os objetos do tipo programas podem ser lidos, escritos e executados.

O sistema operacional deve garantir que os processos que são executados tenham acesso apenas aos objetos para os quais tenham a autorização correspondente para operá-los, por sua vez, cada processo deve ter acesso apenas aos objetos necessários para a sua execução. Essa restrição limita o impacto sobre o sistema caso o processo seja alvo de um ataque bem-sucedido.

A permissão para um processo executar uma operação sobre o objeto é chamada de controle de acesso. Este controle pode ser dividido em duas categorias:

- **Controle de Acesso Discricionário (DAC)**: As regras de controle de acesso ao objeto são definidas pelo proprietário do objeto.
- **Controle de Acesso Mandatório (MAC)**: As regras de controle de acesso ao objeto são definidas para o sistema, ou seja, as regras de acesso são pré-definidas e não podem ser alteradas pelo usuário.

A seguir será descrito com mais detalhes o MAC, pois o IoTMAC, proposta deste trabalho, é baseado neste modelo de controle de acesso.

2.4 Modelo Biba

Em 1977 Biba[9] propôs um modelo de políticas MAC com o objetivo de garantir a integridade dos dados no sistema, sua proposta consistiu em separar os objetos e os processos do sistema em níveis de integridade.

Na proposta de Biba os objetos não podem ser lidos por processos com nível de integridade maior que o nível do objeto. O processo pode escrever em níveis de integridade igual ou menor que o seu nível de integridade.

Na Figura 5 é exemplificado o comportamento do modelo Biba, onde o nível de integridade 1 corresponde a menor integridade do sistema. Um processo pode ler e escrever dados sobre um objeto que se encontra no mesmo nível, como é o caso do processo B e o objeto 2. O processo B pode realizar a leitura de um objeto com integridade mais alta, como é o caso do objeto 3, porém não pode ler objetos com integridade mais baixa, como é o caso do objeto 1. O único processo capaz de realizar uma leitura do objeto 1 é o processo A.

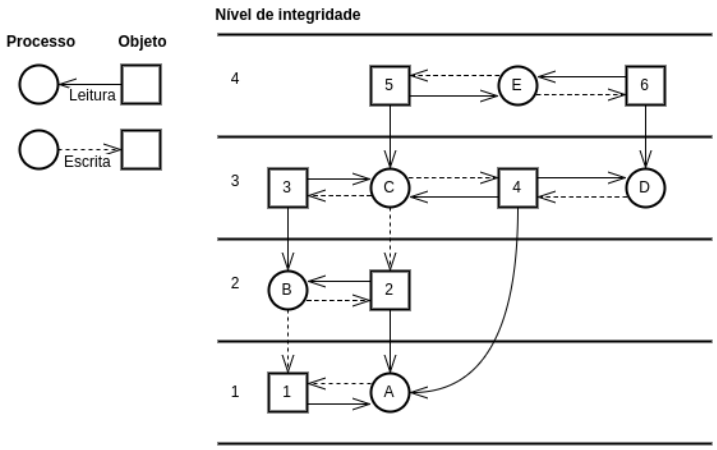


Figura 5 – Modelo multi-nível Biba, adaptado de [15]

Note na Figura 5 que o fluxo de informações do modelo proposto por Biba sempre segue o sentido dos processos mais íntegros para o menos íntegro.

2.5 Modelo LOMAC

O modelo de integridade proposto por Biba é inflexível, por isso Biba também propôs algumas flexibilizações a sua política de acesso, tendo em vista tornar o modelo mais viável em uma implementação prática.

Esse novo modelo recebeu o nome de *Low Water-Mark Mandatory Access Control* (LOMAC), essa proposta consiste em rebaixar a integridade de um objeto ou processo quando necessário.

Utilizando a Figura 5 como exemplo, caso o processo B necessite realizar a leitura do objeto 1, essa leitura só é possível rebaixando a integridade do processo B quando todos os objetos sobre o seu controle forem liberados. Porém uma vez rebaixado sua integridade o processo precisa liberar o objeto 1 e poderá voltar a seu nível de integridade original.

Também é possível rebaixar a integridade de um objeto. Rebaixando o objeto 3, o processo B poderia escrever informações sobre ele, porém uma vez rebaixado o objeto, este não recuperaria sua integridade original.

A adição de novas funcionalidades aos medidores de energia e a integração com a segurança do sistema embarcado destes dispositivos é um desafio constante. A seguir será discutido alguns trabalhos que buscam aperfeiçoar a segurança dos *Smart Meters*.

3 TRABALHOS RELACIONADOS

A seguir serão abordados os principais trabalhos relacionados os quais tratam da segurança aplicada aos *Smart Meters*.

3.1 AMIDS

McLaughlin e Holbert[16] introduziram em seu artigo o AMIDS, um sistema de detecção de fraude para *Smart Meters* que utiliza os dados de múltiplos sensores, reduzindo assim os falsos positivos que ocorrem em alarmes individuais e diminuindo o custo com inspeções físicas ao equipamento.

O AMIDS classificou as ameaças ao medidor em três categorias diferentes: *Cyber*, Físico e Efeitos sobre a medição de energia, como mostrado na Tabela 1.

Foram testados três ataques distintos: *bypass* do medidor (Ap6), desconexões periódicas (Ap3) e redução do consumo por uma constante (Ad4), como, por exemplo, o uso de ímãs para adulterar a medição do *Smart Meter*. Nos testes realizados pelos autores, apesar dos sensores não conseguirem detectar todos os ataques, o medidor conseguiu descobrir uma tentativa de ataque com 87% de confiança.

Este trabalho realizou uma pesquisa aprofundada sobre as diversas ameaças que um *Smart Meter* está sujeito e apresentou uma técnica para identificar as ocorrências de fraude.

Tabela 1 – Técnicas de ataque.

Id	Técnica
	Cyber
Ac1	Ataque através de rede remota
Ac2	Modificação de *software*/*storage*
Ac3	Roubo de credenciais para *login*
Ac4	Sobrecarga da CPU/memória
Ac5	Alteração/Intercepção da comunicação
Ac6	Sobrecarga da comunicação NAN
	Físico
Ap1	Quebra do medidor
Ap2	Inversão da medição
Ap3	Desconexão do medidor
Ap4	Extração física do *password*
Ap5	Abuso da porta óptica para obter acesso ao medidor
Ap6	*bypass* do medidor para remover a carga da medição
	Efeito sobre as medições de energia
Ad1	Parar de relatar o consumo
Ad2	Remover grandes aparelhos de medição
Ad3	Cortar a carga por um determinado período
Ad4	Alterar o perfil de carga do aparelho para esconder grandes cargas
Ad5	Reportar consumo zero
Ad6	Reportar consumo negativo (agir como um gerador)

Fonte: Adaptado de [16]

3.2 Sistema de medição de energia inteligente para detectar furto de eletricidade

Mohammad *et al.*[6] descrevem como adulterar a medição de consumo dos Smarts Meters. Neste trabalho é descrito cinco técnicas de ataques a medição do *Smart Meter*. Esses ataques têm como objetivo mascarar o consumo real de energia,

fazendo com que o medidor obtenha um consumo de energia menor.

Na Figura 6, são exibidos os tipos de furtos considerados por Mohammad e as técnicas para a sua identificação:

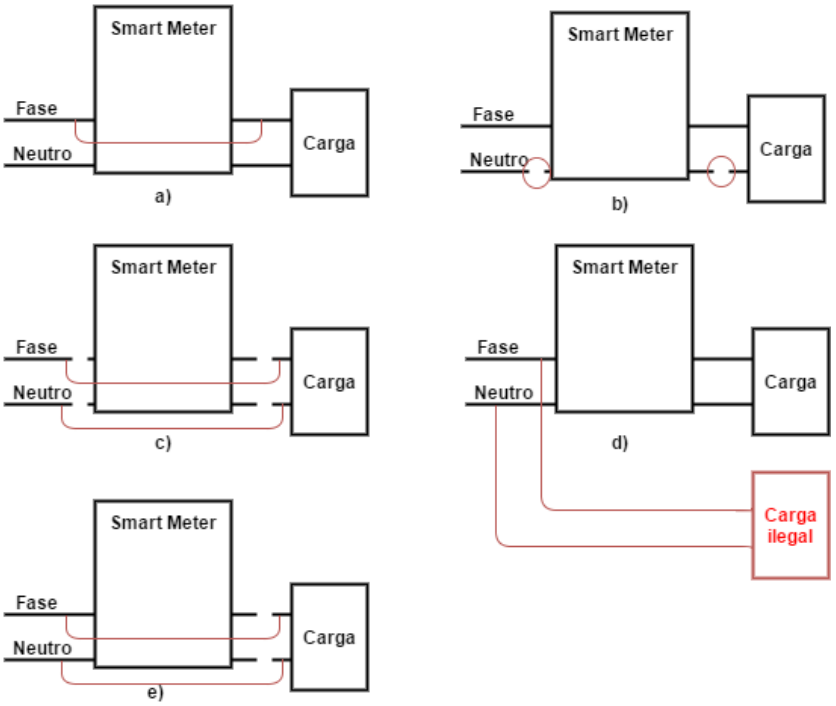


Figura 6 – Técnicas de furto de energia, adaptado de [6]

Na Figura 6 a) o furto consiste em realizar uma ligação direta entre a carga e a fase, sem passar pelo medidor, desta maneira a corrente elétrica irá percorrer o trecho de menor resistência, por 'fora' do *Smart Meter* e a medição de energia não será realizada. É possível detectar essa fraude através da corrente de retorno no Neutro, caso a corrente do Neutro seja maior que a somatória das correntes das fases significa que há uma corrente não medida pelo *Smart Meter*. Indicando falha dos sensores no *Smart Meter* ou o furto de energia.

Na Figura 6 b) o cabo referente ao Neutro do *Smart Meter* é cortado, ao fazer isso o medidor perde sua referência para medir a corrente da fase. A detecção desta técnica é feita por meio análogo ao a), ou seja, é realizado a comparação entre as correntes das fases e neutro, e caso a diferença entre as correntes de fase e

neutro sejam incompatíveis, o cabo do neutro pode estar desconectado.

Na Figura 6 c) a fase e neutro são desconectados e a ligação para a carga é realizada através de uma conexão externa, sem passar pelo *Smart Meter*. Esta técnica é detectada pela ausência de energia no *Smart Meter*. Como em operações normais é possível que ocorram quedas de energia, para a validação desta fraude é necessário que o *Smart Meter* se comunique com um servidor, informando a ausência de energia e confirmando se a ausência ocorreu por falha na *Grid* ou se representa uma tentativa de furto de energia.

Na Figura 6 d) é inserida uma carga antes do *Smart Meter*, impedindo que a corrente consumida por essa carga seja medida, pois a corrente não passará pelo *Smart Meter*. A detecção é possível através de um *Smart Meter* de observação, este *Smart Meter* tem como objetivo identificar cargas ilegais na rede. Caso a soma do consumo dos *Smart Meter* da vizinhança seja inferior ao consumo do *Smart Meter* de observação, este indica que, nesta vizinhança, foi inserido uma carga ilegal.

Na Figura 6 e) o corte do Neutro e fase ocorrem após o *Smart Meter*, nesta configuração a carga irá ser equivalente ao d), pois o medidor estará energizado, porém, haverá uma carga ilegal na rede de energia. A sua detecção é análoga a d), utilizando um medidor de observação, como explicado do parágrafo acima.

Mohammad também garante a integridade dos circuitos internos do *Smart Meter* através de um botão de pressão inserido abaixo na tampa do *Smart Meter*. Caso a tampa do medidor seja aberta, a pressão do botão diminui e este evento é detectado pelo microcontrolador do *Smart Meter* que sinaliza a abertura.

Neste trabalho foi pouco abordado a respeito de ataques cibernéticos sobre o *Smart Meter*. Para a proteção durante a atualização remota do *software* os autores utilizam a solução proposta por LeMay e Gunter[17] que impede que o *software* seja atualizado por pessoas não autorizadas.

3.3 TyTAN

Brasser *et al.*[18] desenvolveram um sistema de tempo real para nomeado TyTAN (*Tiny Trust Anchor for Tiny Devices*), este estudo tem como objetivo

fornecer um meio de comunicação seguro entre as tarefas do sistema.

A Figura 7 demonstra, de maneira simplificada, a operação do sistema, em que cada tarefa controla e monitora um sensor ou atuador. A troca de informações entre as tarefas é controlada pelo bloco IPC (*Secure inter-process communication*) implementado pelos autores.

Para cada mensagem trocada entre as tarefas o IPC identifica o emitente e o receptor, validando a troca de informações aplicando política de acesso pré-determinada. A política de acesso é armazenada em um espaço de memória seguro fornecido pela arquitetura do *hardware*

A plataforma utilizada nesta pesquisa utilizou o Intel® *Siskiyou Peak*, a arquitetura deste controlador utiliza uma unidade de proteção a memória (MPU), que controla todo acesso de escrita e leitura de memória utilizando as informações armazenadas em uma tabela confiável, região de memória do processador que não pode ser alterada [19].

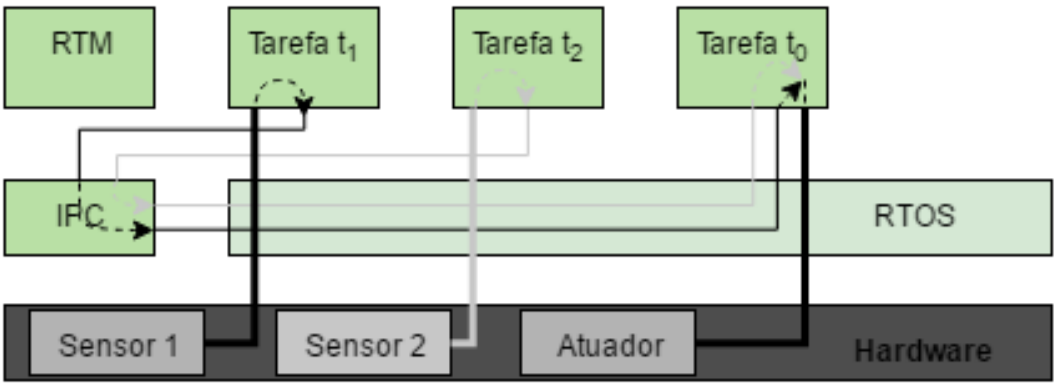


Figura 7 – Arquitetura do TyTAN, adaptado de [18]

O TyTAN é um sistema embarcado com uma segurança robusta, capaz de manter a integridade entre as tarefas. Porém sua implementação só é possível em arquiteturas de *hardware* que possuem proteção de acesso a memória.

3.4 Comentários

O AMIDS fez um levantamento de dados sobre os ataques mais comuns sofridos nos *Smart Meters* monofásicos e utiliza as informações de diversos sensores para identificar furtos de energia. Porém, algumas técnicas não foram utilizadas como, por exemplo, o sensor da tampa ser um botão de pressão, em alguns casos, o agente que deseja acessar a parte interna do *Smart Meter* não abre a tampa, utilizando outras técnicas, uma furadeira por exemplo. Também há técnicas de elevação de temperatura com o objetivo de queimar os componentes internos, item este que não foi discutido.

O AMIDS também propõe utilizar a técnica *Cumulative Attestation Kernel* (CAK), solução proposta por LeMay e Gunter[17] visando evitar que o *firmware* seja atualizado por pessoas não autorizadas. Esta proposta não oferece outros recursos de segurança, como, por exemplo, evitar ataques lógicos ao sistema durante a comunicação convencional, contramedidas caso um agente malicioso consiga acesso físico ao *hardware* ou um ataque a lógica do programa.

Mohammad aponta quais são as principais técnicas para furto de energia, como detectá-las e possíveis contramedidas, como o uso de harmônicas da rede (sinal parasita) que danifique o equipamento que está consumindo energia irregularmente. Porém neste trabalho não é discutido os ataques lógicos ao medidor.

O sistema TyTAN proposto por Brasser *et al.* aparenta ser uma solução robusta para proteger as mensagens trocadas entre as tarefas de um sistema operacional. Neste modo, mesmo que uma tarefa sofra um ataque, a tarefa não irá corromper outras, as quais não possui acesso. Porém essa técnica é baseada em um *hardware* que fornece proteção de acesso a memória, sem essa proteção o agente malicioso é capaz de acessar as políticas de acesso e alterá-las. Assim, sistemas embarcados com um *hardware* mais simples não se beneficiam com esta proposta.

4 PROPOSTA

A proposta deste trabalho é o desenvolvimento de um *Smart Meter* utilizando um sistema de tempo real (RTOS) preemptivo, capaz de responder a ataques físicos (*tampering)* e lógicos ao equipamento. A proteção aos ataques lógicos será garantida por um mecanismo de integridade multi-nível, o IoTMAC, que irá fornecer maior segurança ao acesso à memória. Segurança similar aos componentes em nível de *hardware* chamado de modo seguro.

4.1 RTOS para o 8051

As arquiteturas ARM, por exemplo, possuem em sua estrutura registradores e *bits* de proteção em nível de *hardware*, que promovem o isolamento de memória entre os processos em execução. O trabalho desenvolvido por Brasser *et al.*[18], utiliza este recurso de *hardware* para garantir uma comunicação segura inter-tarefas em um RTOS. Nossa proposta busca proteger espaços de memória crítica em uma arquitetura que não oferece tal proteção por *hardware*.

O RTOS foi escolhido com base nos objetivos propostos pelo trabalho, a proteção aos espaços críticos de memória e a portabilidade para arquitetura 8051, que não possuem proteção de memória via *hardware*.

A portabilidade (*port*) do código-fonte de um *software* para uma determinada arquitetura de *hardware* depende da dependência do *software* em relação ao uso de registradores e recursos da arquitetura alvo. Assim, a portabilidade do RTOS foi feita manualmente, configurando a utilização de registradores e interrupções específicas do 80515.

A escolha do RTOS foi feita após analisar vários sistemas disponíveis no mercado, suas características, licenças de uso e a portabilidade para a arquitetura 8051. Essa análise pode ser vista na Tabela 2.

O RTOS escolhido para a implementação do IoTMAC foi o FreeRTOS,

Tabela 2 – Comparação entre os RTOS do mercado.

RTOS	Compilador	Licença	Acesso ao Fonte	Preemptivo
Abassi	Keil	Proprietário	Fechado	Não
FreeRTOS	SDCC	GPL	Aberto	Sim
RTX51*	Keil	Proprietário	Aberto	Sim
RTX51 Tinny	Keil	Proprietário	Aberto	Não
Salvo	Keil	Proprietário	Aberto	Não
Euros	EUROScope	Proprietário	Fechado	Não
QP	Qtools/Keil/IAR	Dual	Aberto	Sim

Nota: * RTOS descontinuado.

Nota: Licença dual representa os dois formatos de licença. GPL (para uso não comercial) e Proprietário (para uso comercial).

Fonte: O Autor

sistema operacional *open-source*, distribuído sobre a licença GPL[20] e desenvolvido e mantido pela *Real Time Engineers Ltd*[21]. O FreeRTOS possui porte para diversas plataformas (ARM, Atmel AVR, AVR32, MSP430, PIC, etc.)[20], logo a implementação deste sistema também permite sua utilização destas arquiteturas.

O FreeRTOS se mostrou o mais viável para a implementação, pois oferece preempção entre as tarefas e possui um código portado para o microcontrolador 8051F120 da Silabs [22], que é mais próximo da arquitetura utilizada nesta proposta. O 8051F120 é uma arquitetura expandida do padrão de 8051 proposto pela Intel.

O Intel 8051 é um microcontrolador de 8 *bits* com dois contadores/*timers* internos e uma interface de comunicação serial[23]. Entre as expansões presentes no microcontrolador 8051F120 está a utilização de um terceiro *timer*, característica presente apenas no 8052 da Intel. Também existem alguns periféricos adicionais embutidos (*e.g.* conversores AD) e a utilização de paginação SFR para expandir o número de periféricos controlados pelo processador [22].

Considerando os motivos citados acima, não foi possível utilizar o porte já existente, porém o porte do 8051F120 foi a base para gerar o porte do FreeRTOS

do SoC nesta proposta.

O primeiro esforço desta pesquisa foi a realização na portabilidade a partir do código disponível para a arquitetura 8051. Este esforço consistiu na alteração dos arquivos relacionados com a arquitetura de *hardware*, endereços de registradores, nomenclatura e diferenças existentes entre o 8051 e 8051F120. As implementações e alterações realizadas foram:

- Criação dos arquivos com o mapeamento dos endereços de memória;
- Alteração do uso do *timer* 2 (usado do 8051F120) para utilizar o *timer* 1.
- Montagem do *script* de compilação utilizando o *Makefile*[24].

O porte foi gerado utilizando o SDCC[25] v.2.8.0, sendo que a versão mais (v.3.6.0) lançada em Junho de 2016, não se mostrou compatível para a implementação do FreeRTOS. A incompatibilidade ocorreu devido a alterações da definição de declarações de algumas variáveis na nova versão. Também há relatos que em versões mais recentes do SDCC, após corrigidas as declarações de variáveis, o sistema apresentou uma queda de desempenho[26], apresentando problemas com a biblioteca de filas ou problemas durante os testes de estresse.

Os testes de implementação foram realizados com o simulador ucSim[27], disponibilizado junto com o compilador. O simulador foi configurado respeitando os limites de memória (5 kB) presentes no SoC escolhido para a implementação do sistema.A limitação de memória do microcontrolador restringiu a quantidade de tarefas em execução durante os testes.

O ucSim é um programa que carrega o arquivo hexadecimal gerado por um compilador e simula as ações do processador 8051, de acordo com as instruções de máquina do arquivo carregado.

Durante a execução da simulação é possível alterar registradores e espaços de memória, desta maneira é possível simular eventos de interrupção. Por exemplo, durante a execução foi alterado o registrador que corresponde a *flag* de interrupção externa (IE0), após a alteração manual deste registrador o simulador irá tratar o evento correspondente a esta ação.

Também é possível configurar paradas de execução da simulação, quando ocorrer a alteração de um endereço de memória. Por exemplo, é configurado para a simulação pausar do momento em que ocorrer a escrita do registrador SFR 0xFF (que não é utilizado no 8051), neste momento a simulação é pausada, sendo exibida um resumo do estado atual do processador, como mostrado na Figura 8

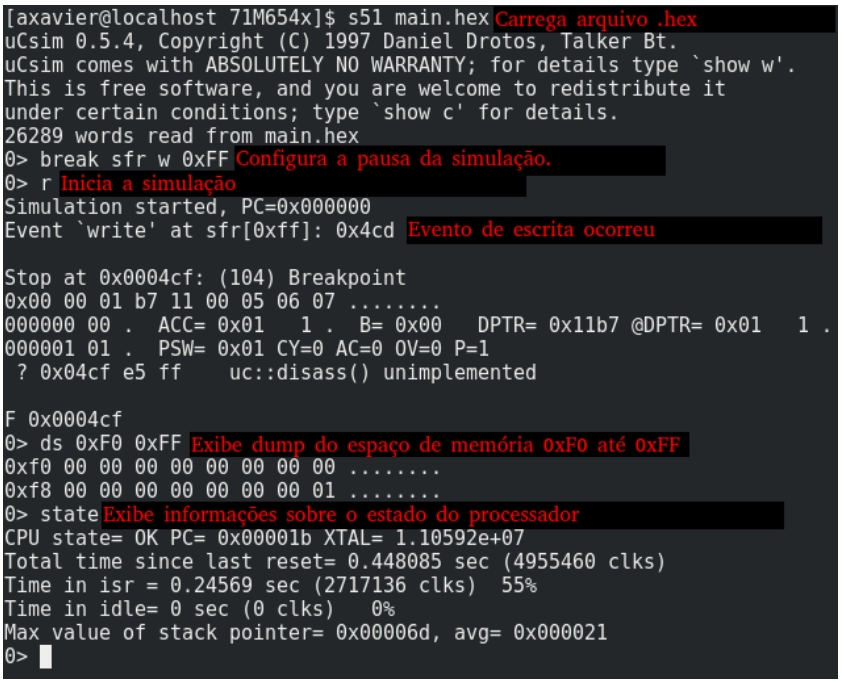


Figura 8 – Simulação realizada pelo uCSim. Fonte: O Autor

Os testes realizados para a portabilidade do sistema consistiram em executar múltiplos processos independentes, com dois níveis de prioridades. Na simulação foram executados 10 processos (5 com baixa prioridade e 5 com alta prioridade), todos os processos iniciaram e concluíram múltiplas vezes.

O porte também foi testado na placa de prototipação 71M6543[28], neste teste foi utilizando uma implementação com dois processos independentes, cada processo tinha a responsabilidade de acender e apagar um led da placa em um intervalo regular de tempo.Este teste é conhecido como *Blink* e tem como objetivo verificar se a compilação e execução do sistema está funcionando, nas sessões seguintes serão descritos testes mais aprofundados para a validação do IoTMAC.

4.2 IoTMAC

No mercado de circuitos integrados há várias soluções SoC para a medição de energia elétrica e AMI (*Advanced Metering Infrastructure*). Existem soluções simples como o 71M6543 desenvolvido pela Silergy[29] o qual possui um núcleo de processamento da família 8051, até soluções mais robustas como o ATSAM4C32[30], desenvolvido pela Atmel, possuindo um núcleo de processamento *dual-core* ARM.

A diferença de preço entre essas duas soluções é de aproximadamente 125%. Como as fabricantes de Smarts Meters escolhem a melhor relação custo/benefícios, o SoC escolhido acaba sendo uma arquitetura mais simples, porém com um alto conjunto de soluções integradas. [1]

Por isso foi escolhido o desenvolvimento do trabalho proposto sobre o SoC 71M6543 da Silergy, que possui uma arquitetura padrão *de facto* e é amplamente utilizado no mercado de medidores de energia elétrica.

O SoC 71M6543 possui um núcleo de cálculo de 32 *bits*, esse núcleo realiza todas as operações métricas de energia, fornecendo informações sobre o consumo de energia dos quatro quadrantes, análises de qualidade de energia como registros de elevação ("*swell*") ou queda de tensão ("*sag*"). Este núcleo é chamado de *Compute Engine* (CE) alcançando uma precisão de 0,1% em uma espaço de 2000 amostras. O núcleo de 8 *bits* 80515, supera 5 MIPS representando uma melhora no desempenho médio de oito vezes (em termos de MIPS) se comparado ao *Intel* 8051 funcionando com a mesma frequência de *clock* [32].

Mesmo com um núcleo exclusivo para as funções de metrologia e um núcleo de processamento robusto, o 71M6543 possui uma debilidade em relação a segurança no gerenciamento de memória. Sua única proteção consiste em impedir a leitura ou escrita da memória *flash* para operações externas, via SPI ou ICE (*in-circuit emulator*, porta para emulação do SoC). Esse esquema de segurança impossibilita

[1] Em Abril de 2010 a empresa *Maxim Integrated* anunciou a compra da empresa *Teridian Semiconductor Corporation* por aproximadamente US$315 milhões. Na nota de empresa divulgada, a *Maxim* relata alguns motivos que motivaram a compra: a *Teridian* possuía 50% do mercado de SoC para medição de energia, sendo uma importante fornecedora para 3 das 4 principais fabricantes de medidores nos Estados Unidos e para mais de 50 fabricantes em nível mundial[31]

que um usuário tenha acesso ao código fonte ou realize testes de depuração, porém não impede que um programa que já esteja executando na memória do *chip* invada espaços críticos de memória.

O IoTMAC tem o objetivo de garantir a integridade lógica dos dados do sistema em eventuais ataques a partir de conexões de rede. Os ataques lógicos correspondem a tentativa de corromper o estado do dispositivo, tentando injetar um código malicioso no sistema ou provocando o funcionamento de maneira não esperada. Estes ataques visam acessar e modificar espaços de memória de maneira ilegal.

Um exemplo simples é uma função de passagem de um único parâmetro, que consiste em uma sequência de caracteres finalizados com o caractere nulo, que será armazenada em um espaço da RAM. Caso o usuário esqueça o caractere nulo ao final da sequência, outras funções que dependem deste caractere para detectar o fim da *string* continuarão registrando dados sequenciais de memória, invadindo e alterando variáveis de outros processos.

Essa invasão de memória pode provocar instabilidade no sistema, levando a erros ou, quando essa alteração busca alterar espaços de memórias conhecidos, uma fraude na medição do *Smart Meter*.

A proteção do IoTMAC consiste no mapeamento da memória considerada crítica para o sistema. Essa deve ser acessada apenas pelas tarefas internas ao medidor, isolando os processos de metrologia dos processos que coletam informações. A coleta de dados é executada através de sensores e da comunicação de rede, os quais, para a proposta, são consideradas de menor integridade.

A Figura 9 mostra uma exemplificação de como o microcontrolador (MPU) do 71M6543 se comunica com o CE (*Compute Engine*) através do compartilhamento da memória RAM. O CE fornece uma interface *front-end* para a medição de energia, o bloco CE é equipado com ADC (conversor analógico-digital) e processador aritmético. As configurações do CE, algoritmo do processador aritmético e configuração do ADC, ficam salvas na RAM e são carregados durante a inicialização do sistema e podem ser alteradas pelo MPU a qualquer momento (Figura 9, evento I). Por isso sua proteção é crucial para garantir a integridade na medição de energia.

Após carregado as configurações, o CE está apto a coletar as amostras para a metrologia de energia elétrica (tensão, corrente e temperatura - Figura 9, evento II). Estas amostras são processadas e ficam disponíveis para a MPU, Figura 9, evento III, estes dados podem ser analisados para detectar possíveis técnicas de *tamper* e salvar informações detalhadas de consumos para a concessionária elétrica, Figura 9, evento IV.

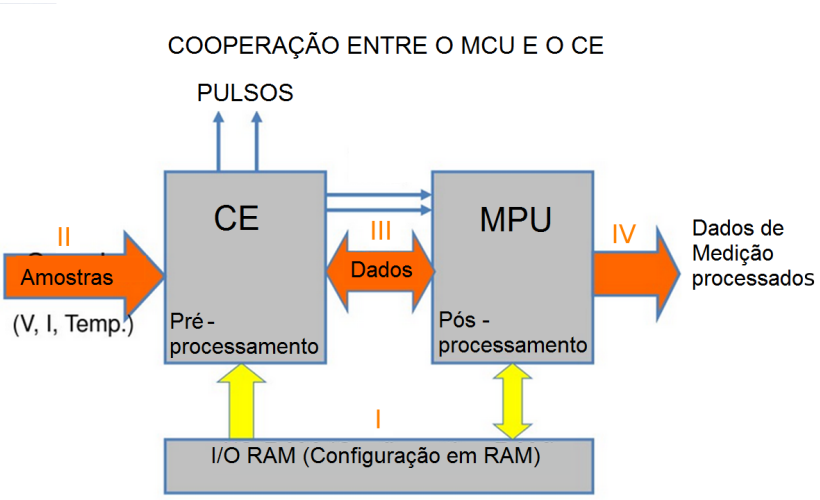


Figura 9 – Cooperação entre o CE e Microcontrolador, adaptado de [32]

Neste trabalho foi considerado como a função primária do *Smart Meter* a execução da metrologia, devido a importância para o faturamento da empresa de energia elétrica. Por esse motivo, o objetivo do IoTMAC é manter a sua integridade, garantindo o seu funcionamento.

Para a proteção adequada da memória sem o suporte de *hardware* é necessário mapear os endereços físicos de memória considerados críticos ao sistema. Os endereços críticos foram considerados como sendo aqueles que alteram a precisão e o comportamento da metrologia do medidor. Ou seja, as alterações dos dados nestes endereços afetariam a medição, *i.e.*, modo como o SoC realiza os registros referente ao consumo de energia.

A Figura 10 mostra como a memória RAM do 71M6543 é dividida, cada região armazena um código com uma funcionalidade específica do sistema.

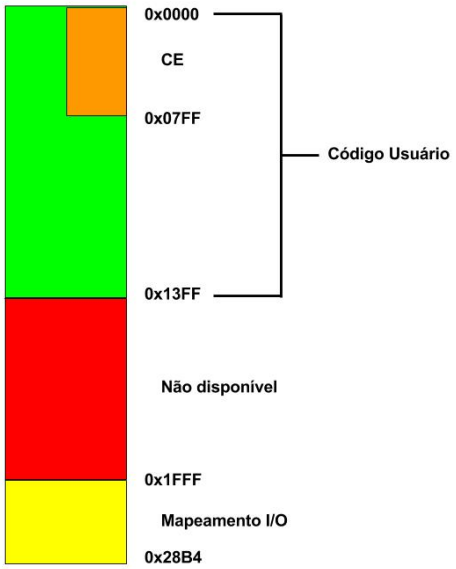


Figura 10 – Mapeamento da memória RAM do MCU. Fonte: O Autor

A primeira área, na cor verde, descrita como código usuário, compreende a região entre 0x0000 e 0x13FF, correspondendo a área que armazena as variáveis locais do sistema em tempo de execução. Esta área é compartilhada com o CE do SoC, ou seja, entre os endereços 0x0000 e 0x07FF são armazenados os valores referentes as variáveis de metrologia que são utilizadas pelo CE e podem ser utilizadas pelo usuário (Figura 9). Está é uma área de grande importância e que deve ser mantida integra. No espaço de memória entre os endereços 0x0800 e 0x13FF são armazenadas as variáveis e programas do usuário, ou seja, tarefas que são executados pelo sistema operacional. Essa área, por não compreender o funcionamento da metrologia do sistema, foi considerado não crítica.

A segunda área, destacada em vermelho, corresponde ao endereçamento entre 0x14FF e 0x1FFF, esta é uma área que não está disponível para o sistema. Qualquer alteração deste endereço de memória não altera o sistema ou o *hardware*. Esta área foi considerada não crítica.

A terceira área, em amarelo no endereçamento de memória, entre 0x2000 e 0x28B4, realiza a interface de comunicação com os periféricos internos no SoC, a área é chamada de Mapeamento I/O. Esta área controla o multiplexador, responsável

por alternar a coleta entre os diversos sensores de tensão e corrente, o relógio de tempo real (RTC), portas de comunicação, etc.. Por estes motivo esta área também foi considerada um espaço crítica de memória que deve ser protegido.

Após realizar o mapeamento dos espaços de memória, estas informações devem ser armazenadas em um local seguro, onde não é não é permitida a sua sua alteração e apenas a leitura seja concedida. O 71M6543 possui proteção para leitura externa, porém não possui para as aplicações internas.

Apesar do *Smart Meter* proposto possuir proteção física (*i.e.*, sensores de vibração e de abertura os quais impedem que um agente malicioso tenha acesso físico ao sistema; seção 4.4) e o SoC possuir segurança de gravação, durante a inicialização do sistema é feita uma verificação de integridade do código. A verificação é realizada validando o *Checksum* do código presente da memória *flash*.

O *Checksum* é gerado pelo gravador da Teridian durante a carga do programa para o SoC e armazenado na memória de programa. Ainda há a possibilidade de ocorrerem colisões entre o *Checksum* de um código válido e de um código malicioso. Porém, está ocorrência é minimizada de acordo com o algoritmo utilizado. Essa discussão não é abordada neste trabalho e pode ser vista em [33].

Para microcontroladores que não possuem a mesma solução proposta pela Teridian será necessário a geração do *Checksum* ou *Hash* criptográfico antes da gravação e seu valor deve ser armazenado junto com o código binário. Com esta abordagem é possível realizar uma avaliação da integridade do código do sistema durante a sua inicialização.

Após garantir que o algoritmo em execução está integro, o FreeRTOS foi alterado para a implementação do IoTMAC.

No FreeRTOS as informações referentes as tarefas são controladas por uma estrutura chamada TCB (*Task Control Block*). As informações armazenadas no TCB são:

- Ponteiro para o último item da pilha de memória do evento;
- Lista de tarefas em estado bloqueado e prontas para serem inicializadas;

- Fila de eventos (mensagens que podem ser repassadas de uma tarefa para outra);
- Prioridade da tarefa;
- Ponteiro para o início da pilha de memória da tarefa;
- Nome da tarefa.

O TCB também armazena informações adicionais de acordo com a configuração do FreeRTOS. Essas informações são:

- Rastreamento: número sequencial criado durante a criação da tarefa, utilizado para propósitos de depuração;
- Uso de mutex: habilita a capacidade do uso de mutex e semáforos do sistema;
- Etiqueta da tarefa: configuração para executar tarefas como "*Hook*", tarefa que é executada quando todas as demais estão em espera.

A informação de integridade da tarefa foi adicionada como um dado adicional ao sistema, ou seja, a implementação do IoTMAC não afetará o comportamento do FreeRTOS e só ficará disponível caso a configuração esteja parametrizada para tal.

Foram utilizadas instruções de pré-compilação no código-fonte. Estas instruções orientam o compilador a adicionar ou não trechos de código a partir de variáveis presentes nos arquivos de configuração do FreeRTOS.

Na Figura 11 é exibido um trecho do arquivo de configuração do RTOS, são exibidos algumas variáveis de configuração e a variável introduzida com o objetivo de habilitar ou não o sistema IoTMAC.

Entre as configurações listadas na figura abaixo, suas funcionalidades são:

- **INCLUDE_vTaskPrioritySet**: Alterar a prioridade de uma tarefa;
- **INCLUDE_uxTaskPriorityGet**: Recuperar a prioridade de uma tarefa;

- **INCLUDE_vTaskDelete**: Apagar uma tarefa alocada em memória, liberando espaço;
- **INCLUDE_vTaskCleanUpResources**: Funcionalidade descontinuada, apesar de existir no arquivo de configuração a função atrelada a essa variável não existe no FreeRTOS;
- **INCLUDE_vTaskSuspend**: Altera o status de uma tarefa de *Ready*(pronta para executar) para *Suspend*(não será executada);
- **INCLUDE_vTaskDelayUntil**: Sincronização de tarefas, permite que uma tarefa aguarde o fim de outra;
- **INCLUDE_vTaskDelay**: Congela a execução de uma tarefa por uma constante de tempo, geralmente em segundos ou ciclos de *clock*.
- **INCLUDE_xResumeFromISR**: Permite que uma tarefa seja iniciada ao final de uma rotina do serviço de interrupção (ISR);
- **INCLUDE_xProctetionBIBA**: Variável introduzida para habilitar a proteção de leitura e escrita em memória via *software*.

```
/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet      0
#define INCLUDE_uxTaskPriorityGet     0
#define INCLUDE_vTaskDelete           0
#define INCLUDE_vTaskCleanUpResources 0
#define INCLUDE_vTaskSuspend          1
#define INCLUDE_vTaskDelayUntil       1
#define INCLUDE_vTaskDelay            1

#define INCLUDE_xResumeFromISR        1

#define INCLUDE_xProtectionBIBA       1
```

Figura 11 – Configurando a proteção IoTMAC para o FreeRTOS. Fonte: O Autor

Na Figura 12 é exibido um trecho do código utilizando as diretivas de pré-compilação do SDCC. Estas diretivas têm como objetivo incluir, ou não, o código referente a implementação do IoTMAC de acordo com a configuração da variável

INCLUDE_xProtectionBIBA indicada na Figura 11. Nesta figura também é representado a implementação da função responsável por obter o conteúdo de um espaço de memória de maneira segura e será detalhada adiante.

```
#if ( INCLUDE_xProtectionBIBA == 1 )
void* getMemory(portCHAR *taskIntegrity, unsigned portSHORT address, portSHORT sizeBuffer)
{
  static portSHORT sSize = sizeBuffer;
  __xdata static portSTACK_TYPE * __data pReturn;
  __xdata static portSTACK_TYPE * __data target;

  target = address;
  if ( *taskIntegrity == secHiIntegrity )
  {
    if ( (address >= startMemoryLow ) && ( address < (startMemoryLow + lengthMemoryLow) ) ){
      //return errMemoryLowIntegrity;
      return NULL;

    }else if( (address + sizeBuffer) >= startMemoryLow ){
      //return errMemoryOverflow;
      return NULL;
    }
  }
  //Get memory space to copy requested content
  portENTER_CRITICAL();
  {
    pReturn = pvPortMalloc( (size_t)sizeBuffer );
  }
  portEXIT_CRITICAL();

  //Copy requested content to new memory space
  if ( pReturn != NULL )
  {
    do{
      *pReturn = *target;
      pReturn++;
      target++;
      sSize--;
    }while(sSize > 0);
    pReturn -= sizeBuffer;
    return (void*)pReturn;

  }else
  {
    //return errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;
    return NULL;
  }
}
```

Figura 12 – Uso de diretivas de pré-compilação. Fonte: O Autor

A implementação do IoTMAC buscou garantir o desacoplamento da solução, ou seja, ser uma opção de segurança adicional ao RTOS, deixando a critério do usuário o uso desta funcionalidade. Está abordagem permitirá a inclusão desta solução do repositório oficial no FreeRTOS futuramente, caso a comunidade queira adotar esta solução.

A configuração da primitiva **INCLUDE_xProtectionBIBA** permitirá a utilização de funções que garantem a proteção de acesso ao endereço de memória.

As funções são:

- xSecureTaskCreate;
- xGetMemory;
- vSetMemory.

A função **xSecureTaskCreate** possui a mesma funcionalidade da função **xTaskCreate** padrão ao sistema, que é de criar a tarefa solicitada, alocar a memória necessária para sua execução e adicionar a tarefa a uma fila de tarefas prontas para serem executadas. Além disso a **xSecureTaskCreate** possui um parâmetro a mais para informar a integridade relacionada com esta tarefa, este parâmetro é o ponteiro para um espaço de memória *flash* protegida.

A implementação da tarefa segura deverá ser realizada sem o uso de acesso direto à memória, que é permitido pela linguagem C, porém utilizando apenas as funções **xGetMemory** e **vSetMemory**. Estas funções confirmam a integridade do processo e da tarefa durante a leitura ou escrita em um endereço da memória XRAM.

O algoritmo 1 mostra o pseudocódigo referente a implementação da função **xGetMemory**. Ao solicitar a leitura de uma região de memória o usuário obtém um endereço de memória com a cópia do conteúdo do espaço informado, caso a tarefa possua permissão de leitura. Nesta maneira a tarefa não obtém acesso direto à região protegida, impedindo uma eventual escrita nesta região.

Como a alocação de memória é dinâmica o espaço de memória também é protegido pelo princípio de obscuridade, ou seja, uma vez que o usuário não consegue prever o endereço retornado pela função, se torna mais complexo estruturar um ataque de *buffer overflow* [34].

Caso a tarefa não possua o devido nível de integridade para acessar o conteúdo é retornado um valor nulo, indicando que ocorreu problema ao acessar a memória.

O algoritmo também pode retornar o valor nulo caso não consiga alocar um espaço de memória para realizar a cópia do endereço solicitado, neste caso o acesso

Algoritmo 1 Recupera dados da memória XRAM

```
1:  função xGetMemory(memoryAddress, memorySize)
2:      se tarefa atual é de alta integridade então          ▷ tarefa é o subject
3:          se memoryAddress é de baixa integridade então
4:              devolve Erro - Acesso Negado
5:
6:          senão se memorySize provoca vazamento de memória então
7:              devolve Erro - Acesso Negado
8:          fim se
9:      fim se
10:
11:     se memória não está cheia então
12:         Copia conteúdo de memoryAddress para newBuffer
13:         devolve newBuffer
14:     senão
15:         devolve Erro - Não foi possível alocar memória
16:     fim se
17: fim função
```

a memória foi autorizado, porém o sistema não consegue copiar a informação para o usuário.

Para a melhor compreensão, o algoritmo 1 pode ser comparado com sua implementação real representada da Figura 12. A implementação da função **getMemory** possui um parâmetro adicional, o *taskIntegrity*, este parâmetro não é informado pelo usuário, ele é informado pelo SO que sabe qual é a integridade da tarefa em execução, sendo que a integridade da tarefa foi informada anteriormente pela função **xSecureTaskCreate**.

A implementação real é possível notar as variáveis: *startMemoryLow*, *lengthMemoryLow*; essas variáveis são constantes e são definidas pelo usuário. Também é possível notar que a implementação real possui mais linhas que seu pseudocódigo, isto ocorre pois a cópia de memória foi feita manualmente e foram feitas validações adicionais, para evitar problemas de concorrência com outras tarefas em paralelo, por exemplo.

O algoritmo 2 representa o pseudocódigo referente a função **vSetMemory**. Essa função é responsável por permitir a alteração de uma região de memória

apenas se o processo possuir a integridade correta para esta operação.

Algoritmo 2 Altera os dados de um espaço de memória

```
1: função vSetMemory(memoryAddress, buffer, bufferSize)
2:     se tarefa atual é de baixa integridade então          ▷ a tarefa é o subject
3:         se memoryAddress é de alta integridade então  ▷ memoryAddress é o object
4:             devolve Erro - Acesso negado
5:         senão se memoryAddress + bufferSize provoca estouro de memória então
6:             devolve Erro - Estouro de memória
7:         fim se
8:     fim se
9:     Escreve conteúdo do buffer para memoryAddress
10:    devolve Sucesso
11: fim função
```

Em caso de suspeitas de tentativa de fraudar o *Smart Meter* o sistema diminui a integridade de todas as tarefas. A suspeita ocorre quando o sistema detecta uma tentativa de acesso não autorizado a memória ou uma violação física ao sistema.

Tal medida é importante para garantir a integridade dos dados, uma vez que, caso o sistema seja fraudado (abertura de tampa, por exemplo) todo o sistema pode ser considerado comprometido e não é possível confiar nos componentes internos, que podem ter sofrido substituição ou reprogramados.

A queda de integridade das tarefas é realizada alterando o conteúdo da variável que armazena o nível de integridade alta para baixo, assim todas as tarefas que utilizem essa variável para armazenar sua integridade serão afetadas.

Complementado a segurança referente a integridade dos dados, também é registrado *log* de eventos. Os registros destas ocorrências armazenam informações de depuração (horário que ocorreu o evento e um *dump* de memória) que podem auxiliar em auditorias posteriores. Os registros de *log* são salvos em uma memória externa segura (*Smart Card*) ou enviadas a um servidor remoto para monitoramento.

A segurança do registro de *log* é fornecido pela arquitetura do *Smart Card*, que permite a leitura e escrita de dados apenas mediante a um usuário e senha ou

outros tipos de proteção dependendo do *Smart Meter* escolhido.

As ocorrências registradas são:

- Evidência do furto de energia;
- Evidência de fraude do sistema;

Para as ocorrências de furto é registrado o horário do evento e uma descrição dos sensores ou medições que levaram ao registro do *log*.

Para as ocorrências de fraude do *Smart Meter*, além da hora do evento, também é salvo um dump da memória. O *dump* de memória poderá ser auditado posteriormente e confirmado se houve ou não alteração das regiões consideradas críticas. Ao analisar estes registros é possível verificar se o conteúdo da memória foi alterado, por exemplo o espaço onde é armazenado o código no CE (Figura 10, espaço alaranjado) deve permanecer inalterado durante toda a execução do *Smart Meter*, caso seja encontrado alguma divergência significa que o sistema foi fraudado. E utilizando a hora registada no *log* é possível confirmar que após este momento as informações referentes a metrologia não são mais confiáveis.

O diagrama representado na Figura 13 permite visualizar de uma maneira mais ampla o funcionamento do sistema, desde a sua inicialização até a preempção entre as tarefas.

Na Figura 13 são destacadas sete atividades. Ao iniciar o sistema é verificado a integridade através de um *Checksum* (atividade I), caso o sistema encontre uma irregularidade é registrado um *log* do evento e a integridade de todas as tarefas é rebaixada. Desta maneira após a ocorrência de uma possível adulteração o sistema não permite alterações dos espaços de memória considerados críticos (Figura 10).

Após a perda de privilégios e o registro da ocorrência, caso a memória *flash* não esteja integra, o sistema continua sua operação normalmente, mas possivelmente causará pouco ou nenhum dano, dado que as tarefas que executam ações nas áreas críticas não terão integridade suficiente para isto.

A atividade II representa a calibração dos sensores presentes no *Smart Meter* (*i.e.*, temperatura e acelerômetros). Esta atividade é importante ser destacada, pois

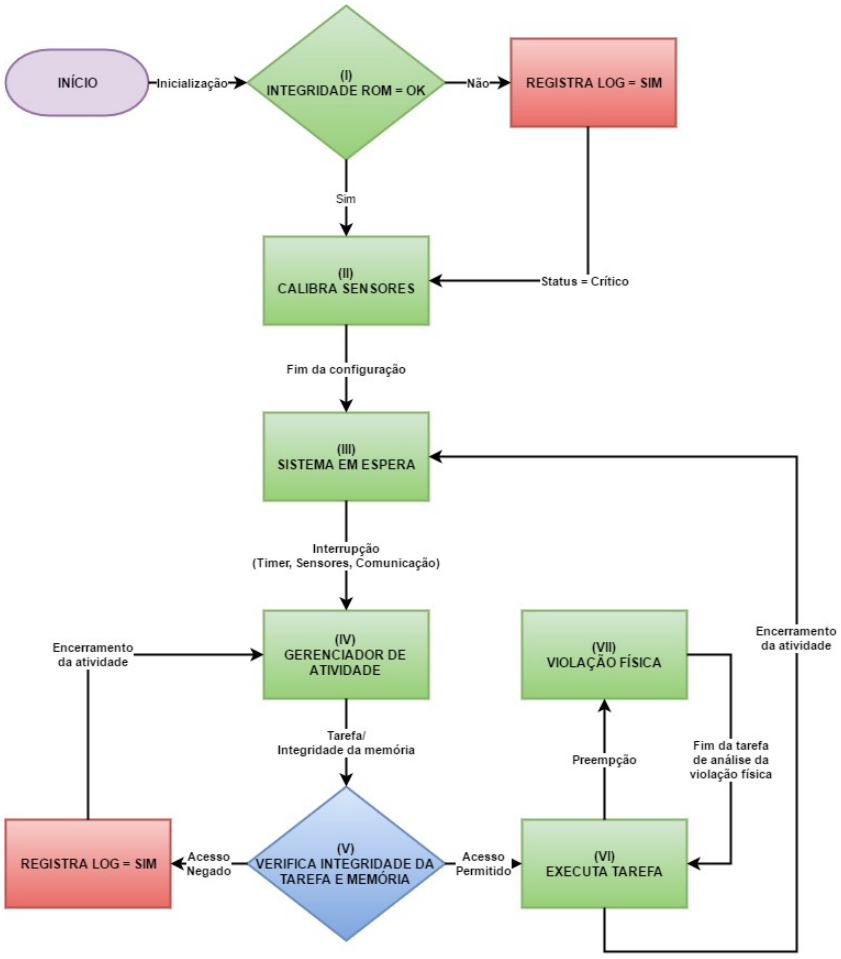


Figura 13 – Fluxo geral do FreeRTOS com o IoTMAC. Fonte: O Autor

esses sensores são utilizados para detectar a tentativa de fraude do *Smart Meter*. Caso os sensores não sejam calibrados o sistema deixa de detectar as tentativas de fraude.

Após finalizada a configuração inicial, atividades I e II, o FreeRTOS carrega todas as atividades do sistema em memória e aguarda a inicialização destas tarefas através de algum evento. A atividade III representa um evento externo, provocado por uma interrupção proveniente de porta serial ou sensor, ou interno, provocado pelo *tick* do sistema, interrupção do RTC ou CE.

Quando ocorre algum evento, um processo controlado pelo gerenciador

de atividades do FreeRTOS (atividade IV) é iniciado, de acordo com o TCB anteriormente citado. A atividade ao tentar interagir com a memória utilizando as funções seguras propostas por este trabalho pode ter sua interação com a memória permitida ou negada, atividade V.

Caso o acesso a memória seja negado é gerado o *log* de tentativa de fraude e a atividade é encerrada. Deste modo o sistema retorna para o modo de espera.

Caso o acesso seja autorizado a atividade é executada normalmente (Atividade VI) porém durante a execução da atividade pode ocorrer a preempção para uma tarefa de maior prioridade. Esta preempção é indicada pela atividade VII e após finalizado o processo de maior prioridade é retornado para a atividade VI. Após o encerramento desta atividade o sistema espera por uma nova interrupção para a execução de nova tarefa, atividade III.

É importante salientar que a preempção entre as atividades podem ocorrer, porém não é um evento obrigatório, considerando que eventos de maior prioridade ocorrem com menor frequência, a preempção das atividades pode ser considerada uma exceção.

A Figura 13 foi criada com o intuito de demonstrar de uma maneira simples uma visão geral do sistema proposto. Esta não deve ser entendido como um diagrama de estados, por exemplo, na figura a atividade VI é suspensa por causa de uma interrupção, porém nem sempre a tarefa em execução é interrompida e o fluxo do sistema pode ser V -> VI -> III (Figura 13).

Nas seções seguintes serão descritas com mais detalhes as atividades relacionadas com a detecção do furto de energia e tentativa de fraudar o *Smart Meter*.

Apesar de não ser o escopo principal deste trabalho, a integração destas duas soluções colabora com o desenvolvimento do *Smart Meter* seguro.

4.3 Detecção de furto de energia

Caso o *Smart Meter* consiga medir a carga consumida pelo usuário, porém tenha sua metrologia adulterada por uma técnica de *tamper* é considerado uma

fraude.

O não registro é provocado por alteração criminosa em algum componente da rede elétrica. Foram identificadas 6 possíveis modificações na instalação de um medidor que consideramos furtos de energia, as variações identificadas estão descritas na Tabela 3 e podem, para melhor identificação, ser comparadas com a Figura 6.

Tabela 3 – Tipos de furto de energia.

Adulteração	Descrição	Referência a Figura 6
R01) Inversão de quadrantes	Tentativa de usar o cabo de energia proveniente da rede para simular a medição de energia de cogeração.	
R02) Derivação da fase	"Dar a volta"em um, dois ou três elementos de fase, mantendo o neutro ligado à rede.	a)
R03) Desconexão da linha de neutro	Desconectar a referência ao neutro para desestabilizar o sistema de medição.	b)
R04) Remoção da carga de medição	Desconectar as cargas do *Smart Meter* e liga-las contornando o *Smart Meter*.	c)
R05) Carga ilegal em uma área local	Adição de carga na rede elétrica entre os *Smart Meters*.	d)
R06) Carga ilegal com a identificação da carga	Adição de carga na rede elétrica entre os *Smart Meters*.	e)

Fonte: O Autor

A identificação do furto de energia para os casos R02, R03 e R04 são realizados comparando as correntes das fases do *Smart Meter* com a corrente de neutro.

Em condições normais de funcionamento a corrente do neutro deve ser aproximadamente igual a somatória das correntes de fase.

No SoC utilizado, as correntes elétricas podem ser obtidas acessando uma região de memória específica, o algoritmo 3 demostra o pseudocódigo para verificar as correntes de fase e de neutro. Sendo que o parâmetro *numPhase* representa a fase que deseja a corrente, considerando o valor zero como a corrente de neutro e

as variáveis *I0S*, *I1S*, *I2S*, *I3S* representam espaços de memória, o I0S representa o espaço de memória 0x8C.

Algoritmo 3 Recupera informações de consumo pelo CE

```
    função GETCURRENTPHASE(numPhase)
        se numPhase = 0 então
 3:         devolve I0S
        fim se
        se numPhase = 1 então
 6:         devolve I1S
        fim se
        se numPhase = 2 então
 9:         devolve I2S
        fim se
        se numPhase = 3 então
12:         devolve I3S
        fim se
        devolve 0                                   ▷ Retorno padrão
15: fim função
```

Para os casos R05 e R06 é necessário a utilização de um *Smart Meter* "agrupador" (concentrador), pois como as correntes de fase e neutro não circulam pelo medidor, não é possível detectar esses casos de furto por um medidor individual e foge do escopo deste trabalho.

A Figura 14 exibe a ligação de um medidor monofásico com a inversão de corrente, adulteração R01. Para medidores instalados em clientes que não realizam cogeração de energia a solução é simples, a soma das correntes é considerada sempre em seu valor absoluto. Então para medidores configurados para não aceitar a cogeração, mesmo que ocorra uma corrente inversa pelo neutro o consumo do cliente não é afetado por esta tentativa de furto. Também é possível registrar a ocorrência que houve uma tentativa de furto caso a companhia elétrica julgue necessária.

Porém para os clientes de companhia de energia elétrica que optaram e assinaram o contrato para a cogeração de energia não é possível detectar esse furto. Algumas técnicas podem ser aplicadas, porém devido ao baixo poder de processamento do processador optamos por registrar o perfil de consumo do usuário.

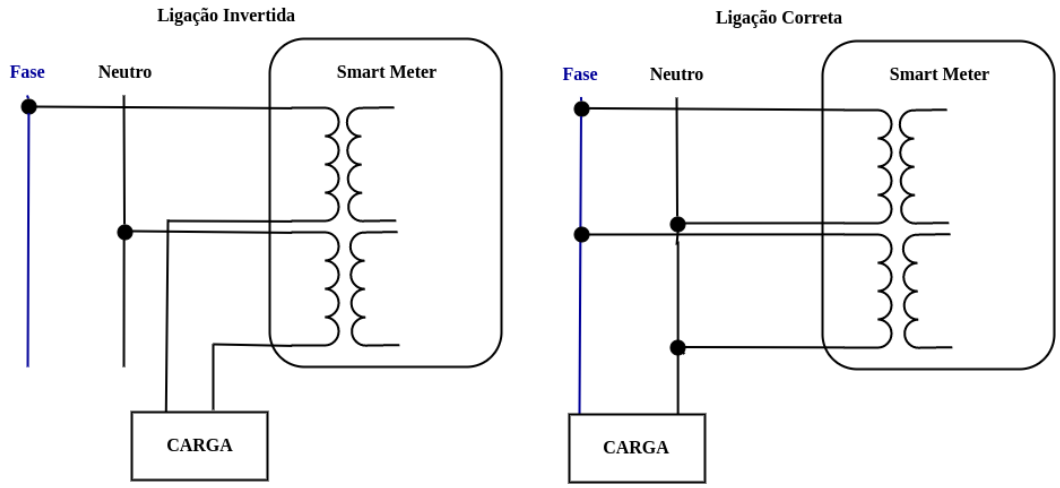


Figura 14 – Ligação do medidor para inverter o fluxo da corrente. Adaptado de [35]

Este perfil pode ser lido e analisado pela companhia elétrica para a identificação de possíveis fraudes.

O perfil de consumo é registrado em intervalos regulares de tempo, a partir destes registros são salvas informações como: horário local, energia ativa, reativa, demanda e tensão, considerado o fluxo direto de energia e o fluxo inverso de energia. As informações salvas e o intervalo de tempo de seu registro, entre 5 e 60 minutos, foram elaborados de acordo com a norma PRODIST – MODULO 5[36].

A Figura 15 demonstra o fluxograma do código de detecção de furto, este código é executado em intervalos regulares de tempo, atividade III na figura Figura 13, e detecta os tipos de furto R01, R02, R03 e R04.

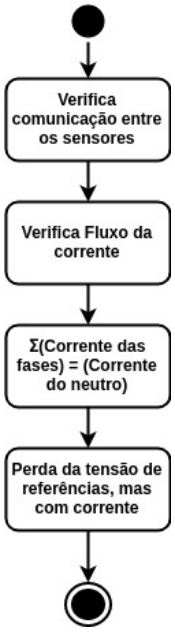


Figura 15 – Fluxo do programa para verificar furto de energia. Fonte: O Autor

4.4 Detecção de fraude do equipamento

Para garantir a integridade dos sensores do *Smart Meter* e suas funções de metrologia, os componentes eletrônicos internos devem ser protegidos, para evitar substituições ou reprogramação e testados para comprovar o seu funcionamento durante o período de operação.

Na pesquisa da literatura foram identificadas 7 ações que são realizadas com o objetivo de prejudicar os componentes internos do *Smart Meter*, para que as funções metrológicas fiquem avariadas. As ações estão listadas na Tabela 4.

Os métodos F01, F02 e F03 descritos na Tabela 4 foram solucionados utilizando um conjunto de acelerômetros, dois acoplados a tampa e um acoplado a placa do *Smart Meter*. Este conjunto monitora a abertura da tampa e vibração anômala proveniente de uma broca ou batida [35].

Na placa do *Smart Meter* também foram anexados medidores de temperatura, para monitorar eventos que indiquem a fraude F04. O monitoramento da temperatura também auxilia da detecção da fraude F05, pois picos de tensão

Tabela 4 – Adulterações físicas do *Smart Meter*.

Adulteração	Descrição
F01) Vibração anômala na caixa do *Smart Meter*	Indica que o *Smart Meter* está sob tentativa física de intrusão (por exemplo, usando uma máquina de perfuração).
F02) Violação da caixa	Indicam que a *Smart Meter* está propenso a danos físicos/ataque ou uma espécie de batida.
F03) Violação da tampa	Indica que o *hardware* do *Smart Meter* está exposto
F04) Superaquecimento e queima de componentes	Superaquecimento para tentar queimar ou derreter componente interno (como um transformador), a fim de desativar funcionalidades específicas.
F05) Pulso de tensão elétrica para queima	Tentativa de queimar sensores aplicando um pulso de alta tensão diretamente sobre o sensor para desativar a capacidade de medição
F06) Número de desconexões	Tentativa de danificar os componentes eletrônicos internos.
F07) Remoção da bateria	Danificar a bateria para evitar a detecção de tentativas de violação quando a energia é desligada.

Fonte: O Autor

ocasionam um aumento de temperatura dos componentes internos, porém não pode ser considerado o único critério para a validação da Fraude, por descargas elétricas atmosférica.

Para a fraude F05 é recomendado fazer uma auditoria dos logs do sistema para identificar o momento da avaria e verificar se o horário da ocorrência não coincide com fenômenos naturais como tempestades que ocorreram do período.

Para os eventos F06 e F07 o monitoramento é realizado exclusivamente via *software*, sem a necessidade de sensores adicionais. Como descrito na Figura 13, ao iniciar o sistema é gerado um *log* de início de operação, logo após o registro deste *log* é feito uma comparação do horário atual com o horário de inicialização anterior, caso o intervalo médio, das últimas 5 reinicializações, destes dois tempos seja inferior a 1ms o sistema considera que o evento F06 está ocorrendo.

O evento F07 é feito através do monitoramento constante da tensão da bateria do sistema, caso a tensão se torne baixa é registrado um *log* informando a situação. Este *log* tem como objetivo informar para a empresa de energia elétrica

que o *Smart Meter* precisa de uma manutenção preditiva.

Caso o tempo de vida da bateria seja menor que o esperado a companhia de energia elétrica pode inclusive acionar a fabricante do *Smart Meter* indicando que houve um problema de fabricação ou uma tentativa de drenagem de carga com o objetivo de adulterar o *Smart Meter*.

Os sensores que detectam a tentativa de intrusão física no equipamento estão ligados a interrupção externa do sistema, caso ocorra uma ação proveniente destes sensores o sistema trata este sinal o mais rápido possível, diminuindo o tempo de exposição ao risco.

Os acelerômetros não estão ligados diretamente ao MCU do *Smart Meter*, os sinais provenientes dos acelerômetros, que precisam ser tratados e multiplexados, são realizados em um coprocessador ligado ao MCU. Este coprocessador realiza a lógica de identificação de abertura de tampa e diferenças entre vibrações provenientes de uma furadeira ou de um caminhão passando pela rua, por exemplo.

4.5 *Smart Meter* com proteção multi-nível e detecção de fraude

Ao integrar as várias soluções propostas até o momento (IoTMAC, detecção de furto e fraude), é apresentado uma versão de *Smart Meter* com maior integridade a vulnerabilidades conhecidas até o momento.

Um equipamento que, apesar de seu baixo poder de processamento, controla o acesso de memória e toma ações objetivas ao detectar uma possível intrusão.

A Figura 16 mostra um diagrama completo do *Smart Meter*, incluindo o SoC e sensores anexados para a identificação de fraudes. Este diagrama representa um conjunto de trabalhos integrados que possibilitaram o desenvolvimento deste equipamento.

O trabalho aqui proposto busca aprimorar a segurança para a memória do SoC do *Smart Meter*, a Figura 17 mostra com mais detalhes o 71M6543, seus componentes internos, sensores e dispositivos conectados.

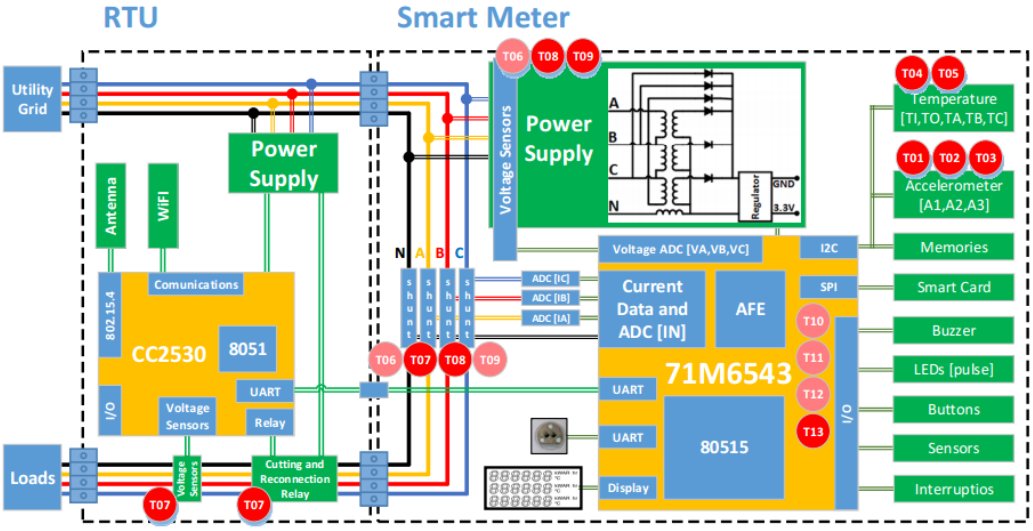


Figura 16 – Diagrama do *Smart Meter* e RTU propostos. (Fonte: O Autor e Lando [35])

Como é possível ver na Figura 17 há duas maneiras de se comunicar com o microcontrolador externamente, ambas por comunicação serial. Porém a diferença entre essas duas comunicações consiste no modo de acesso: uma está conectada ao *Remote Terminal Unit* (RTU), que é acessado pela companhia e recebe comandos externamente via comunicação remota pela UART1; a outra porta serial, UART2, está conectada a um sensor de comunicação infravermelho para o *Smart Meter*.

A proteção referente a autenticação e acesso pela RTU foge do escopo deste trabalho. Porém para o *Smart Meter* proposto foi utilizada a solução implementada por Witkovski *et al.* [37].

Os comandos recebidos pela RTU são considerados de alta integridade, ou seja, a companhia elétrica consegue remotamente realizar leituras de consumo, registros de ocorrência e a gravação de informações como horário do sistema ou uma atualização de *firmware* remotamente. O canal de comunicação entre o RTU e o *Smart Meter*, por estar dentro da caixa do *Smart Meter*, é considerado integro e seguro.

No entanto, sugere-se que a companhia elétrica tenha ferramentas para o eventual desvio de conduta de agentes internos à empresa. Ações como a auditoria

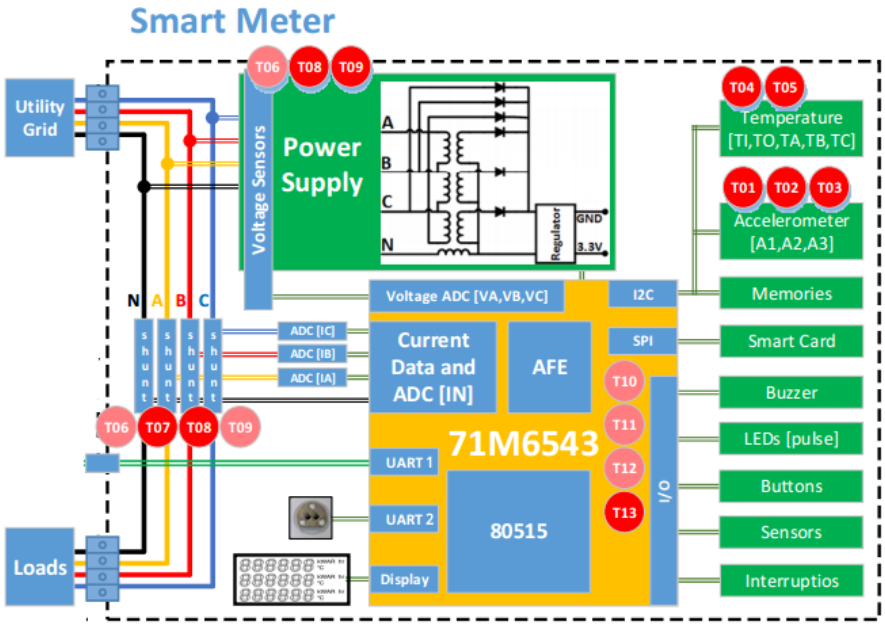


Figura 17 – Diagrama mostrando SoC 71M6543 com os sensores propostos e fonte de alimentação com neutro virtual (Fonte: O Autor e Lando [35])

dos *logs* gerados pelo medidor para a identificação de possíveis problemas são aconselháveis.

A fonte de alimentação representada na Figura 17 é um diferencial da implementação neste trabalho, pois utiliza um neutro virtual para manter a referência de alimentação do SoC. Mesmo que o neutro seja desconectado (R03) o sistema conseguirá manter sua precisão de metrologia, caso esteja conectado ao *Smart Meter* ao menos duas fases o sistema conseguirá realizar as medições corretamente.

Para as técnicas de roubo, descritos anteriormente, os algoritmos presentes do 71M6543 são responsáveis pela a sua detecção e registros de tais ocorrência no *Smart Card*. Para as técnicas de fraude são utilizados os sensores de temperatura e acelerômetro para detectar possíveis problemas. Também há o uso de *buzzer* caso o empresa queira emitir um sinal sonoro de alarme.

A comunicação realizada pelo segundo canal de comunicação serial é considerada de menor integridade e permite apenas a leitura de dados.

5 AVALIAÇÃO

5.1 IoTMAC

Os testes apresentados a seguir foram realizados no simulador uCsim 0.5.4[27] parametrizado para trabalhar como controlador C51 e com configurações semelhantes para o 71M6543. O uCSim imitou um microcontrolador com: RAM de 5KB e 64KB de *flash*, para que pudéssemos medir os dados com maior facilidade.

O primeiro teste consistiu em executar o RTOS com duas tarefas distintas e com duas prioridades diferentes. Uma tarefa executando constantemente, simulando uma operação do CE e com baixa prioridade. Esta atividade por não receber parâmetros externos executa sem proteção de memória. A função da tarefa era gerar números aleatórios (representando informação de consumos) e salvar no endereço de memória de alta integridade $0x2350_{h}$ (valor em hexadecimal).

A outra tarefa de maior prioridade, apesar de carregada em memória, só é executada após um evento de interrupção. A tarefa é acionada através do evento de interrupção da UART2, ou seja, ao receber uma mensagem por esta interface a função interpreta a mensagem e executa o comando solicitado. Como é uma atividade que recebe comandos externos, esta tarefa é executada utilizando as funções de proteção a memória com baixa integridade. A atividade da tarefa era salvar a informação enviada para o espaço de endereço solicitado.

A Figura 18 ilustra o comportamento do sistema ao tratar uma interrupção externa com escalonamento por prioridade. Ao ocorrer uma interrupção o sistema inicia o *Interrupt Service Routine* (ISR) (t2) e ao finalizar ocorre a preempção para a tarefa de maior prioridade (t3). Apenas ao final na tarefa de maior prioridade (t4) o sistema retorna a executar a tarefa que foi interrompida no momento T2. Este comportamento é chamado de função adiada, pois é executada após o final da ISR.

A Figura 19 representa os procedimento que foram adotados nos testes e quais foram os critérios de aprovação ou recusa. O RTOS e o modelo de segurança

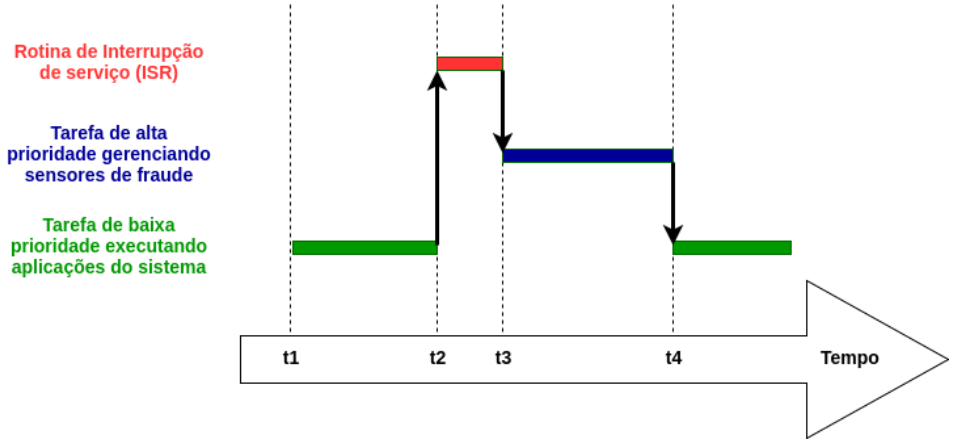


Figura 18 – Comportamento no sistema ao executar tarefas por interrupção. Adaptado de [38]

proposto passaram nos testes propostos na figura Figura 19

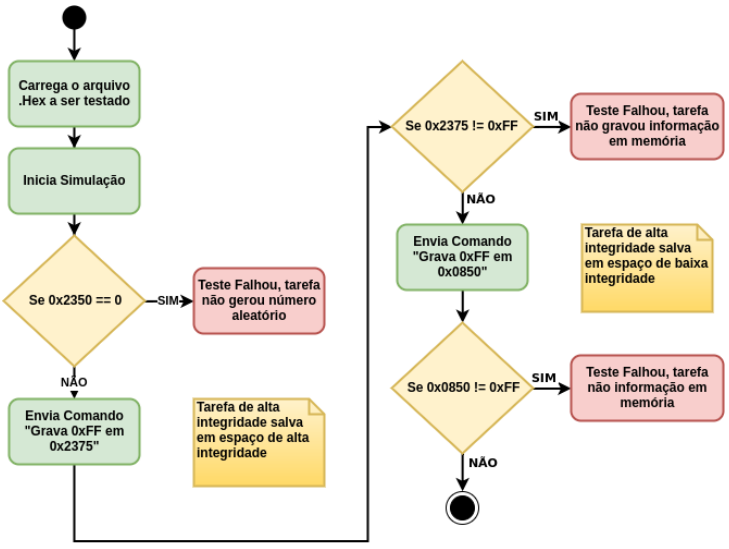


Figura 19 – Diagrama de estados utilizado para o teste da função setMemory em tarefa com alta integridade. Fonte: O Autor

Após o primeiro teste, a tarefa referente à interrupção foi alterada para realizar a leitura do espaço de memória solicitado, a Figura 20 representa os procedimentos que foram adotados na avaliação, incluindo o RTOS e o modelo de segurança proposto.

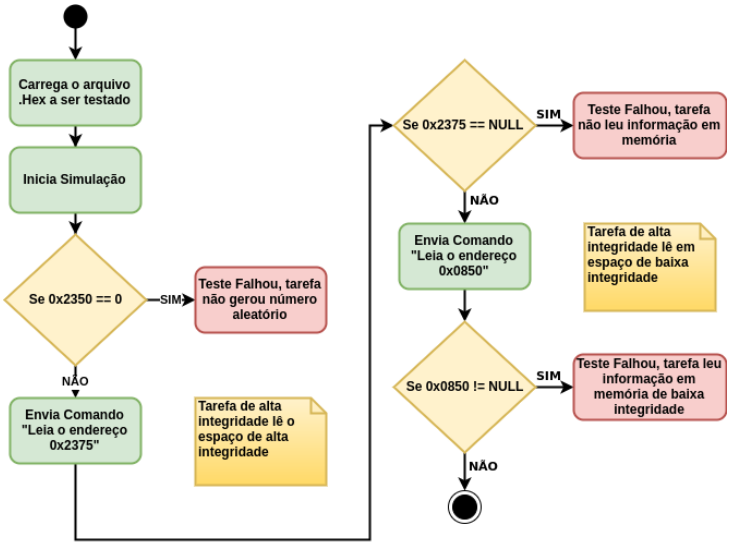


Figura 20 – Diagrama de estados utilizado para o teste da função getMemory em tarefa com alta integridade. Fonte: O Autor

Também foram realizados os testes das funções de leitura e escrita em memória protegida em uma tarefa de baixa integridade. Os testes foram executados seguindo o modelo proposto nas figuras 21 e 22.

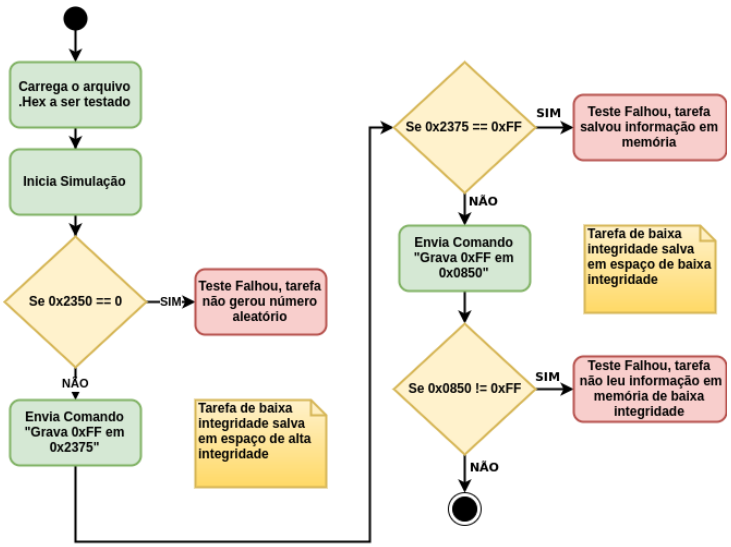


Figura 21 – Diagrama de estados utilizado para o teste da função setMemory em tarefa com baixa integridade. Fonte: O Autor

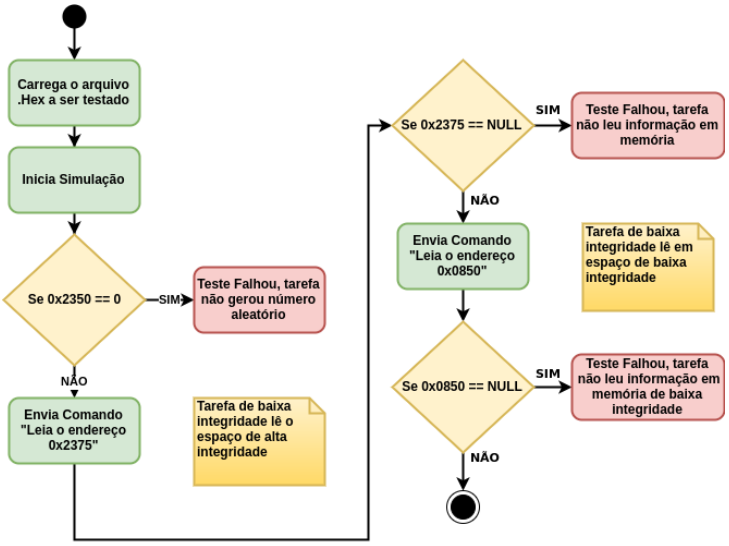


Figura 22 – Diagrama de estados utilizado para o teste da função getMemory em tarefa com baixa integridade. Fonte: O Autor

Com o objetivo de medir o impacto de desempenho da implementação do IoTMAC no sistema, foi medido o tempo para executar uma tarefa de acesso a memória em duas versões diferentes do sistema, com e sem a implementação do controle de acesso a memória. A tarefa consistiu em uma atividade simples, acessar o espaço de memória protegido onde se esperava um valor do tipo inteiro, realizar uma operação de multiplicação e armazenar o resultado desta operação em outro espaço de memória também protegido.

Este tarefa foi executada 30 vezes, a Tabela 5 mostra a mediana dos tempos obtidos e realiza uma comparação entre os tempos das atividades executando com e sem a devida proteção de memória.

Tabela 5 – Tempo de execução das atividades.

Atividade	*Clock*	Tempo(s)	%Desvio Padrão
Sem proteção	4621428	0,4179	0,0061%
Com proteção	19618788	1,774	0,0040%

Fonte: O Autor

Ao dividirmos o tempo da atividade com proteção sobre a atividade com

proteção, percebemos que houve uma queda de desempenho de aproximadamente 4,25 vezes em relação ao modo sem proteção.

Também foi analisado o tempo de execução das novas funções criadas, responsáveis pelo gerenciamento de acesso a memória, **getMemory** e **setMemory**, pois o principal impacto no tempo de execução das tarefas ocorre na chamada destas funções.

A Tabela 6 realiza um resumo do tempo de processamento gasto atribuído exclusivamente as funções adicionais de proteção a memória.

Tabela 6 – Tempo de processamento para as funções de proteção.

Descrição	Clock	Tempo(s)	%Desvio Padrão
getMemory com sucesso	3419064	0,3091	0,0309%
setMemory com sucesso	2141928	0,1937	0,0155%

Fonte: O Autor

Ao analisar o tempo das funções **getMemory** e **setMemory**, percebemos que apenas elas não justificam um aumentando de tempo encontrado na Tabela 5, pois a função de teste deveria executar uma vez a leitura e uma vez a escrita, totalizando uma diferença de aproximadamente 0,5028 segundos, porém a diferença de tempo entre as funções com e sem proteção é de 1,3561 segundos. Esta diferença tem outras atividades do sistema operacional como será explicado a seguir.

Para a execução das tarefas de escrita e leitura protegida também foi adicionado um grau de complexidade ao sistema operacional, ou seja, cada vez que é chamado as funções **getMemory** e **setMemory** o sistema precisa alocar memória para a execução destas atividades, salvar o contexto da função anterior e após a execução da tarefa retornar o contexto. Essa adição de ações do sistema operacional elevou o tempo para a execução da tarefa mencionada acima como um todo, este tempo podemos considerar de aproximadamente 0,8533 segundos (diferença de tempo entre as funções com e sem proteção - soma das funções **getMemory** e **setMemory**).

Apesar das funções com proteção impactar negativamente o desempenho

do sistema, uma queda de desempenho de 324,52%, sua segurança adicional justifica a abordagem, pois diminui a possibilidade de fraude e roubo de energia e, consequentemente, perdas com o faturamento.

A queda de performance também é mitigada pois não impacta todo o sistema, as funções protegidas são necessárias apenas as funções que interagem externamente (comunicação serial ou SPI, por exemplo).

O Apêndice A agrupa os valores obtidos durante os testes.

5.2 Sensores de Fraude

A avaliação dos sensores de fraude levou em consideração o tempo do microcontrolador para responder a um evento de interrupção provocado pelo sensor. Este intervalo entre a interrupção e o tratamento do evento demonstra o tempo de latência de detecção do sistema(Figura 23)

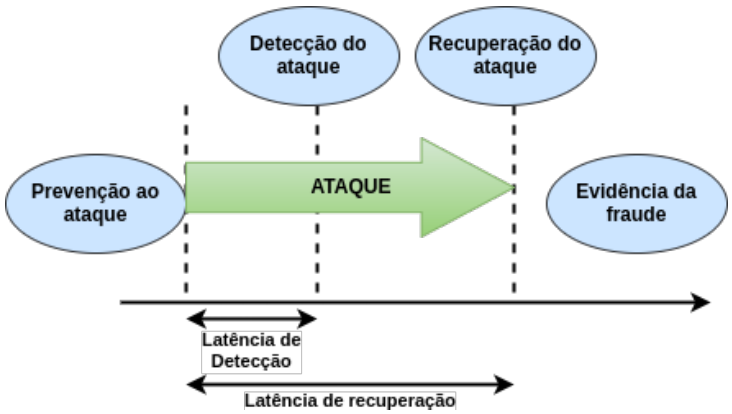


Figura 23 – Tempos de resposta para um ataque, adaptado de [39]

O uso da preempção tem como principal objetivo diminuir este tempo de latência de detecção. Para medir o tempo de detecção de uma tentativa de fraude foram anotados os tempo T2 e T3 indicados na Figura 18. O momento T2 indica o momento em que ocorre uma interrupção externa, o momento T3 indica o início da tarefa com maior prioridade.

Para identificar o tempo de latência de detecção do sistema sem utilizar o modo preemptivo foram executadas 10 tarefas distintas, cada tarefa executando um algoritmo simples de números aleatórios conhecido como Xorshift[40]. O tempo de início de cada atividade era anotado, o objetivo foi encontrar o tempo médio de latência para uma tarefa reiniciar utilizando o escalonamento *Roud-Robin* (cooperativo).

A mediana nos valores de ciclo de *clock* apresentados foi de 8470152 com um desvio padrão de 6,42% indicando que, para o sistema reiniciar uma tarefa com escalonamento *Roud-Robin* e com 10 tarefas simultâneas, o tempo de latência, na pior hipótese, é de aproximadamente 8470152 ciclos de *clock*.

A Tabela 7 realiza um resumo e comparação dos tempos descritos acima.

Tabela 7 – Tempo para o início da atividade de acordo com a técnica de escalonamento escolhida.

Escalonamento	Clock	Tempo(s)	%Desvio Padrão
Round-Robin	8470152	0,7659	6,4200%
Prioridade	345090	0,0312	1,5301%

Fonte: O Autor

Analisando a Tabela 7, percebemos que o uso de prioridades para o tratamento de interrupções impacta positivamente o tempo para a atividade ser iniciada, diminuindo o tempo em aproximadamente 95,93%. Justificando o uso desse tipo de escalonamento.

O Apêndice B agrupa os valores obtidos durante os testes.

6 CONCLUSÃO

O uso de sistema preemptivo se mostrou importante para atender aos sensores de fraude presentes no *Smart Meter* em um intervalo menor de tempo comparado com outra técnica de escalonamento. Executar as tarefas a partir de interrupção por *hardware* reduz o tempo de início da atividade e não depende de outros fatores, como a quantidade de tarefas em execução no momento da ocorrência.

Atender as tarefas relacionadas com evento de fraude rapidamente é um fator importante da proposta, porque diminui o intervalo de tempo em que o sistema fica vulnerável. A diferença para o início das atividades entre o escalonamento *Round-Robin* e de prioridade é de aproximadamente 95,93%, sendo que o tempo de latência para o escalonamento *Round-Robin* é diretamente dependente na quantidade de tarefas em execução.

Ao analisar o uso de proteção de memória via *software*, é perceptível uma queda de desempenho ao acessar e modificar o espaço de memória, uma performance de aproximadamente 4,25 vezes inferior.

Apesar da queda de desempenho o uso de proteção de memória possui a vantagem de manter a integridade do sistema e evitar ataques de *buffer overflow*, por exemplo, sendo que o microcontrolador não possui proteção de memória em nível de *hardware*.

O modelo proposto também tem a vantagem de não impactar em todo o sistema, uma vez que a proteção de memória é utilizada apenas nas atividades que oferecem leitura e escrita de memória para as interfaces de comunicação. Assim outras atividades do medidor, como os registros de eventos em *log* ou a detecção de fraude através dos sensores não são afetados.

Como as funções que precisam de proteção não necessitam de um rápido processamento (comunicação com o servidor ou com um dispositivo conectado via serial) a perda de desempenho pode ser tolerada.

Utilizar as funções que garantem a integridade da memória através do modelo Biba acarreta um aumento do tempo de processamento da tarefa.

6.1 Publicação

A ideia principal do trabalho é parte integrante de um artigo, aceito para publicação (https://secplab.ppgia.pucpr.br/files/papers/2017-3.pdf):

- ABREU, VILMAR; SANTIN, A. O. ; XAVIER, A. S. ; LANDO, A. L. ; WITKOVSKI, A. ; RIBEIRO, R. C. ; STIHLER, M. ; ZAMBENEDETTI, V. C. ; CHUEIRI, I. J. . **A Smart Meter and Smart House Integrated to an IdM and Key-based Scheme for Providing Integral Security for a Smart Grid ICT**. MOBILE NETWORKS AND APPLICATIONS, 2017. (Qualis A2)

6.2 Trabalhos Futuros

Como trabalho futuro pretende-se terminar a implementação do FreeRTOS com a proposta IoTMAC em um dispositivo físico, pois sua validação foi feita através de simulações.

Também buscaremos mostrar que a solução proposta é válida para todos os dispositivos de baixo processamento e não apenas para o microcontrolador apresentado neste trabalho. Desta maneira a contribuição aqui proposta não ficaria restrita ao mercado de *Smart Meters*, mas a todo os produtos que se beneficiaram com a proteção de memória, mesmo sem possuir um *hardware* robusto.

Durante a realização do trabalho foram identificados alguns casos de furto que não fizeram parte deste trabalho e que necessitariam de adição de sensores (*e.g* a adulteração dos sensores de metrologia por magnetismo[41]). Apesar desta técnica não interferir diretamente na proteção de memória, este caso será abordado em novas publicações que serão realizadas para validar o *Smart Meter* seguro.

REFERÊNCIAS

[1] K. Ashton, "That 'internet of things' thing," *RFiD Journal*, vol. 22, no. 7, pp. 97–114, 2009.

[2] T. Resende, "Perdas na distribuição: baixa tensão, altos prejuízos – Reportagem Especial Canal Energia - Abradee - Associação Brasileira de Distribuidores de Energia Elétrica." http://www.abradee.com.br/imprensa/artigos-e-releases/1018-perdas-na-distribuicao-baixa-tensao-altos-prejuizos-reportagem-especial-canal-energia, Acessado em: 01 de Ago de 2076, 2013.

[3] ABRADEE, "Furto e Fraude de Energia - Abradee - Associação Brasileira de Distribuidores de Energia Elétrica." http://www.abradee.com.br/setor-de-distribuicao/perdas/furto-e-fraude-de-energia. Acessado em 02 de Ago de 2017, 2017.

[4] B. Krebs, "Fbi: Smart meter hacks likely to spread," *Krebs on Security. Disponível em: http://krebsonsecurity. com/2012/04/fbi-smart-meter-hacks-likely-to-spread/ Acessado em 25 de Abr. de 2016*, 2012.

[5] X. Li, X. Liang, R. Lu, X. Shen, X. Lin, and H. Zhu, "Securing smart grid: cyber attacks, countermeasures, and challenges," *IEEE Communications Magazine*, vol. 50, no. 8, 2012.

[6] N. Mohammad, A. Barua, and M. A. Arafat, "A smart prepaid energy metering system to control electricity theft," in *Power, Energy and Control (ICPEC), 2013 International Conference on*, pp. 562–565, IEEE, 2013.

[7] N. Tangsunantham, S. Ngamchuen, V. Nontaboot, S. Thepphaeng, and C. Pirak, "Experimental performance analysis of current bypass anti-tampering in smart energy meters," in *Telecommunication Networks and Applications Conference (ATNAC), 2013 Australasian*, pp. 124–129, IEEE, 2013.

[8] Silergy Teridian, *Smart Metering SoCs*, 04 2016. Rev. 2.

[9] K. J. Biba, "Integrity considerations for secure computer systems," tech. rep., DTIC Document, 1977.

[10] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid—the new and improved power grid: A survey," *IEEE communications surveys & tutorials*, vol. 14, no. 4, pp. 944–980, 2012.

[11] S. S. S. R. Depuru, L. Wang, and V. Devabhaktuni, "Smart meters for power grid: Challenges, issues, advantages and status," *Renewable and sustainable energy reviews*, vol. 15, no. 6, pp. 2736–2742, 2011.

[12] H. Farhangi, "The path of the smart grid," *IEEE power and energy magazine*, vol. 8, no. 1, 2010.

[13] G. R. Barai, S. Krishnan, and B. Venkatesh, "Smart metering and functionalities of smart meters in smart grid-a review," in *Electrical Power and Energy Conference (EPEC), 2015 IEEE*, pp. 138–145, IEEE, 2015.

[14] A. Silberschatz, P. B. Galvin, G. Gagne, and A. Silberschatz, *Operating system concepts*, vol. 4. Addison-wesley Reading, 1998.

[15] A. Tanenbaum, *Modern operating systems*. Pearson Education, Inc.„ 2009.

[16] S. McLaughlin, B. Holbert, A. Fawaz, R. Berthier, and S. Zonouz, "A multi-sensor energy theft detection framework for advanced metering infrastructures," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 7, pp. 1319–1330, 2013.

[17] M. LeMay and C. A. Gunter, "Cumulative attestation kernels for embedded systems," in *European Symposium on Research in Computer Security*, pp. 655–670, Springer, 2009.

[18] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "Tytan: tiny trust anchor for tiny devices," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2015.

[19] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in *Proceedings of the Ninth European Conference on Computer Systems*, p. 10, ACM, 2014.

[20] "FreeRTOS License Details." http://www.freertos.org/a00114.html. Acessado em: 02 de Mar. de 2017.

[21] "FreeRTOS contact and support details." http://www.freertos.org/RTOS-contact-and-support.html. Acessado em: 02 de Junho. de 2017.

[22] Silicon Labs, *8K ISP FLASH MCU Family*, 12 2003. Rev. 1.4.

[23] Intel, *NMOS SINGLE-CHIP 8-BIT MICROCONTROLLERS*, 02 1995. Rev. 7.

[24] R. M. Stallman and R. McGrath, *GNU Make: A Program for Directed Recompilation, Version 3. 79. 1.* Free Software Foundation, 2002.

[25] C. SDCC–Small Device, "Compiler [Home Page]. 2006." http://sdcc.sourceforge.net/. Acessado em: 02 de Junho. de 2016, 2016.

[26] A. Malinowski, "Notes of the Port FreeRTOS." http://olek.matthewm.com.pl//courses/ee-ces/examples/R00_ported.txt. Acessado em: 06 de Fev de 2016.

[27] D. Drótos, "μCSim: Software Simulator for Microcontrollers." http://mazsola.iit.uni-miskolc.hu/ drdani/embedded/ucsim/, 1999. Acessado em: 06 de Fev de 2016.

[28] Maxim Integrated Products, "71M6543 Demo Board: USER'S MANUAL," 2012.

[29] Silergy Teridian, *Energy Meter ICs 71M6543FT / 71M6543HT / 71M6543GT / 71M6543GHT*, 04 2016. Rev. 8.

[30] Atmel, *Atmel | SMART ARM-based Flash MCU*, 10 2016. Rev. 11102G.

[31] Maxim Integrated Products, "MAXIM TO ACQUIRE TERIDIAN, A LEADING SUPPLIER OF SYSTEM ON CHIP SOLUTIONS FOR THE SMART METER MARKET," 2010.

[32] Maxim, *Energy Meter ICs 71M6543F / 71M6543G*, 10 2013. Rev. 2.

[33] Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. H. Jakubowski, "Oblivious hashing: A stealthy software integrity verification primitive," in *International Workshop on Information Hiding*, pp. 400–414, Springer, 2002.

[34] R. Hastings and B. Joyce, "Purify: Fast detection of memory leaks and access errors," in *In proc. of the winter 1992 usenix conference*, Citeseer, 1991.

[35] A. L. Lando, "Medidor elétrico inteligente com mecanismo de segurança física e lógica," tech. rep., Pontifícia Universidade Católica do Paraná, 2016.

[36] Agência Nacional de Energia Elétrica – ANEEL, "Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional – PRODIST Módulo 5 – Sistemas de Medição," 2017.

[37] A. Witkovski, A. Santin, V. Abreu, and J. Marynowski, "An idm and key-based authentication method for providing single sign-on in iot," in *Global Communications Conference (GLOBECOM), 2015 IEEE*, pp. 1–6, IEEE, 2015.

[38] "FreeRTOS - FreeRTOS Deferred Interrupt." http://www.freertos.org/deferred_interrupt_processing.html. Acessado em: 17 de Fev de 2016.

[39] S. Ravi, A. Raghunathan, and S. Chakradhar, "Tamper resistance mechanisms for secure embedded systems," in *VLSI Design, 2004. Proceedings. 17th International Conference on*, pp. 605–611, IEEE, 2004.

[40] G. Marsaglia *et al.*, "Xorshift rngs," *Journal of Statistical Software*, vol. 8, no. 14, pp. 1–6, 2003.

[41] Infineon, *Anti-Tampering Solution for E-Meter Application*, 07 2015. Rev. 1.

Apêndices

APÊNDICE A – TEMPO DE EXECUÇÃO DAS FUNÇÕES PROTEGIDAS NO SISTEMA

A tabela abaixo agrupa os tempos dos testes descritos na seção 5.1.

Tempo de execução da tarefa sem proteção		Tempo de execução da tarefa com proteção		Tempo de execução getMemory com sucesso		Tempo de execução SetMemory com sucesso	
sec	clks	sec	clks	sec	clks	sec	clks
0,4178	4621428	1,7743	19623108	0,3094	3422604	0,1936	2141928
0,4178	4621428	1,7739	19618788	0,3090	3418284	0,1937	2142708
0,4178	4620648	1,7739	19618788	0,3090	3418284	0,1936	2141928
0,4178	4621428	1,7739	19618788	0,3090	3418284	0,1936	2141928
0,4179	4621428	1,7740	19618788	0,3091	3418284	0,1937	2141928
0,4179	4621428	1,7740	19618788	0,3090	3419064	0,1937	2141928
0,4178	4620648	1,7740	19618788	0,3091	3418284	0,1938	2142708
0,4179	4621428	1,7740	19618788	0,3090	3419064	0,1937	2141928
0,4178	4621428	1,7740	19618788	0,3091	3419064	0,1938	2142708
0,4178	4621428	1,7740	19618788	0,3091	3418284	0,1937	2142708
0,4179	4621428	1,7740	19618788	0,3091	3418284	0,1938	2142708
0,4179	4621428	1,7739	19618788	0,3091	3418284	0,1937	2141928
0,4179	4621428	1,7740	19618788	0,3091	3419064	0,1936	2141928

Tempo de execução da tarefa sem proteção		Tempo de execução da tarefa com proteção		Tempo de execução getMemory com sucesso		Tempo de execução SetMemory com sucesso	
sec	clks	sec	clks	sec	clks	sec	clks
0,4179	4621428	1,7739	19618788	0,3091	3419064	0,1936	2141928
0,4178	4620648	1,7740	19618788	0,3092	3419064	0,1937	2141928
0,4178	4620648	1,7740	19618788	0,3091	3419064	0,1936	2141928
0,4179	4621428	1,7739	19618788	0,3091	3418284	0,1937	2141928
0,4178	4621428	1,7740	19618788	0,3091	3418284	0,1935	2141928
0,4179	4621428	1,7739	19618788	0,3091	3418284	0,1937	2141928
0,4179	4621428	1,7740	19618788	0,3095	3422604	0,1936	2141928
0,4178	4621428	1,7740	19618788	0,3091	3418284	0,1937	2142708
0,4179	4621428	1,7740	19618788	0,3089	3419064	0,1936	2141928
0,4178	4621428	1,7740	19618788	0,3089	3419064	0,1937	2141928
0,4179	4621428	1,7740	19618788	0,3089	3418284	0,1936	2141928
0,4179	4621428	1,7739	19618788	0,3089	3418284	0,1935	2141928
0,4179	4621428	1,7739	19618788	0,3089	3419064	0,1935	2141928
0,4178	4621428	1,7740	19618788	0,3089	3418284	0,1936	2141928
0,4178	4620648	1,7739	19618788	0,3090	3419064	0,1937	2141928
0,4179	4621428	1,7740	19618788	0,3089	3419064	0,1929	2142708
0,4178	4621428	1,7740	19618788	0,3089	3419064	0,1940	2141928
0,4178	4621428	1,7740	19618788	0,3089	3419064	0,1939	2141928

APÊNDICE B – TEMPO PARA INICIAR UMA ATIVIDADE DE ACORDO COM O ESCALONAMENTO DO SISTEMA

A tabela abaixo agrupa os tempos dos testes descritos na seção 5.2.

Round-robin		Interrupção	
sec	clks	sec	clks
0.4288	4742256	0,03128	345936
0.7658	8470056	0,03056	338040
0.7659	8470260	0,03083	341052
0.7658	8470152	0,03114	344340
0.7658	8470104	0,03160	349428
0.7658	8470164	0,03150	348540
0.7659	8470152	0,03079	339840
0.7658	8470200	0,03040	336672
0.7658	8470116	0,03100	342120
0.7658	8470152	0,03160	348636
0.7658	8469324	0,03130	346944
0.7659	8470308	0,03129	346644
0.7658	8470056	0,03129	346476
0.7658	8470152	0,03200	354096
0.7659	8470212	0,03089	341712

Round-robin		Interrupção	
sec	clks	sec	clks
0.7659	8470152	0,03170	350088
0.7659	8470056	0,03050	337416
0.7659	8470260	0,03180	351564
0.7659	8470152	0,03029	335748
0.7659	8470104	0,03049	336996
0.7659	8470164	0,03150	348228
0.7658	8469372	0,03170	351072
0.7659	8470200	0,03070	338988
0.7658	8470164	0,03079	341064
0.7658	8470152	0,03040	336852
0.7659	8470104	0,03170	350748
0.7658	8470260	0,03160	348444
0.7659	8470056	0,03090	342552
0.7657	8470152	0,03069	339828
0.7659	8470212	0,03050	336612
0.7658	8470152	0,03159	348744