



UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

JOÃO LUCAS LIMA MONTEIRO

**IMPLEMENTANDO UMA ATUALIZAÇÃO DE FIRMWARE SEGURA EM
MEDIDORES DE ENERGIA POR MEIO DO PADRÃO DLMS/COSEM**

FORTALEZA

2024

JOÃO LUCAS LIMA MONTEIRO

IMPLEMENTANDO UMA ATUALIZAÇÃO DE FIRMWARE SEGURA EM MEDIDORES
DE ENERGIA POR MEIO DO PADRÃO DLMS/COSEM

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Orientador: Prof. Me. Ricardo Jardel Nunes da Silveira.

Coorientador: Me. José Danilo da Silva Coutinho Filho.

FORTALEZA

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M777i Monteiro, João Lucas Lima.
Implementando um atualização de Firmware segura em medidores de energia por meio do padrão
DLMS/COSEM / João Lucas Lima Monteiro. – 2024.
76 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia,
Curso de Engenharia de Computação, Fortaleza, 2024.
Orientação: Prof. Me. Ricardo Jardel Nunes da Silveira.
Coorientação: Prof. Me. José Danilo da Silva Coutinho Filho.

1. Medidores de energia. 2. Atualização de imagem. 3. Redes inteligentes. I. Título.

CDD 621.39

JOÃO LUCAS LIMA MONTEIRO

IMPLEMENTANDO UMA ATUALIZAÇÃO DE FIRMWARE SEGURA EM MEDIDORES
DE ENERGIA POR MEIO DO PADRÃO DLMS/COSEM

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Aprovada em: 23 de Setembro de 2024

BANCA EXAMINADORA

Prof. Me. Ricardo Jardel Nunes da
Silveira (Orientador)
Universidade Federal do Ceará (UFC)

Me. José Danilo da Silva Coutinho
Filho (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Alexandre Augusto da Penha Coelho
Universidade Federal do Ceará (UFC)

À minha família, por acreditar em mim e sempre me apoiar. O cuidado, dedicação e amor de vocês foram essenciais para eu seguir em frente, especialmente nos momentos mais difíceis. Sua presença e apoio me deram segurança e a certeza de que nunca estive sozinho nessa jornada.

AGRADECIMENTOS

Gostaria de expressar minha gratidão ao Prof. Me. Ricardo Jardel Nunes da Silveira, pela orientação, pela transmissão de conhecimentos e pela paciência ao longo do desenvolvimento deste trabalho.

Agradeço também à banca examinadora, composta pelos professores Prof. Dr. Alexandre Augusto da Penha Coelho e Me. José Danilo da Silva Coutinho, pela dedicação, interesse e sugestões construtivas durante a avaliação deste projeto.

Finalmente, meu sincero agradecimento a todos que, de alguma forma, contribuíram para minha jornada acadêmica, oferecendo apoio, incentivo e colaboração ao longo dessa trajetória.

"Tudo o que você pode fazer ou sonhar que pode, comece. A ousadia tem genialidade, poder e magia em si." (Johann Wolfgang von Goethe)

RESUMO

A *Advanced Metering Infrastructure* (AMI) é parte das Redes Inteligentes (Smart Grids) e desempenha um papel crucial na comunicação entre concessionárias e consumidores. Um de seus componentes principais são os medidores inteligentes de energia, usados em residências e indústrias para coletar dados da rede elétrica. Com o aumento da popularidade desses dispositivos, tornou-se necessário desenvolver novos códigos de programação para adicionar serviços e corrigir falhas, garantindo alta confiabilidade e segurança nas medições. No entanto, os medidores podem apresentar problemas de conformidade regulatória e falhas de segurança se o firmware não for atualizado regularmente. Então, sendo a atualização da programação destes dispositivos uma solução eficaz para garantir esses requisitos, o objetivo deste trabalho é propor uma implementação da atualização da programação de medidores inteligentes de maneira segura, segundo o padrão DLMS/COSEM. Para isso, é utilizada assinatura no arquivo do código de programação do medidor além do uso da criptografia e autenticação do mecanismo de segurança de alto nível para assegurar a comunicação entre o medidor e o dispositivo cliente. Sendo assim, este trabalho inclui um passo a passo constituído de um estudo aprofundado, codificação de um algoritmo com máquina de estados bem definida e um sistema completo que permite a atualização do medidor inteligente via comunicação serial. Este trabalho também compara o processo manual realizado pelo software do fabricante que ocorre em 10 minutos, comparado aos 12 minutos do tempo de duração da implementação desenvolvida. Além disso, utilização do mecanismo de segurança de alto nível do DLMS/COSEM oferece proteção contra interceptações, enquanto a autenticação impede acessos não autorizados, garantindo um alto nível de segurança e confiabilidade para as atualizações de firmware dos medidores inteligentes. Por fim, a solução proposta mostrou-se competitiva em termos de desempenho, representa uma abordagem promissora para futuras otimizações e pode gerar impactos significativos no setor de energia elétrica.

Palavras-chave: Medidores de energia; Atualização de imagem; Redes inteligentes.

ABSTRACT

AMI is part of Smart Grids and plays a crucial role in communication between utilities and consumers. One of its main components are smart energy meters, used in homes and industries to collect data from the electricity grid. With the increasing popularity of these devices, it has become necessary to develop new programming codes to add services and correct faults, ensuring high reliability and security in measurements. However, meters can present regulatory compliance problems and security flaws if the firmware is not updated regularly. Therefore, since updating the programming of these devices is an effective solution for guaranteeing these requirements, the aim of this work is to propose an implementation for updating the programming of smart meters in a secure manner, according to the DLMS/COSEM standard. This is done by signing the meter's programming code file and using the encryption and authentication of the high-level security mechanism to ensure communication between the meter and the client device. Therefore, this work includes a step-by-step process consisting of an in-depth study, coding of an algorithm with a well-defined state machine and a complete system that allows the smart meter to be updated via serial communication. This work also compares the manual process carried out by the manufacturer's software, which takes 10 minutes, with the 12-minute duration of the implementation developed. In addition, the use of the DLMS/COSEM's high-level security mechanism provides protection against interception, while authentication prevents unauthorized access, guaranteeing a high level of security and reliability for smart meter firmware updates. Finally, the proposed solution proved to be competitive in terms of performance, represents a promising approach for future optimizations and could generate significant impacts in the electricity sector.

Keywords: Smart Meters; Firmware Update; Smart Grids.

LISTA DE FIGURAS

Figura 1 – Objeto de estudo	16
Figura 2 – Medidor inteligente de estado sólido à esquerda e um medidor eletromecânico à direita	19
Figura 3 – Medidor inteligente NANSEN NSX P213i	20
Figura 4 – Dispositivo coletor de dados	22
Figura 5 – Abordagem do COSEM: Modelagem, Envio de Mensagens e Transporte . .	25
Figura 6 – Representação da encriptação e decriptação do modo Galois/Counter Mode (GCM)	33
Figura 7 – Ferramenta SCV Cryptomanager	34
Figura 8 – Descritografia do <i>InitiateRequest</i> no SCV Cryptomanager	34
Figura 9 – Descritografia do <i>InitiateResponse</i> no SCV Cryptomanager	37
Figura 10 – Descritografia do <i>ActionRequest</i> no SCV Cryptomanager	38
Figura 11 – Autenticação do desafio no SCV Cryptomanager	41
Figura 12 – Descritografia do <i>ActionResponse</i> no SCV Cryptomanager	42
Figura 13 – Autenticação do desafio no SCV Cryptomanager	44
Figura 14 – Estrutura do atributo <i>image_to_activate_info</i>	47
Figura 15 – Estrutura do parâmetro de entrada <i>data</i> do método image_transfer_initiate(data)	48
Figura 16 – Estrutura do parâmetro de entrada <i>data</i> do método image_block_transfer(data)	48
Figura 17 – Interface gráfica do programa Sanplat do medidor de energia NSX P213i . .	57
Figura 18 – Diagrama de sequência	60
Figura 19 – Primeira parte do diagrama de fluxo Image Transfer	62
Figura 20 – Segunda parte do diagrama de fluxo Image Transfer	63
Figura 21 – Instância da variável de controle do processo de Image Transfer	64
Figura 22 – Depuração do processo de transferência de imagem do medidor original . .	71
Figura 23 – Depuração do processo de transferência de imagem do medidor alterada . .	72

LISTA DE TABELAS

Tabela 1 – Estrutura dos códigos OBIS e definição dos grupos	25
Tabela 2 – Interface Class Image Transfer	45
Tabela 3 – Tempo de duração do processo de atualização do <i>firmware</i> do medidor de energia	67

LISTA DE ABREVIATURAS E SIGLAS

ACSE	Association Control Service Element
AES-GCM	Advanced Encryption Standard Galois/Counter Mode
AK	Authentication Key
AMI	<i>Advanced Metering Infrastructure</i>
ANEEL	Agência Nacional de Energia Elétrica
APDU	Application Protocol Data Unit
COSEM	Companion Specification for Energy Metering
DLMS	Device Language Message Specification
EEPROM	Electrically Erasable Programmable Read-Only Memory
EK	Encryption Key
GCM	Galois/Counter Mode
GMAC	Galois Message Authentication Code
HLS	High Level Security
LoRaWAN	Long Range Wide Area Network
MDMS	Meter Data Management System
PDU	Protocol Data Unit
PRODIST	Procedimentos de Distribuição de Energia Elétrica
RAM	Random Access Memory
ROM	Read-Only Memory
XML	Extensible Markup Language

SUMÁRIO

1	INTRODUÇÃO	14
1.1	Justificativa	15
1.2	Objetivos	15
1.3	Metodologia	17
2	FUNDAMENTAÇÃO TEÓRICA	18
2.1	Redes Inteligentes	18
2.2	Infraestrutura Avançada de Medição	18
2.2.1	Medidor Inteligente	19
2.2.1.1	<i>Funcionalidades</i>	19
2.2.1.2	<i>Medidor Inteligente NANSEN NSXi</i>	20
2.2.1.2.1	Especificações	21
2.2.1.2.2	Comunicação do Medidor	21
2.2.2	Sistema de Gerenciamento de Dados	21
2.2.3	Concentrador de Dados	22
2.3	Especificação DLMS/COSEM	22
2.3.1	Segurança do Padrão DLMS/COSEM	25
2.3.1.1	<i>Sem Segurança</i>	26
2.3.1.2	<i>Segurança de Nível Baixo</i>	28
2.3.1.3	<i>Segurança de Nível Alto</i>	30
2.3.1.3.1	Envio do InitiateRequest	31
2.3.1.3.2	Recebimento do InitiateResponse	35
2.3.1.3.3	Envio do Desafio do cliente	37
2.3.1.3.4	Envio do Desafio do servidor	40
2.3.2	Image Transfer	44
2.3.2.1	<i>Descrição dos Atributos</i>	45
2.3.2.2	<i>Descrição dos Métodos</i>	47
2.4	Firmware	48
2.4.1	Segurança de Firmware	49
2.4.2	Atualização de Firmware	50
2.4.3	Linguagem de Programação C	52

2.4.4	<i>Padrão de Codificação BARR</i>	52
3	METODOLOGIA	54
3.1	Estudo do Padrão DLMS/COSEM e Objeto Image Transfer	55
3.2	Instalação do Sanplat e Estudo da Funcionalidade de Atualização de Firmware	56
3.3	Definição dos Pré-requisitos para Elaboração da Implementação	57
3.4	Definição da Arquitetura de Firmware da Implementação	59
3.5	Definição do Design de Firmware da Implementação	60
3.6	Gerenciamento de Estados da Implementação	64
4	RESULTADOS	67
4.1	Descrição dos Cenários Testados	67
4.2	Resultados Quantitativos: Comparação de Tempos de Atualização	67
4.3	Resultados Qualitativos	68
5	CONCLUSÕES E TRABALHOS FUTUROS	73
	REFERÊNCIAS	75

1 INTRODUÇÃO

A preocupação com a geração e distribuição de energia tem aumentado ao longo dos anos, demandando um desenvolvimento tecnológico mais robusto para atender às necessidades crescentes de consumo. A ampliação do uso de medidores inteligentes em residências e indústrias tornou-se uma realidade, impulsionada pelos benefícios oferecidos, como coleta precisa de dados, que permite uma gestão mais eficiente dos recursos, o monitoramento em tempo real, que garante maior transparência para os consumidores, e a adoção de preços dinâmicos (EL-HAWARY, 2014) a qual promove um uso mais consciente da energia. Além disso, as concessionárias de energia no Brasil estão sujeitas aos requisitos das normas regulatórias da Agência Nacional de Energia Elétrica (ANEEL), que publica novas resoluções normativas para estabelecer ou alterar procedimentos de distribuição existentes. Dessa maneira, esse progresso destaca a necessidade de segurança e de confiabilidade na prestação dos serviços nessas atividades. Por conta disso, os temas relacionados às redes inteligentes tornaram-se cada vez mais relevantes para garantir a segurança das operações no setor de energia.

No contexto das redes inteligentes, também chamadas de *Smart Grids*, estas vão além da simples comunicação com dispositivos medidores de energia, abrangendo diversas etapas além da coleta de dados da rede elétrica. Elas emergiram como resposta à necessidade de aprimoramento da infraestrutura energética, buscando torná-la mais eficiente, confiável e segura. O funcionamento da rede está intrinsecamente ligado a uma série de etapas, desde a geração de energia até sua distribuição e consumo. Por exemplo, o processo de cobrança é efetuado apenas após a conclusão e adequado funcionamento dessas etapas.

No processo de consumo da rede elétrica, a AMI desempenha o papel de controlar a leitura das medições e transmitir as informações obtidas. Essa estrutura compreende o medidor inteligente, o dispositivo coletor de dados, o sistema de gerenciamento de dados e as redes de comunicação (KEBOTOGETSE *et al.*, 2021). O medidor de energia, responsável pelo monitoramento do consumo e armazenamento de informações, transfere ao dispositivo coletor de dados os dados ao longo do tempo. Assim, para garantir o funcionamento adequado do sistema, livre de ameaças e conforme as exigências regulatórias, o medidor inteligente deve atender a certos requisitos. Entre eles, é fundamental que o sistema seja completo e robusto, impedindo leituras, alterações de estado ou atualizações não autorizadas do código de programação, também conhecido como *firmware*.

1.1 Justificativa

A solução proposta neste trabalho consiste na atualização remota segura do *firmware* em medidores de energia, atendendo às crescentes demandas por segurança e confiabilidade no setor energético. Esta abordagem oferece uma vantagem significativa em relação ao processo manual realizado pelos fabricantes de medidores, uma vez que a característica remota da solução impacta positivamente os processos de controle. Enquanto a atualização presencial do *firmware* demanda a alocação de indivíduos em diferentes locais, gerando dispêndio de tempo e custos significativos, a proposta remota deste trabalho se destaca por sua eficiência e economia, característica importante que garante o processo em locais que seriam de difícil acesso. Dessa forma, os benefícios deste estudo tornam-se claros, especialmente na minimização de intervenções indesejadas nos processos de medições e na circulação de informações no sistema, o que é crucial para manter a integridade da rede elétrica. Entre as ameaças estão as possíveis flutuações de energia na rede elétrica, apagões, interceptação de informações (OTUOZE *et al.*, 2018) ou aplicação de multas elevadas para as empresas do setor de energia que não seguirem as normas regulatórias da ANEEL. Tais situações podem ocorrer caso o medidor de energia esteja com uma versão de *firmware* que possua falhas de segurança ou a ausência de funcionalidade obrigatórias.

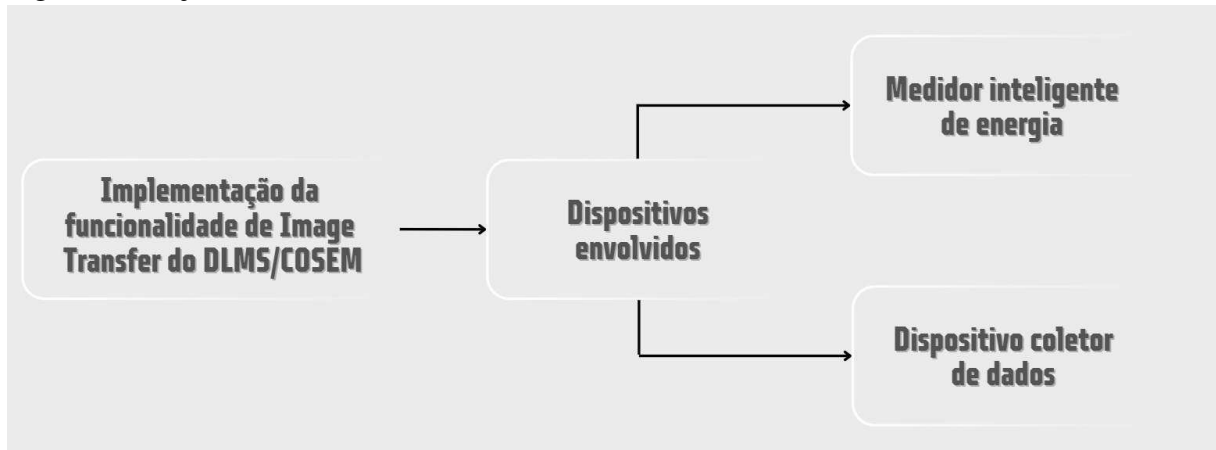
A proposta busca manter a excelência operacional desses equipamentos amplamente empregados em residências e indústrias, aprimorando os dispositivos de medição de consumo de energia elétrica. Ao garantir uma atualização segura do *firmware*, objetiva-se preservar a integridade e confiabilidade dos dados coletados pelos consumidores, conferindo-lhes uma gestão mais eficaz dos recursos energéticos e contribuindo para uma colaboração mais eficaz com as concessionárias. Além de fortalecer a cibersegurança dos medidores de energia, a melhoria proposta nesta pesquisa também promove o uso mais consciente e sustentável dos recursos, em perfeita sintonia com os princípios das redes inteligentes e as necessidades emergentes do setor energético.

1.2 Objetivos

O presente trabalho propõe uma implementação sólida e relevante da codificação de um *firmware* em linguagem de programação C para um dispositivo coletor de dados. Para facilitar o entendimento, a Figura 1 foi elaborada para explicitar o objeto de estudo principal desta proposta. Esse objeto é a implementação de forma viável e segura de uma atualização de

firmware em medidores de energia, a fim de garantir a autenticidade, integridade e confiabilidade das atualizações de imagem, ou seja, prevenindo possíveis falhas de segurança e ameaças à infraestrutura elétrica e aprimorando a confiabilidade, a segurança e a eficiência do sistema.

Figura 1 – Objeto de estudo



Fonte: Elaborado pelo autor.

Os objetivos específicos deste estudo são delineados da seguinte forma:

- a) Analisar e compreender a arquitetura e o funcionamento dos medidores de energia inteligentes, incluindo a AMI e os princípios das redes inteligentes *Smart Grids*;
- b) Identificar a importância da atualização de *firmware* em relação ao cumprimento de normas regulatórias dos Procedimentos de Distribuição de Energia Elétrica (PRODIST);
- c) Estudar em detalhes o padrão DLMS/COSEM e sua aplicação na transmissão segura de atualizações de *firmware* para medidores de energia, identificando as principais características de segurança do padrão;
- d) Propor e implementar uma estrutura de verificação da imagem de um medidor através da utilização de assinatura de imagem para garantir a autenticidade e integridade além da implementação de uma atualização de *firmware* segura baseada no padrão DLMS/COSEM;
- e) Realizar testes e simulações para avaliar a eficácia e a segurança da solução proposta, comparando-a com o método tradicional de atualização de *firmware* feito pelo próprio fabricante;
- f) Identificar possíveis desafios e limitações da implementação da solução proposta, bem como sugestões para futuras melhorias e aprimoramentos no contexto da segurança em medidores de energia.

1.3 Metodologia

Este trabalho utilizou uma metodologia estruturada para o desenvolvimento de uma funcionalidade de atualização de firmware em dispositivos acoplados a medidores de energia, utilizando o padrão DLMS/COSEM. Inicialmente, foi realizado um estudo detalhado desse padrão, com ênfase no Objeto *Image Transfer*, essencial para o processo de atualização.

Em seguida, foi utilizado o *software* Sanplat, da fabricante do medidor, para investigar a funcionalidade de atualização de *firmware* e entender a comunicação entre o dispositivo e o medidor. A partir disso, foram definidos os pré-requisitos técnicos para a implementação.

Com os pré-requisitos em mãos, a arquitetura de *firmware* foi planejada, estruturando as funcionalidades principais e integrando as operações de comunicação e armazenamento. O design de *firmware* detalhou a lógica de controle e as interações com o medidor durante o processo de atualização.

Finalmente, o gerenciamento de estados foi implementado para garantir o funcionamento adequado em diferentes cenários, lidando com falhas de comunicação e assegurando a conclusão segura do processo de atualização.

2 FUNDAMENTAÇÃO TEÓRICA

Os conceitos e fundamentos teóricos essenciais necessários que constituem a base para o desenvolvimento deste trabalho são apresentados neste capítulo.

2.1 Redes Inteligentes

Devido à crescente adoção da geração de energia distribuída por fontes renováveis, a evolução do sistema de geração e de distribuição de energia elétrica foi responsável por dar início às redes inteligentes (GUNGOR *et al.*, 2011). Estas redes representam uma infraestrutura moderna que busca otimizar a eficiência, confiabilidade e segurança da rede elétrica, empregando tecnologias de ponta e mecanismos automatizados. Para que isso seja possível, é necessário a coleta de informações de forma remota. As redes inteligentes apresentam inúmeros sensores tecnológicos conectados para obtenção de dados precisos. Por conta disso, torna-se mais fácil monitorar o sistema, suas limitações e a ocorrência de falhas, o que facilita o fornecimento de energia segura e permite um diagnóstico em tempo real da rede.

2.2 Infraestrutura Avançada de Medição

Para compreender esta infraestrutura, considera-se o seguinte trecho adaptado da Smart Energy International (2022):

A AMI, ou infraestrutura de medição avançada, permite que os serviços de utilidade pública se adaptem às mudanças na procura dos consumidores, como os recursos energéticos amplamente distribuídos e o rápido aumento da utilização de automóveis elétricos. As novas tecnologias de comunicação estão a permitir a avaliação de problemas de medição e de rede através de streaming de dados, que podem digerir e interpretar milhões de mensagens em tempo real. Estes desenvolvimentos contribuem para um processo mais rápido de modernização da rede, o que abre novas oportunidades de funcionamento dos serviços públicos e melhora a satisfação dos clientes.

Especificamente, a AMI funciona como sistema que adiciona a comunicação à rede inteligente. A comunicação ocorre de maneira bidirecional entre concessionárias de energia e usuários. Dessa forma, a qualidade do fornecimento de energia é significativa, pois isso facilita o gerenciamento, adição e substituição de serviços públicos (NATIONAL ENERGY TECHNOLOGY LABORATORY, 2008).

Segundo Kebotogetse *et al.* (2021), a AMI divide-se em quatro componentes: medidor inteligente, Meter Data Management System (MDMS), concentrador de dados e redes de

comunicação.

2.2.1 Medidor Inteligente

O medidor de energia é o dispositivo capaz de medir o consumo de energia elétrica na rede a qual ele está conectado. Inicialmente, utilizavam-se somente medidores eletromecânicos convencionais. Estes dispositivos registravam somente o total de energia consumida durante um período de tempo. Com a evolução do setor energético, tornou-se necessário um dispositivo mais robusto com diversos sensores para cobrir uma maior quantidade de dados coletados. Dessa forma, foi desenvolvido com estas funcionalidades o medidor de energia inteligente, também chamado de *Smart Meter*. Este dispositivo pode registrar o consumo em tempo real e comunicar com uma central gerenciadora enviando informações de estado, notificação de interrupção ou necessidade de restauração (BARAI *et al.*, 2015).

Figura 2 – Medidor inteligente de estado sólido à esquerda e um medidor eletromecânico à direita



Fonte: National Energy Technology Laboratory (2008, p. 17).

2.2.1.1 Funcionalidades

Além da funcionalidade já citada na seção anterior, o medidor inteligente apresenta também as seguintes funcionalidades (NATIONAL ENERGY TECHNOLOGY LABORATORY, 2008):

- a) Preços baseados no tempo;
- b) Dados de consumo para o consumidor e para o serviço público;
- c) Medição da rede;
- d) Notificação de perda de energia (e restabelecimento);
- e) Operações remotas de ligar/desligar;
- f) Limitação da carga para efeitos de "mau pagamento" ou de resposta à procura;
- g) Pré-pagamento de energia;
- h) Monitorização da qualidade da energia;
- i) Detecção de adulteração e roubo de energia;
- j) Comunicações com outros dispositivos inteligentes em casa;

2.2.1.2 Medidor Inteligente NANSEN NSXi

O medidor utilizado neste trabalho foi o modelo NSX P213i fabricado pela *NANSEN*, como mostra a Figura 3. Este equipamento foi desenvolvido para garantir as demandas das empresas que precisam monitorar a rede elétrica de seus clientes. Segundo o próprio fabricante, a linha NSXi foi construída para ser estável, robusta e durável, com vida útil de até 13 anos.

Figura 3 – Medidor inteligente NANSEN NSX P213i



Fonte: Nansen Instrumentos de Precisão (2023).

2.2.1.2.1 Especificações

As características técnicas do medidor inteligente utilizado estão especificadas a seguir:

- a) Classe B (1%) ligação direta;
- b) Corrente Nominal: 15A;
- c) Corrente Máxima: 120A;
- d) Constante de pulso: 1,0 Wh/pulso;
- e) Tensão de operação: 60V a 285V;
- f) Tensão Nominal: 120V, 240V, 120/240V;
- g) Frequência: 50Hz ou 60Hz;
- h) Temperatura de operação: -40°C a 85°C;
- i) Medições: Ativa, Reativa, Demanda e Postos Horários.

2.2.1.2.2 Comunicação do Medidor

A comunicação com o medidor Nansen NSXi pode ser realizada local ou remotamente. Para este estudo, foi utilizado a saída física RS485 para comunicação com o padrão DLMS/COSEM.

2.2.2 Sistema de Gerenciamento de Dados

Para gerenciar os dados da AMI, torna-se necessário a existência de um componente responsável por importar, verificar, editar e processar os dados coletados da infraestrutura antes de tornar os dados disponíveis para faturamento e análise. Esse mecanismo é chamado de MDMS, um sistema ou aplicação que na sua essência recebe dados de um dispositivo Concentrador de Dados, responsável por fazer a coleta de informações dos medidores, e os processa com fins de cobrança (DUSA *et al.*, 2015).

Além disso, este sistema possui um papel muito importante que será utilizado neste estudo: enviar e receber comandos para controlar o processo de atualização de imagem implementado no dispositivo Concentrador de Dados. Isto será importante, pois a inicialização e verificação do processo ocorre por meio do envio e recepção de comandos remotos.

2.2.3 Concentrador de Dados

A coleta de dados do medidor é feita por um dispositivo acoplado responsável por intermediar a comunicação entre o MDMS e o Medidor Inteligente. Este dispositivo chama-se Concentrador de Dados ou dispositivo coletor. Dessa forma, para os fins desse estudo, a Figura 4 mostra o dispositivo utilizado.

Figura 4 – Dispositivo coletor de dados



Fonte: Elaborado pelo autor (2023).

2.3 Especificação DLMS/COSEM

O protocolo Device Language Message Specification (DLMS), em conjunto com sua extensão Companion Specification for Energy Metering (COSEM), representa uma inovação significativa no campo da comunicação entre dispositivos de medição e sistemas de controle de energia elétrica. Esse protocolo, amplamente adotado no setor, foi desenvolvido para garantir interoperabilidade, segurança e flexibilidade nas trocas de informações entre dispositivos, especialmente em ambientes de medição inteligente.

No contexto deste trabalho, a implementação da atualização de *firmware* do medidor de energia utiliza a arquitetura cliente-servidor do protocolo DLMS/COSEM. Nessa arquitetura, o medidor de energia desempenha o papel de servidor, sendo responsável por armazenar e fornecer os dados de medição, bem como receber comandos de controle e atualizações. Por outro lado, o dispositivo coletor de dados, que está fisicamente acoplado ao medidor, atua como o cliente, iniciando as comunicações e enviando solicitações para leitura de dados ou comandos para atualização do *firmware*. Essa relação de cliente-servidor é fundamental para garantir que o dispositivo cliente possa gerenciar remotamente as operações do medidor de forma segura

e eficiente, utilizando os mecanismos de segurança e autenticação fornecidos pelo protocolo DLMS/COSEM.

Inicialmente, o protocolo DLMS surgiu como uma linguagem de especificação de mensagens cujo objetivo era padronizar a troca de informações entre equipamentos de medição e sistemas de gerenciamento. Na primeira fase, o foco foi estabelecer um conjunto de diretrizes e formatos para permitir uma comunicação confiável e eficiente entre dispositivos de diferentes fabricantes. Essa abordagem fornece uma base sólida para interoperabilidade, permitindo que medidores de energia de diferentes fontes se comuniquem perfeitamente.

Posteriormente, reconhecendo a necessidade de uma estrutura mais ampla e detalhada para enfrentar os desafios específicos da medição de energia e informações relacionadas, as extensões do COSEM foram incorporadas ao protocolo. Estas extensões trazem o conceito de modelos de objeto e serviço para o primeiro plano, introduzindo um conjunto abrangente de conceitos e definições que cobrem todos os aspectos relacionados à medição de energia. Isso inclui definições de objetos de medição, parâmetros, eventos, históricos e serviços que podem ser usados para coletar, armazenar e recuperar informações específicas. Em suma, o desenvolvimento do protocolo DLMS/COSEM passou por duas etapas complementares, resultando em um padrão de comunicação completo e comum em sistemas avançados de medição.

Dessa forma, o referido protocolo está conforme os padrões internacionais, garantindo sua robustez, abrangência, interoperabilidade e aceitação global. Um desses padrões é o IEC 62056 (anteriormente conhecido como IEC 61107), amplamente utilizado para comunicação entre medidores de energia e sistemas de leitura remota. Este padrão estabelece uma estrutura unificada de comunicação, abrangendo aspectos como formatos de dados, protocolos de transmissão e requisitos gerais para troca de dados.

A relação entre o protocolo DLMS/COSEM e o padrão IEC 62056 é direta, já que o DLMS/COSEM é uma das principais especificações adotadas no contexto do IEC 62056. O DLMS/COSEM atende aos requisitos definidos no padrão, oferecendo uma estrutura de comunicação interoperável que facilita a troca de dados segura e confiável entre dispositivos de medição e sistemas de leitura. A conformidade com o IEC 62056 reforça o uso do DLMS/COSEM como uma solução de referência para implementações globais, garantindo sua adoção em diferentes contextos e regiões.

Além disso, este protocolo incorpora conceitos da norma IEC 61334, que tem como foco a comunicação eficiente em redes de distribuição de energia. A norma define princípios

e diretrizes para troca de informações entre dispositivos em um sistema de gerenciamento de energia, abrangendo questões relacionadas à transferência de dados em tempo real, resiliência de rede e integração de diferentes dispositivos.

A convergência desses padrões normativos (especialmente IEC 62056 e IEC 61334) fornece uma base sólida e bem fundamentada para DLMS/COSEM. Isso não apenas garante a interoperabilidade entre dispositivos de diferentes fabricantes, mas também reforça sua capacidade de fornecer comunicações confiáveis e padronizadas em ambientes complexos de medição e controle de energia.

Com base nas extensões que o COSEM incorporou ao protocolo DLMS citadas anteriormente, pode-se acessar as funcionalidades do medidor por meio de instâncias dos objetos.

A especificação DLMS/COSEM segue as etapas da Figura 5 como descrito abaixo:

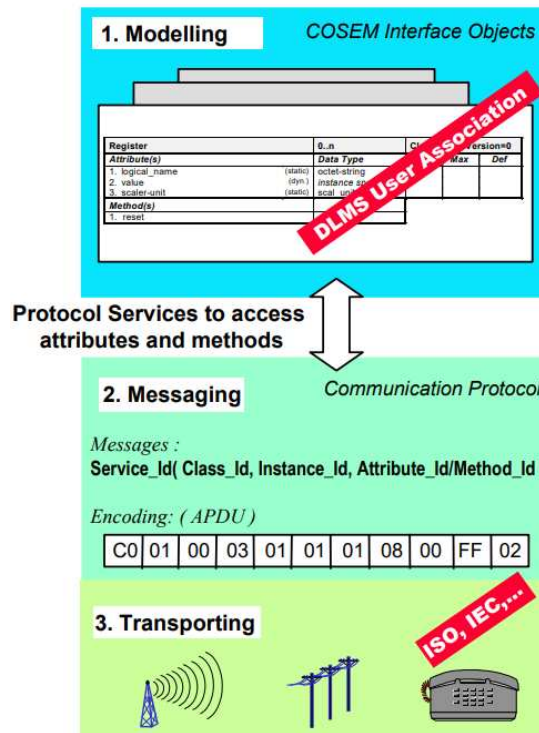
1. Modelagem: esta etapa abrange o modelo de interface do equipamento de medição e as regras de identificação dos dados;
2. Envio de mensagens: esta etapa abrange os serviços de mapeamento do modelo de interface para unidades de dados de protocolo também chamada de Application Protocol Data Unit (APDU);
3. Transporte: abrange o transporte das mensagens através do canal de comunicação.

A etapa de Modelagem é descrita no livro azul DLMS User Association (2020a) e as etapas de Envio de Mensagens e Transporte são descritas no livro verde DLMS User Association (2020b).

Para exemplificar melhor a etapa de Modelamento, o protocolo define identificações chamadas OBIS. Cada identificação do objeto possui 6 parâmetros definidos nos seguintes grupos: A, B, C, D, E e F. Para cada grupo, tem-se a definição na Tabela 1.

As possibilidades de instanciação são organizadas por classes e todos os objetos pertencentes a mesma classe possuem os mesmos atributos e métodos. Cada classe tem um *class_id* usado para referência, a classe utilizada neste estudo, chamada de *Image Transfer*, para atualizar a imagem do medidor é representada pelo *class_id* = 18. Os atributos são representações de características de um objeto, sendo o primeiro atributo sempre o *logical_name*. Este atributo representa o código de identificação OBIS, que no caso da classe *Image Transfer* tem valor: A = 0, B = 0, C = 44, D = 0, E = 0 e F = 255. Em relação aos métodos, são oferecidos com finalidade de realizar procedimentos responsáveis por modificar ou não os atributos do objeto instanciado.

Figura 5 – Abordagem do COSEM: Modelagem, Envio de Mensagens e Transporte



Fonte: DLMS User Association (2020b).

Tabela 1 – Estrutura dos códigos OBIS e definição dos grupos

Grupo	Utilização dos valores
A	Identifica o tipo de energia com o qual a medição está relacionada.
B	Identifica o número do canal de medição, ou seja, o número da entrada de um equipamento de medição com várias entradas para a medição de energia do mesmo tipo ou de tipos diferentes (por exemplo, em concentradores de dados, unidades de registo).
C	Identifica elementos de dados abstratos ou físicos relacionados com a fonte de informação, por exemplo, corrente, tensão, potência, volume, temperatura. As definições dependem do valor do grupo A.
D	Identifica tipos, ou o resultado do processamento de grandezas físicas identificadas por valores nos grupos de valores A e C, de acordo com diversos algoritmos específicos. Os algoritmos podem fornecer quantidades de energia e demanda, bem como outras quantidades físicas.
E	Identifica processamento ou classificação adicional de quantidades identificadas por valores nos grupos de valores A a D.
F	Identifica valores históricos de dados, identificados por valores nos grupos de valores A a E, de acordo com diferentes períodos de faturamento.

Fonte: Adaptador de DLMS User Association (2020b).

2.3.1 Segurança do Padrão DLMS/COSEM

A segurança do padrão DLMS/COSEM é definida nos livros verde e azul da documentação (Green Book e Blue Book). A documentação é extensa, complexa e de custo elevado

haja vista que não é disponibilizada gratuitamente. Entretanto, pode-se ter mais informações sobre a documentação no endereço DLMS User Association (2024). Apesar disso, essa subseção tem papel de explicar as noções básicas de segurança do padrão.

Existem dois tipos de segurança: segurança de acesso e segurança de transporte. A segurança de acesso aborda as permissões concedidas a um usuário para acessar as informações armazenadas em um dispositivo de medição de energia. Por outro lado, a segurança de transporte diz respeito à criptografia aplicada às informações trocadas entre o servidor e o cliente.

No padrão COSEM, a segurança de acesso é estabelecida por meio da troca de requisições de associação (*AssociationRequest*) e respostas de associação (*AssociationResponse*). A identificação do cliente e do servidor é feita mutuamente por seus endereços, e, por conta disso, negociam um contexto de autenticação. No pacote de *AssociationRequest* é passado o tipo de mecanismo de segurança, chamado de *MechanismName*, ele especifica o nível de segurança que o cliente utiliza para acessar o servidor. Existem três opções possíveis para o tipo de mecanismo:

1. Sem segurança: esse nível de mecanismo é caracterizado por não existir nenhum tipo de negociação entre cliente e servidor no que diz respeito a parâmetros de segurança, porém o conjunto de serviços disponíveis nesse modo de associação é limitado pelo servidor;
2. Segurança de nível baixo: nesse caso, o cliente precisa fornecer uma senha ao servidor e corresponder com a senha de nível baixo armazenada por ele. Por conta disso, na associação de nível baixo, o servidor disponibiliza uma quantidade um pouco maior de serviços para uso;
3. Segurança de nível alto: o cliente e o servidor devem se identificar usando um processo composto por 4 etapas. Dessa forma, como meio de comunicação mais confiável, o servidor disponibiliza um número significativamente maior de serviços.

2.3.1.1 Sem Segurança

Para exemplificar melhor um pacote de associação do padrão DLMS/COSEM, podemos representar o pacote com a linguagem de marcação Extensible Markup Language (XML) que fornece regras para definir quaisquer dados. A principal característica dessa linguagem é facilitar o compartilhamento de dados. Nessa linguagem utilizam-se símbolos de marcação, chamados de etiquetas, para definir dados. Por exemplo, para exemplificar uma *AssociationRequest*

pode-se criar uma etiqueta <AssociationRequest> e subetiquetas dentro dessa etiqueta principal representando outros parâmetros da *AssociationRequest*. Exemplo, temos o seguinte pacote de requisição da associação:

Código-fonte 1 – Exemplo de pacote XML de requisição da associação sem segurança

```

1 <AssociationRequest>
2   <ApplicationContextName Value="LN" />
3   <InitiateRequest>
4     <ProposedDlmsVersionNumber Value="06" />
5     <ProposedConformance>
6       <ConformanceBit Name="Action" />
7       <ConformanceBit Name="Set" />
8       <ConformanceBit Name="Get" />
9     </ProposedConformance>
10    <ProposedMaxPduSize Value="FFFF" />
11  </InitiateRequest>
12 </AssociationRequest>

```

Nesse pacote de requisição do DLMS/COSEM, pode-se perceber que existem os seguintes parâmetros: nome do contexto da aplicação(*ApplicationContextName*) com valor "LN" e início da requisição com algumas subetiquetas componentes.

O nome do contexto da aplicação é um elemento utilizado para identificar a aplicação ou contexto dentro do qual as mensagens são trocadas entre dispositivos. Ele desempenha um papel importante na comunicação entre dispositivos DLMS, pois permite que os participantes identifiquem o tipo de aplicação ou serviço que está sendo solicitado ou fornecido.

A *ApplicationContextName* é uma sequência de bytes que representa um identificador único para um determinado contexto de aplicação. Cada aplicação ou serviço no DLMS pode ter seu próprio *ApplicationContextName* para distinguir-se de outras aplicações. Isso é fundamental para garantir a interoperabilidade entre diferentes dispositivos DLMS que podem oferecer ou solicitar diferentes serviços.

No caso desse pacote de requisição da associação, "LN" é uma abreviação de "Logical Name" (Nome Lógico). O *ApplicationContextName* = "LN" é usado para indicar que o contexto

de aplicação está associado a um nome lógico específico

Continuando na explicação do XML mostrado, a primeira subetiqueta do *InitiateRequest* contém a versão do DLMS proposta, já na subetiqueta seguinte também contém componentes e é definida como os tipos de requisições propostas para a associação. No padrão DLMS, uma requisição do tipo *Action* significa acesso a um método de um objeto, as requisições *Get* e *Set* significam leitura e modificação de atributo do objeto respectivamente.

No contexto do DLMS/COSEM, Protocol Data Unit (PDU) é uma unidade de dados que facilita a troca de informações entre dispositivos durante a comunicação. Essa unidade de dados é composta por um cabeçalho, que fornece informações sobre o tipo de mensagem e seus parâmetros, e uma carga útil de dados, que contém os próprios dados a serem transmitidos. Essas unidades de dados são utilizadas para transmitir comandos, solicitações e respostas entre os dispositivos cliente e servidor.

Dessa forma, no que diz respeito ao componente *ProposedMaxPduSize* da *AssociationRequest*, ele se refere ao tamanho máximo proposto para as PDU durante a comunicação entre o cliente e o servidor. Durante o processo de inicialização da comunicação, o cliente propõe um tamanho máximo para o conjunto de PDU que é trocado entre os dispositivos. Essa proposta é especificada no cabeçalho da *AssociationRequest* e é ajustada conforme as capacidades e restrições dos dispositivos envolvidos. O *ProposedMaxPduSize* é fundamental para garantir uma comunicação eficiente e livre de erros, permitindo que os dispositivos coordenem o tamanho das PDUs de maneira apropriada para a transmissão de dados.

Conforme o pacote XML apresentado, percebe-se a ausência do parâmetro que define o tipo de mecanismo de segurança (*MechanismName*). Portanto, nesse tipo de associação em que o mecanismo não é declarado, o mecanismo é definido como sem segurança. Entretanto, nesse tipo de associação pública só é possível ler apenas um subconjunto de dados, caso haja necessidade de acesso a mais informação pode-se estabelecer uma associação usando a segurança de nível baixo, também conhecida como *Low Security*.

2.3.1.2 Segurança de Nível Baixo

No tipo de associação com nível de segurança baixo, o cliente precisa fornecer uma senha chamada *CallingAuthentication*. O pacote XML abaixo representa esse tipo de associação. Código-fonte 2 – Exemplo de pacote XML de requisição da associação com segurança de nível baixo

```

1 <AssociationRequest>
2   <ApplicationContextName Value="LN" />
3   <SenderACSERequirements Value="1" />
4   <MechanismName Value="Low" />
5   <CallingAuthentication Value="3030303030303030" />
6   <InitiateRequest>
7     <ProposedDlmsVersionNumber Value="06" />
8     <ProposedConformance>
9       <ConformanceBit Name="Action" />
10      <ConformanceBit Name="SelectiveAccess" />
11      <ConformanceBit Name="Set" />
12      <ConformanceBit Name="Get" />
13      <ConformanceBit Name="BlockTransferWithGetOrRead" />
14      <ConformanceBit Name="Attribute0SupportedWithGet" />
15    </ProposedConformance>
16    <ProposedMaxPduSize Value="FFFF" />
17  </InitiateRequest>
18 </AssociationRequest>

```

O pacote XML de associação com segurança de nível baixo difere um pouco do pacote apresentado para associação sem segurança. A primeira diferença diz respeito à sub-etiqueta *SenderACSERequirements* que define os requisitos do Association Control Service Element (ACSE) para o remetente. O valor "1" indica que o remetente requer a autenticação no nível de aplicação durante o processo de associação. Além disso, o tipo de mecanismo agora é definido e tem valor *Low*, representando a segurança de nível baixo. Juntamente a esse tipo de mecanismo, também é definido a senha fornecida pelo cliente com valor "3030303030303030" no formato hexadecimal ou "00000000" no formato de texto.

Como já vimos anteriormente, os campos de conformidade do dispositivo indicam certas funcionalidades e recursos do DLMS. No exemplo dado, os bits "*SelectiveAccess*", "*BlockTransferWithGetOrRead*" e "*Attribute0SupportedWithGet*" indicam que o dispositivo suporta funcionalidades específicas, como acesso seletivo, transferência de bloco com leitura ou obtenção de atributo com suporte para o atributo 0.

2.3.1.3 Segurança de Nível Alto

Agora que já conhecemos a associação que utiliza o mecanismo sem segurança e com segurança de baixo nível iremos partir para último mecanismo genérico: segurança de nível alto, também conhecido como High Level Security (HLS). Há vários tipos de mecanismos específicos para segurança de alto nível. Este trabalho utiliza-se o mecanismo específico *HighGMac* no qual é baseado no uso do Galois Message Authentication Code (GMAC) como método de autenticação de mensagens. GMAC é uma variante do GCM que fornece autenticação de integridade de mensagens, mas não criptografa os dados, porém na implementação feita neste trabalho há a presença da criptografia. O uso de *HighGMac* no DLMS/COSEM combina esse mecanismo de autenticação com outros elementos para fornecer segurança robusta nas comunicações entre o cliente e o servidor.

Para que a abertura da associação seja realizada com sucesso neste nível de segurança, é necessário seguir o seguinte procedimento:

1. **Envio do *InitiateRequest***: o processo de abertura de uma associação segura começa com o envio de uma mensagem de *InitiateRequest* pelo dispositivo cliente. Esta mensagem tem o objetivo de iniciar a sessão de comunicação segura e inclui informações como a versão do protocolo, parâmetros de timeout e as capacidades do cliente. Além disso, nesta fase, o cliente especifica o tipo de segurança utilizado, no caso, o *HighGMac*, que é um mecanismo de segurança que utiliza o algoritmo Advanced Encryption Standard Galois/Counter Mode (AES-GCM);
2. **Recebimento do *InitiateResponse***: após o envio do pacote inicial, o servidor responde com um pacote contendo o nível de conformidade suportado. Esse campo de conformidade define as funcionalidades e os serviços que o medidor suporta e que serão utilizados na associação. Por exemplo, o campo de conformidade pode definir que o medidor suporta a leitura e gravação de objetos, a transferência de arquivos, entre outras funcionalidades. Essa resposta permite ao cliente adaptar sua estratégia de comunicação para garantir que ambas as partes compartilhem o mesmo nível de segurança e conformidade;
3. **Envio do desafio**: Após a troca inicial de informações de conformidade, o cliente gera um valor de desafio, utilizando o valor gerado pela função de autenticação fornecido pelo servidor no campo *Response-Authentication* do primeiro pacote de resposta do servidor. Este valor de desafio é então enviado para o servidor dentro de uma mensagem criptografada de autenticação de resposta;

4. **Recebimento do desafio:** Após receber o desafio do cliente, o servidor valida o valor fornecido. Se o desafio do cliente for aceito, o servidor responde gerando um novo valor de desafio baseado na função de autenticação do cliente, que o cliente forneceu no campo *CallingAuthentication* do primeiro pacote de envio. Este segundo desafio, gerado pelo servidor, é enviado de volta ao cliente para autenticar o servidor, fechando assim o ciclo de autenticação mútua. A partir deste ponto, a associação é considerada segura, e as mensagens subsequentes podem ser trocadas de maneira criptografada utilizando HighGMac, assegurando a integridade e a confidencialidade das comunicações.

2.3.1.3.1 Envio do InitiateRequest

Para exemplificar o envio do *InitiateRequest*, o pacote XML abaixo representa a associação com segurança de nível alto utilizando o mecanismo *HighGMac*:

Código-fonte 3 – Pacote XML de InitiateRequest criptografado

```

1 <AssociationRequest>
2   <ApplicationContextName Value="LN_WITH_CIPHERING" />
3   <CallingAPTtitle Value="4155580000000000" />
4   <SenderACSERrequirements Value="1" />
5   <MechanismName Value="HighGMac" />
6   <CallingAuthentication Value="3342786B33385070" />
7   <glo_InitiateRequest Value="200000001
      A14969B6FC7A0030BC9C65AFF2EF4" />
8 </AssociationRequest>

```

Percebe-se no primeiro momento que a etiqueta *InitiateRequest* foi removida e substituída pela etiqueta *glo_InitiateRequest*. O termo *glo* no campo indica que os dados estão cifrados globalmente para garantir a confidencialidade, ou seja, agora nosso pacote que negocia os tipos de comandos que o medidor irá aceitar dentro dessa abertura de associação está criptografado, o que aumenta significativamente a segurança da comunicação. Em um cenário real, o conteúdo desse campo pode ser decodificado para revelar os detalhes específicos da mensagem de solicitação inicial.

Acerca do pacote XML mostrado, visualiza-se primeiramente o campo *Application-*

ContextName que indica o contexto de aplicação que está sendo utilizado para a comunicação entre o cliente e o servidor. No caso do valor **LN_WITH_CIPHERING**, ele especifica que o contexto de aplicação se refere à comunicação baseada em **Logical Name**(LN) e que haverá cifragem (criptografia) nas mensagens trocadas. Essa cifragem adiciona uma camada de segurança à comunicação, garantindo que os dados transmitidos estejam protegidos contra interceptações não autorizadas.

Logo após, o campo *CallingAPTtitle* representa o título da aplicação chamadora (ou seja, o cliente). Este campo é utilizado para identificar de forma única o cliente que está iniciando a associação. No exemplo fornecido, o valor **4155580000000000** é o identificador específico do cliente. Ele pode ser um número único ou um endereço associado ao dispositivo cliente.

O campo *MechanismName* especifica o mecanismo de segurança que está sendo utilizado na associação. Como dito anteriormente, o mecanismo utilizado é o *HighGMac*.

No campo *CallingAuthentication*, contém o valor de autenticação gerado pelo cliente para validar sua identidade ao servidor.

Por último, o campo *glo_InitiateRequest* contém a solicitação inicial cifrada que o cliente envia ao servidor. O valor **200000001A14969B6FC7A0030BC9C65AFF2EF4** representa a mensagem criptografada, que inclui informações sobre a versão do protocolo DLMS, conformidade proposta, tamanho máximo da PDU e outros parâmetros necessários para iniciar a comunicação.

Para descriptografar o campo *glo_InitiateRequest*, é necessário conhecer o processo do *AES-GCM*. No contexto do protocolo DLMS/COSEM, o *AES-GCM* é amplamente utilizado para garantir a segurança das comunicações. O *AES* é um algoritmo de criptografia de bloco, enquanto o *GCM* é um modo de operação que adiciona tanto confidencialidade quanto autenticação às mensagens. No caso deste trabalho, o *GCM* pode ser utilizado sem a parte de autenticação, para realizar apenas a criptografia e a descriptografia dos dados.

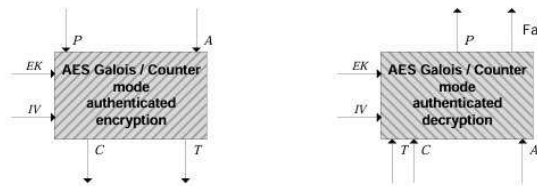
Acerca do *AES*, é um algoritmo de criptografia simétrica, ou seja, ele usa a mesma chave tanto para criptografar quanto para descriptografar os dados. Ele opera em blocos de 128 bits (16 bytes) e utiliza chaves que podem ter tamanhos de 128, 192 ou 256 bits. O processo de criptografia consiste em uma série de transformações que incluem substituição de bytes, permutação de bits e combinação de blocos com a chave de criptografia. Cada etapa é projetada para confundir e difundir os dados, tornando-os irreconhecíveis.

O *GCM* é um modo de operação de blocos do *AES* que combina criptografia. O *GCM*

opera como um contador (CTR), que transforma a *AES* em um cifrador de fluxo, permitindo a criptografia e a descryptografia de blocos de dados de tamanho variável.

O modo GCM é descrito como mais detalhes no livro verde DLMS User Association (2020b). A Figura 6 ilustra as funções de encriptação e descryptação do algoritmo juntamente aos seus parâmetros de entrada e saída.

Figura 6 – Representação da encriptação e descryptação do modo GCM



Fonte: DLMS User Association (2020b).

No GCM, a criptografia é feita por um contador incrementado a cada bloco de dados. Cada bloco é cifrado utilizando uma combinação da chave de criptografia, chamada Encryption Key (EK), e de um vetor de inicialização *IV*. O resultado é o *ciphertext* representado por *C* na Figura 6, que é o dado criptografado.

A descryptografia funciona de maneira inversa, onde o *ciphertext* é processado para reverter o conteúdo ao seu estado original, utilizando a mesma chave e *IV*. Utiliza-se o dado de autenticação representado por *A* na Figura 6 somente na função do lado direito da figura a qual pode-se utilizar para descryptografar ou autenticar valores. Não se utiliza a *tag* de autenticação *T* em ambas funções neste trabalho.

Para exemplificar, foi utilizado a ferramenta **SCV Cryptomanager** da Figura 7 na descryptografia das mensagens.

Para descryptografar o *glo_InitiateRequest*, primeiramente vamos identificar cada parâmetro do AES-GCM:

- a) **IV**: é o valor do *CallingAPTitle* seguido do contador do cliente. O contador do cliente começa no segundo *byte* do *glo_InitiateRequest* e contém ao todo 4 *bytes*. Dessa forma, o valor desse parâmetro fica "**415558000000000000000000000000001A**";
- b) **EK**: nesse caso iremos utilizar uma chave de encriptação fictícia para não compartilhar a chave privada do medidor utilizado. Nesse caso, o valor da chave escolhido foi "**00000000000000000000000000000000**";
- c) **C**: o valor que desejamos descryptografar corresponde aos valores depois do

Figura 7 – Ferramenta SCV Cryptomanager

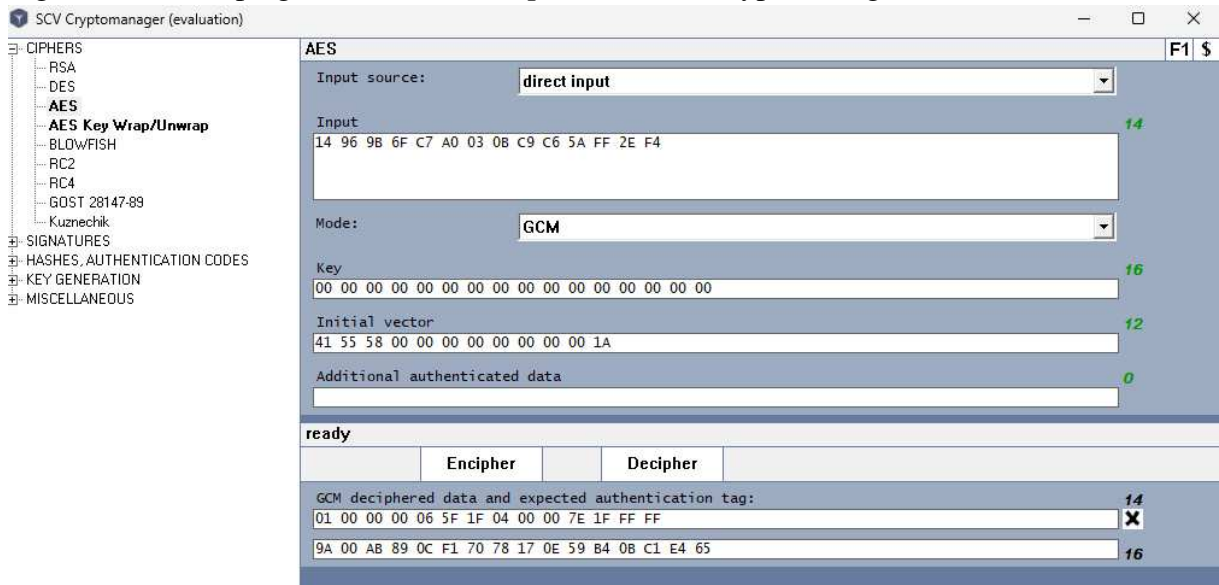


Fonte: Elaborado pelo autor.

contador do cliente no campo *glo_InitiateRequest*. Então, o valor de entrada é igual a **"14969B6FC7A0030BC9C65AFF2EF4"**;

A Figura 8 contém o resultado da descryptografia. O resultado final do processo **"0100000065F1F0400007E1FFFFF"** corresponde a um pacote DLMS válido.

Figura 8 – Descryptografia do *InitiateRequest* no SCV Cryptomanager



Fonte: Elaborado pelo autor.

A seguir segue o XML do pacote de iniciação da associação descryptografado o qual requisita negociar alguns comandos com o medidor:

Código-fonte 4 – Pacote de InitiateRequest descriptografado

```

1 <InitiateRequest>
2   <ProposedDlmsVersionNumber Value="06" />
3   <ProposedConformance>
4     <ConformanceBit Name="Action" />
5     <ConformanceBit Name="EventNotification" />
6     <ConformanceBit Name="SelectiveAccess" />
7     <ConformanceBit Name="Set" />
8     <ConformanceBit Name="Get" />
9     <ConformanceBit Name="MultipleReferences" />
10    <ConformanceBit Name="BlockTransferWithAction" />
11    <ConformanceBit Name="BlockTransferWithSetOrWrite" />
12    <ConformanceBit Name="BlockTransferWithGetOrRead" />
13    <ConformanceBit Name="Attribute0SupportedWithGet" />
14    <ConformanceBit Name="PriorityMgmtSupported" />
15  </ProposedConformance>
16  <ProposedMaxPduSize Value="FFFF" />
17 </InitiateRequest>

```

2.3.1.3.2 Recebimento do InitiateResponse

Na etapa posterior ao envio do primeiro pacote, o servidor envia o pacote de resposta que contém os parâmetros do medidor e também a resposta do *InitiateRequest*. Segue o XML do *InitiateResponse*:

Código-fonte 5 – Pacote XML de *InitiateResponse* criptografado

```

1 <AssociationResponse>
2   <ApplicationContextName Value="LN_WITH_CIPHERING" />
3   <AssociationResult Value="0" />
4   <ResultSourceDiagnostic>
5     <ACSEServiceUser Value="0" />

```

```

6   </ResultSourceDiagnostic>
7   <RespondingAPTitle Value="41555867720ABC00" />
8   <ResponderACSERequirement Value="1" />
9   <MechanismName Value="HighGMac" />
10  <RespondingAuthentication Value="F72E5014ACF2BC03" />
11  <glo_InitiateResponse Value="2000009746
    D63AABC10C4BC08F20652B9AE989" />
12 </AssociationResponse>

```

Para descriptografar o *glo_InitiateResponse*, é preciso definir novamente os seguintes parâmetros do AES-GCM:

- a) **IV**: é o valor do *RespondingAPTitle* seguido do contador do servidor. O contador do servidor começa no segundo *byte* do *glo_InitiateResponse* e contém ao todo 4 *bytes*. Dessa forma, o valor desse parâmetro fica "**41555867720ABC0000009746**";
- b) **EK**: a chave de encriptação compartilhada entre cliente e servidor definida continua sendo "**00000000000000000000000000000000**";
- c) **C**: o valor que desejamos descriptografar corresponde aos valores depois do contador do cliente no campo *glo_InitiateResponse*. Então, o valor de entrada tem valor "**D63AABC10C4BC08F20652B9AE989**";

A Figura 9 contém o resultado da descriptografia. O resultado final do processo "**0800065F1F040000181D00D00007**" corresponde a um pacote DLMS válido.

A seguir segue o XML da resposta do pacote de iniciação da associação descriptografado:

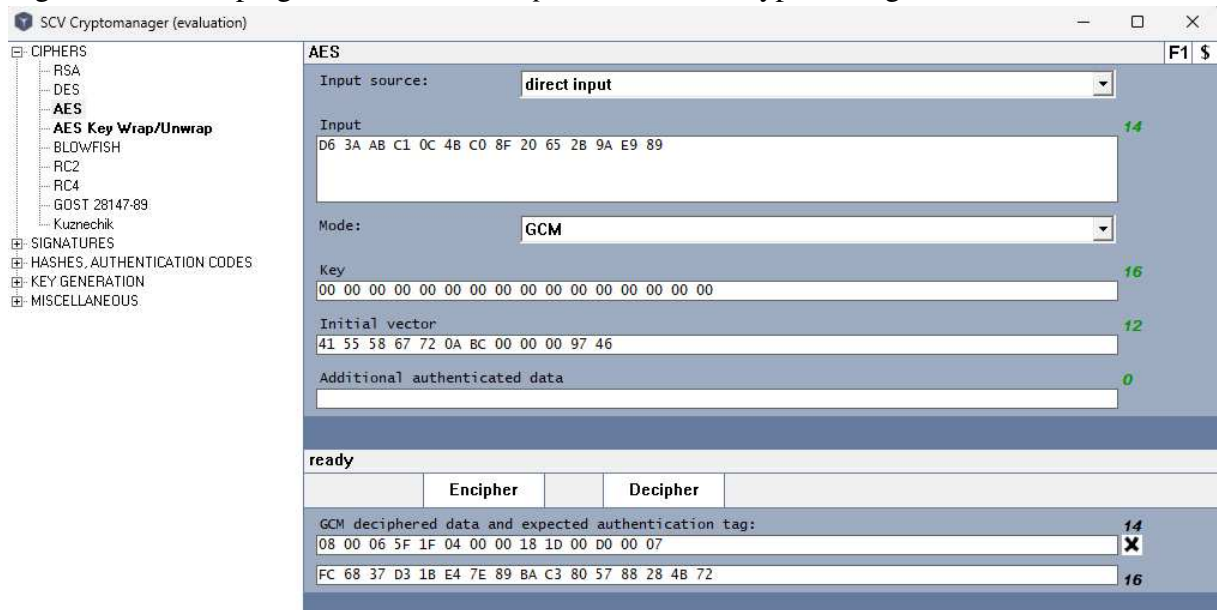
Código-fonte 6 – Pacote de *InitiateResponse* descriptografado

```

1 <InitiateResponse>
2   <NegotiatedDlmsVersionNumber Value="06" />
3   <NegotiatedConformance>
4     <ConformanceBit Name="Action" />
5     <ConformanceBit Name="SelectiveAccess" />
6     <ConformanceBit Name="Set" />
7     <ConformanceBit Name="Get" />

```

Figura 9 – Descritografia do *InitiateResponse* no SCV Cryptomanager



Fonte: Elaborado pelo autor.

```

8     <ConformanceBit Name="BlockTransferWithSetOrWrite" />
9     <ConformanceBit Name="BlockTransferWithGetOrRead" />
10    </NegotiatedConformance>
11    <NegotiatedMaxPduSize Value="00D0" />
12    <VaaName Value="0007" />
13 </InitiateResponse>

```

Percebe-se nessa resposta os comandos no qual o medidor aceitou receber dentro dessa abertura de associação e o tamanho máximo da PDU aceito.

2.3.1.3.3 Envio do Desafio do cliente

Na etapa posterior ao recebimento de resposta do primeiro pacote, o cliente envia o pacote criptografado contendo um desafio para o servidor autenticar a comunicação. Dessa forma, segue o seguinte XML com envio do desafio:

Código-fonte 7 – Pacote de *ActionRequest* criptografado

```

1 <PDU>
2 <glo_ActionRequest Value="200000001C47A12F1A9AB6934CC
3     218C8D47538057B6F9F6AEF628B

```

```

4                                DOBEFF5FF0B3F6E0AA2F " />
5 </PDU>

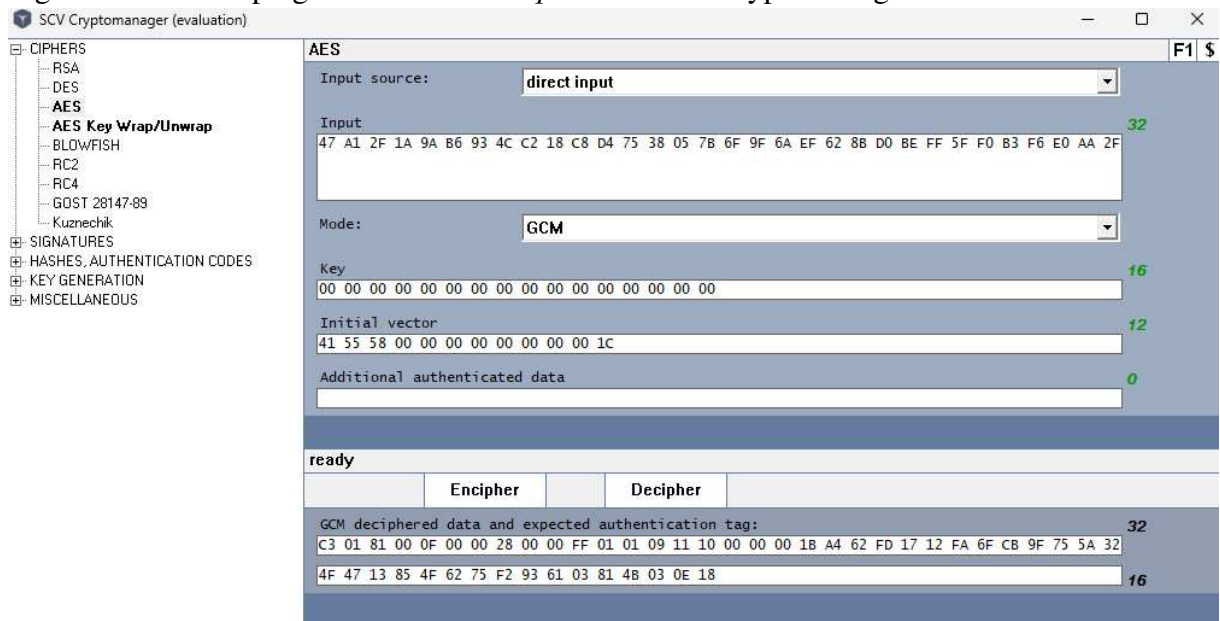
```

Para descriptografar o *glo_ActionRequest*, é preciso definir novamente os seguintes parâmetros do AES-GCM:

- a) **IV**: é o valor do *CallingAPTtitle* no Código-fonte 3 seguido do contador do cliente. O contador do cliente começa no segundo *byte* do *glo_ActionRequest* e contém ao todo 4 *bytes*. Dessa forma, o valor final desse parâmetro de entrada é igual a **"41555800000000000000000000001C"**;
- b) **EK**: a chave de encriptação compartilhada entre cliente e servidor definida continua sendo **"00000000000000000000000000000000"**;
- c) **C**: o valor que desejamos descriptografar corresponde aos valores depois do contador do cliente no campo *glo_ActionRequest*. Então, o valor de entrada tem valor **"47A12F1A9AB6934CC218C8D47538057B6F9F6AEF628BD0BEFF5FF0B3F6E0AA2F"**;

A Figura 10 contém o resultado da descriptografia. O resultado final do processo **"C3018100F000028000FF01010911100000001BA462FD1712FA6FCB9F755A32"** corresponde a um pacote DLMS válido.

Figura 10 – Descriptografia do *ActionRequest* no SCV Cryptomanager



Fonte: Elaborado pelo autor.

A seguir segue o XML do envio do pacote de desafio da associação descriptografado:

Código-fonte 8 – Pacote de ActionRequest descriptografado

```

1 <ActionRequest>
2   <ActionRequestNormal>
3     <!--Priority: HIGH ServiceClass: UN_CONFIRMED invokeID:
4       1-->
5     <InvokeIdAndPriority Value="81" />
6     <MethodDescriptor>
7       <!--ASSOCIATION_LOGICAL_NAME-->
8       <ClassId Value="000F" />
9       <!--0.0.40.0.0.255-->
10      <InstanceId Value="0000280000FF" />
11      <MethodId Value="01" />
12    </MethodDescriptor>
13    <MethodInvocationParameters>
14      <OctetString Value="100000001B
15        A462FD1712FA6FCB9F755A32" />
16    </MethodInvocationParameters>
17  </ActionRequestNormal>
18 </ActionRequest>

```

Percebe-se nessa requisição que o dispositivo cliente envia um *ActionRequest* para o objeto *Association Logical Name* do servidor com finalidade de autenticar a comunicação. O parâmetro do método invocado tem valor gerado pelo cliente com base na chave Authentication Key (AK) e no campo *RespondingAuthentication* respondido pelo servidor no Código-fonte 5. O valor desse parâmetro no pacote é igual a **"100000001BA462FD1712FA6FCB9F755A32"**. O primeiro *byte* com valor igual a **"10"** significa que o parâmetro deverá ser entendido pelo servidor como um parâmetro de autenticação. Os próximos 4 *bytes* com valor igual a **"0000001B"** representa o contador do cliente antes do envio da mensagem do Código-fonte 7. Por último, o resto do parâmetro que tem valor igual a **"A462FD1712FA6FCB9F755A32"** indica o valor a ser autenticado pelo servidor. Nessa etapa, é utilizada a chave AK compartilhada entre o cliente e servidor para autenticação.

Para chegar a esse valor de autenticação o cliente montou esse parâmetro definindo

os seguintes parâmetros do AES-GCM:

- a) **IV**: é o valor do *CallingAPTtitle* no Código-fonte 3 seguido do contador do cliente antes do envio do pacote presente no Código-fonte 7. O contador do cliente começa no segundo *byte* do *OctetString* dentro da etiqueta *MethodInvocationParameters* e contém ao todo 4 *bytes*. Dessa forma, o valor final desse parâmetro de entrada é igual a "**41555800000000000000000000000001B**";
- b) **EK**: a chave de encriptação compartilhada entre cliente e servidor definida continua sendo "**00**";
- c) **AK**: a chave de autenticação compartilhada entre cliente e servidor também é uma chave fictícia, pois não é permitido compartilhar a chave privada do medidor de energia. Portanto, foi utilizado nessa demonstração a AK com valor igual a "**000102030405060708090A0B0C0D0E0F**";
- d) **C**: o parâmetro usado para definir o bloco cifrado não é necessário pois nessa etapa é feito somente a autenticação;
- e) **A**: o parâmetro de dado adicional de autenticação é utilizado nessa etapa. O valor desse parâmetro segue o seguinte padrão: primeiro **byte** igual a "**10**" indicando autenticação seguido da chave AK e do valor do campo *RespondingAuthentication* do Código-fonte 5. Então, o resultado final desse parâmetro fica igual a "**10000102030405060708090A0B0C0D0E0FF72E5014ACF2BC03**";

A Figura 11 contém o resultado da autenticação. O resultado do processo tem valor igual a "**A462FD1712FA6FCB9F755A325E7433B2**". Entretanto, o livro verde DLMS User Association (2020b) descreve que se utiliza somente os primeiros 12 *bytes*. Dessa maneira, o valor final torna-se "**A462FD1712FA6FCB9F755A32**" o qual bate com o valor do Código-fonte 8.

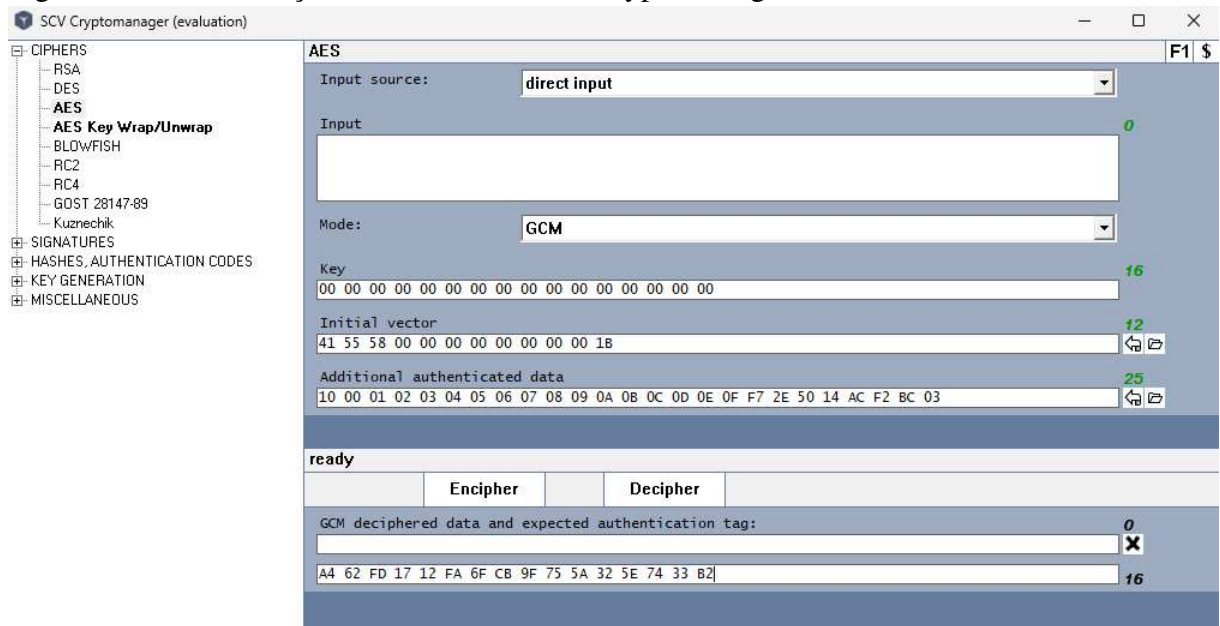
2.3.1.3.4 Envio do Desafio do servidor

Na etapa posterior ao envio do desafio pelo cliente, o servidor também envia o desafio criptografado para autenticar a comunicação. Dessa forma, segue o seguinte XML com pacote do desafio criptografado:

Código-fonte 9 – Pacote de ActionResponse criptografado

1 | <PDU>

Figura 11 – Autenticação do desafio no SCV Cryptomanager



Fonte: Elaborado pelo autor.

```

2 <glo_ActionResponse Value="2000009748BE830D5819A5E1C
3                               BBE82ED165262B875D49D6306
4                               846DDDA065 " />
5 </PDU>

```

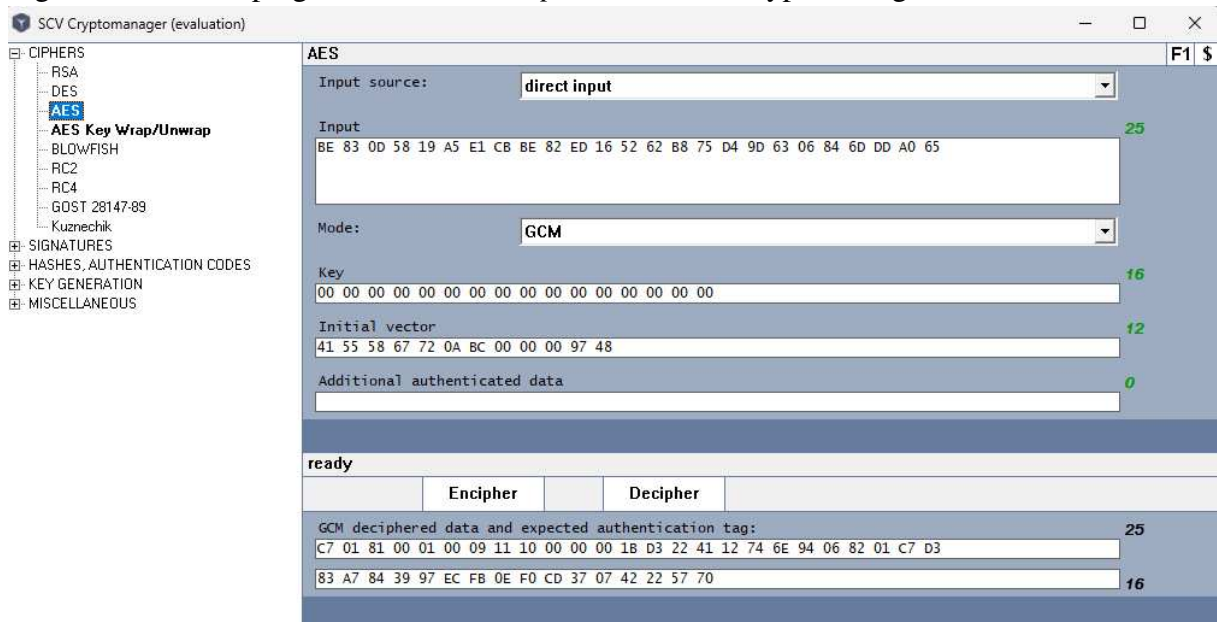
Para descriptografar o *glo_ActionResponse*, é preciso definir novamente os seguintes parâmetros do AES-GCM:

- IV:** é o valor do *RespondingAPTitle* seguido do contador do servidor. O contador do servidor começa no segundo *byte* do *glo_ActionResponse* e contém ao todo 4 *bytes*. Dessa forma, o valor desse parâmetro fica "**41555867720ABC0000009748**";
- EK:** a chave de encriptação compartilhada entre cliente e servidor definida continua sendo "**00000000000000000000000000000000**";
- C:** o valor que desejamos descriptografar corresponde aos valores depois do contador do servidor no campo *glo_ActionResponse*. Então, o valor de entrada é igual a "**BE830D5819A5E1CBBE82ED165262B875D49D6306846DDDA065**";

A Figura 12 contém o resultado da descriptografia. O resultado final do processo "**C701810001000911100000001BD3224112746E94068201C7D3**" corresponde a um pacote DLMS válido.

A seguir segue o XML do pacote de desafio da associação descriptografado:

Figura 12 – Descriptografia do *ActionResponse* no SCV Cryptomanager



Fonte: Elaborado pelo autor.

Código-fonte 10 – Pacote de *ActionResponse* descriptografado

```

1 <ActionResponse>
2   <ActionResponseNormal>
3     <!--Priority: HIGH ServiceClass: UN_CONFIRMED invokeID:
4       1-->
5     <InvokeIdAndPriority Value="81" />
6     <Result Value="Success" />
7     <ReturnParameters>
8       <Data>
9         <OctetString Value="100000001B
10           D3224112746E94068201C7D3" />
11       </Data>
12     </ReturnParameters>
13 </ActionResponseNormal>
14 </ActionResponse>

```

Percebe-se nessa requisição que o dispositivo servidor envia um *ActionResponse* para o cliente com resultado da autenticação e também com o desafio que o cliente deverá utilizar para autenticar o servidor. O desafio também é gerado pelo servidor com base na chave AK e

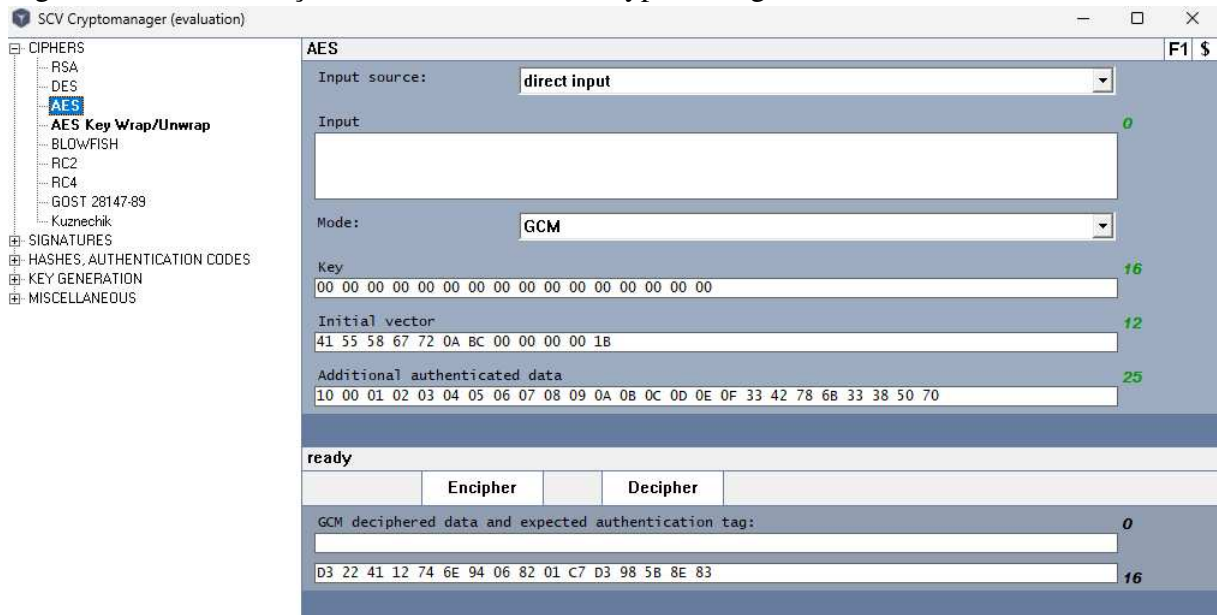
no campo *CallingAuthentication* respondido pelo cliente no Código-fonte 3. Como descrito no pacote, o valor desse parâmetro é igual a "**100000001BD3224112746E94068201C7D3**". O primeiro *byte* com valor igual a "**10**" significa que o parâmetro deverá ser entendido pelo servidor como um parâmetro de autenticação. Os próximos 4 *bytes* com valor igual a "**0000001B**" representa o contador do servidor antes do envio da mensagem do Código-fonte 9. Por último, o resto do parâmetro que tem valor igual a "**D3224112746E94068201C7D3**" indicando o valor a ser autenticado pelo cliente. Nessa etapa, também é utilizada a chave AK compartilhada entre o cliente e servidor para autenticação.

Para chegar a esse valor de autenticação o servidor montou o parâmetro definindo os seguintes parâmetros do AES-GCM:

- a) **IV**: é o valor do *RespondingAPTitle* no Código-fonte 5 seguido do contador do servidor antes do envio do pacote presente no Código-fonte 10. O contador do servidor começa no segundo *byte* do *OctetString* dentro da etiqueta *Data* e contém ao todo 4 *bytes*. Dessa forma, o valor final desse parâmetro de entrada é igual a "**41555867720ABC000000001B**";
- b) **EK**: a chave de encriptação compartilhada entre cliente e servidor definida continua sendo "**00000000000000000000000000000000**";
- c) **AK**: como citado anteriormente, a chave de autenticação compartilhada entre cliente e servidor tem valor "**000102030405060708090A0B0C0D0E0F**";
- d) **C**: o parâmetro usado para definir o bloco cifrado não é necessário pois nessa etapa é feito somente a autenticação;
- e) **A**: o parâmetro de dado adicional de autenticação é utilizado nessa etapa. O valor desse parâmetro segue o seguinte padrão: primeiro *byte* igual a "**10**" indicando autenticação seguido da chave AK e do valor do campo *CallingAuthentication* do Código-fonte 3. Então, o resultado final desse parâmetro fica igual a "**10000102030405060708090A0B0C0D0E0F3342786B33385070**";

A Figura 13 contém o resultado da autenticação. O resultado do processo tem valor igual a "**D3224112746E94068201C7D3985B8E83**". Entretanto, o livro verde DLMS User Association (2020b) descreve que se utiliza somente os primeiros 12 *bytes*. Dessa maneira, o valor final torna-se "**D3224112746E94068201C7D3**" o qual bate com o valor do Código-fonte 10.

Figura 13 – Autenticação do desafio no SCV Cryptomanager



Fonte: Elaborado pelo autor.

2.3.2 Image Transfer

Nesta seção, discutiremos o objeto Image Transfer, um componente essencial no processo de atualização de firmware em medidores de energia inteligentes utilizando o protocolo DLMS/COSEM. Este objeto é responsável por gerenciar a transferência segura de arquivos de imagem de firmware para o medidor, garantindo a integridade e a consistência dos dados durante o processo de atualização. Vamos explorar os atributos e métodos que compõem este objeto, conforme descrito no padrão DLMS, detalhando suas funcionalidades e como são utilizados no contexto prático.

Antes de prosseguir, é importante entender o conceito de objeto. No modelo DLMS/COSEM, um objeto representa uma entidade abstrata que encapsula um conjunto de funcionalidades de um dispositivo, como um medidor de energia. Esses objetos podem ser comparados a classes em linguagens de programação orientadas a objetos, onde eles possuem tanto atributos quanto métodos.

1. **Atributos:** São as propriedades do objeto que contêm dados. Esses dados podem ser lidos ou escritos para monitorar ou modificar o comportamento do dispositivo. No caso do objeto Image Transfer, os atributos podem incluir informações sobre o estado da transferência de imagem, como o progresso atual.
2. **Métodos:** São as operações que podem ser executadas sobre o objeto. Esses métodos permitem ao sistema executar ações específicas, como iniciar uma

transferência de imagem, validar um arquivo de imagem recebido ou aplicar a atualização ao firmware do medidor.

Dessa forma, o objeto Image Transfer encapsula tanto as informações quanto as ações necessárias para conduzir de forma segura e eficiente o processo de atualização de firmware em um medidor de energia. A Tabela 2 está presente no livro DLMS User Association (2020a) define atributos e métodos desse objeto.

Tabela 2 – Interface Class Image Transfer

Image Transfer		0...n		classe_id = 18, versão = 0		
Attributes	Data type	Min.	Max.	Def.	Short name	
1.logical_name	octet-string				x	
2.image_block_size	double-long-unsigned				x + 0x08	
3.image_transferred_blocks_status	bit-string				x + 0x10	
4.image_first_not_transferred_block_number	double-long-unsigned				x + 0x18	
5.image_transfer_enabled	boolean				x + 0x20	
6.image_transfer_status	enum				x + 0x28	
7.image_to_activate_info	array				x + 0x30	
Specific methods	m/o					
1.image_transfer_initiate(data)	mandatory				x + 0x40	
2.image_block_transfer(data)	mandatory				x + 0x48	
3.image_verify(data)	mandatory				x + 0x50	
4.image_activate(data)	mandatory				x + 0x58	

Fonte: adaptado da DLMS User Association (2020a).

2.3.2.1 Descrição dos Atributos

Os atributos apresentados na Tabela 2 são definidos da seguinte forma:

1. **logical_name**: identifica a instância do objeto "Image Transfer", ou seja, tem o valor de OBIS code: 0,0,44,0,e,255;
2. **image_block_size**: contém o *ImageBlockSize*, expresso em octetos, que pode ser manuseado pelo servidor. Entre outras palavras, o *firmware* é dividido em blocos de tamanho igual e esse atributo corresponde a este tamanho. O tamanho do bloco é uma propriedade do servidor. Dessa forma, medidores de energia de fabricantes diferentes podem suportar tamanhos de blocos diferentes;
3. **image_transferred_blocks_status**: fornece informação sobre o estado da transferência de cada *ImageBlock*. Cada bit representa um bloco, dessa forma a sequência de bits fornece informações sobre o processo de transferência de blocos. Exemplo: se o bit número 0, 1 e 2 estão setados em 1 significa que os

blocos 0, 1 e 2 foram transferidos ao medidor corretamente. Caso os bits 3 e 4 estejam setados em 0 significa que o bloco não foi transferido ou foi perdido pelo medidor de energia;

4. **image_first_not_transferred_block_number**: : fornece o *ImageBlockNumber* do primeiro bloco não transferido ou perdido. Uma vez que a Imagem está completa, o valor retornado deve ser igual ou superior ao número de blocos calculados a partir do tamanho da Imagem e do *ImageBlockSize*;
5. **image_transfer_enabled**: controla a ativação do processo de transferência de imagem. Caso o valor desse atributo no medidor seja igual a *false*, o processo de transferência está desabilitado e todos os métodos desse objeto também. Entretanto, caso o valor seja igual a *true* todos os métodos estão habilitados e o processo de transferência pode ocorrer normalmente.
6. **image_transfer_status**: Contém o estado do processo de transferência da imagem com os seguintes valores:
 - a) (0) Image transfer not initiated;
 - b) (1) Image transfer initiated;
 - c) (2) Image verification initiated;
 - d) (3) Image verification successful;
 - e) (4) Image verification failed;
 - f) (5) Image activation initiated;
 - g) (6) Image activation successful;
 - h) (7) Image activation failed.
7. **image_to_activate_info**: fornece informação sobre a Imagem pronta para ativação. Ele é gerado como resultado do processo de verificação de Imagem. O formato desse atributo é definido como uma *structure* formada por 3 parâmetros como demonstrado na Figura 14. Cada parâmetro é definido como:
 - a) **image_to_activate_size** é o tamanho da Imagem a ser ativada, expressa em octetos;
 - b) **image_to_activate_identification** é a identificação da Imagem a ser ativada, e pode conter informações como fabricante, tipo de dispositivo, informações de versão, etc;
 - c) **image_to_activate_signature** é a assinatura da Imagem a ser

ativada.

Figura 14 – Estrutura do atributo *image_to_activate_info*

image_to_activate_info_element::= structure

```
{
```

<i>image_to_activate_size</i> :	double-long-unsigned,
<i>image_to_activate_identification</i> :	octet-string,
<i>image_to_activate_signature</i> :	octet-string

```
}
```

Fonte: DLMS User Association (2020a).

2.3.2.2 Descrição dos Métodos

Os métodos apresentados na Tabela 2 são definidos da seguinte forma:

1. **image_transfer_initiate(data)**: inicializa o processo de Image transfer. O parâmetro de entrada **data** é obrigatório e tem a estrutura da Figura 15, onde o *image_identifier* identifica a Imagem a ser transferida e o *image_size* contém o tamanho da Imagem;
2. **image_block_transfer(data)**: transfere um bloco da Imagem para o servidor. O parâmetro de entrada **data** é obrigatório e tem a estrutura da Figura 16, onde o *image_block_number* é o número do bloco transferido e o *image_block_value* contém o conteúdo do firmware naquele bloco. Depois do sucesso da chamada do método, o bit correspondente no atributo *image_transfer_status* é definido para (1) e o atributo *image_first_not_transferred_block_number* attribute é atualizado;
3. **image_verify(data)** : verifica a integridade da imagem antes da ativação. O parâmetro de entrada **data** é obrigatório e deve ser um inteiro com valor igual a 0. O resultado da chamada deste método pode retornar **sucess**, **temporary_failure** ou **other_reason**. Se ele não for sucesso, então o resultado da verificação pode ser encontrada recuperando o valor do atributo *image_transfer_status*. No caso de sucesso, o atributo *image_to_activate_info* contém a informação sobre a imagem a ser ativada;
4. **image_activate(data)** : ativa a Imagem no servidor. O parâmetro de entrada **data** é obrigatório e deve ser um inteiro com valor igual a 0. Se a imagem transferida não for verificada antes, então será feito como parte do processo de ativação da Imagem.

gem. O resultado da chamada deste método pode ser **success**, **temporary_failure** ou **other_reason**. Se ele não retornar sucesso, então o resultado da ativação pode ser aprendido recuperando o valor do atributo *image_transfer_status*.

Figura 15 – Estrutura do parâmetro de entrada *data* do método **image_transfer_initiate(data)**

```
data ::= structure
{


|                   |                      |
|-------------------|----------------------|
| image_identifier: | octet_string,        |
| image_size:       | double-long-unsigned |


}
```

Fonte: DLMS User Association (2020a).

Figura 16 – Estrutura do parâmetro de entrada *data* do método **image_block_transfer(data)**

```
data ::= structure
{


|                     |                       |
|---------------------|-----------------------|
| image_block_number: | double-long-unsigned, |
| image_block_value:  | octet-string          |


}
```

Fonte: DLMS User Association (2020a).

2.4 Firmware

O firmware é um tipo de software específico que opera no nível mais básico de um dispositivo eletrônico e funciona como uma ponte entre o hardware e os níveis de software adicionais. Diferente do software padrão, o firmware é gravado diretamente em componentes de memória de leitura não volátil, como flash, Read-Only Memory (ROM), Electrically Erasable Programmable Read-Only Memory (EEPROM). Esses componentes permitem a retenção de dados mesmo sem energia. Em resumo, o firmware controla as funções essenciais do dispositivo, permitindo-lhe realizar as operações básicas.

O firmware é extremamente importante para os medidores de energia, que são dispositivos eletrônicos essenciais para leitura, registro e comunicação do uso de eletricidade. Ele supervisiona o processo pelo qual o medidor coleta dados, os processa e envia para a concessionária de energia. Essa função garante medições precisas e comunicação eficiente com os sistemas de gestão da rede elétrica.

O firmware é vital para os medidores de energia por várias razões, incluindo garantir que as medições sejam precisas e proteger os dados transmitidos. Aqui estão alguns pontos importantes:

1. **Precisão nas Medições:** O firmware controla como os medidores de energia coletam e processam os dados do consumo elétrico. A exatidão dessas medições depende diretamente da eficiência e da qualidade do firmware implementado. Um firmware bem projetado garante que as leituras sejam precisas, evitando erros que poderiam resultar em cobranças incorretas para os consumidores ou perdas financeiras para as concessionárias;
2. **Comunicação com a Rede:** A capacidade de um medidor de energia se comunicar eficazmente com a concessionária é fundamental para a operação eficiente do sistema de distribuição elétrica. O firmware permite que o medidor envie informações de consumo e receba atualizações ou comandos da concessionária. Protocolos como o DLMS/COSEM (Device Language Message Specification / Companion Specification for Energy Metering) são implementados no firmware para padronizar essa comunicação, assegurando interoperabilidade e segurança;
3. **Segurança dos Dados:** Em um cenário onde a segurança cibernética se torna cada vez mais crítica, o firmware de medidores de energia deve incorporar mecanismos robustos de segurança. Isso inclui criptografia dos dados, autenticação e verificação de integridade, protegendo assim as informações transmitidas contra acessos não autorizados e manipulações maliciosas;
4. **Facilidade de Atualização:** A possibilidade de atualizar o firmware remotamente é uma característica essencial, especialmente para corrigir problemas, melhorar a segurança ou adicionar novas funcionalidades. Essa capacidade reduz a necessidade de intervenções físicas nos dispositivos, diminuindo custos e aumentando a eficiência operacional das concessionárias de energia.

2.4.1 Segurança de Firmware

O firmware, como o software de controle de baixo nível para dispositivos eletrônicos, é um componente essencial da infraestrutura de medição de energia. Por outro lado, ele também pode ser um ponto de vulnerabilidade importante. A falta de atualizações regulares, códigos maliciosos inseridos durante o desenvolvimento ou até falhas na implementação de protocolos

de segurança são alguns dos vários fatores que podem causar falhas de segurança no firmware. A seguir estão alguns dos principais problemas relacionados ao firmware de medidores de energia:

1. *Inserção de Código Malicioso*: Ataques com código malicioso podem afetar o firmware se ele não for protegido corretamente durante o desenvolvimento ou distribuição. Esses códigos podem causar leituras incorretas, interrupções na comunicação ou até mesmo desativação completa do dispositivo.
2. *Acesso Não Autorizado*: Falhas no sistema de autenticação podem permitir o acesso indevido, o que pode causar possível alteração na maneira como o medidor funciona. Manipulações de dados de consumo, fraude energética ou mesmo ataques coordenados que atingem uma grande parte da infraestrutura elétrica podem resultar disso.
3. *Atualizações Comprometidas*: As atualizações de firmware podem ser interceptadas ou corrompidas sem mecanismos de segurança adequados, o que permite que firmware malicioso seja instalado nos dispositivos. As atualizações realizadas remotamente, sem supervisão física, aumentam o risco.

2.4.2 Atualização de Firmware

A atualização de firmware em dispositivos embarcados é um processo que envolve a substituição ou reprogramação do código que controla o funcionamento do hardware do dispositivo. Esse processo é vital para manter o dispositivo atualizado com melhorias, correções de bugs, ou para adicionar novas funcionalidades. Em sistemas embarcados, a comunicação serial é frequentemente utilizada para a transferência de firmware, devido à simplicidade na comunicação entre dispositivos.

O processo de atualização de firmware por meio de comunicação serial pode ser dividido em várias etapas fundamentais:

- a) **Preparação do Arquivo de Firmware**: o firmware que será atualizado neste trabalho é preparado na forma de um arquivo binário. Esse arquivo contém o código compilado, que será transferido para o dispositivo de destino. O conteúdo do arquivo é organizado de acordo com o layout de memória do dispositivo, definindo seções específicas para o código, dados e áreas de inicialização (*bootloader*). Essas informações devem estar corretamente mapeadas para garantir que o firmware seja gravado no local apropriado da memória do dispositivo;

- b) **Estabelecimento de Comunicação Serial:** para realizar a transferência, uma conexão serial é estabelecida entre o dispositivo de *host* e o dispositivo de destino. Essa comunicação é configurada com parâmetros específicos, como taxa de transmissão (baud rate), bits de dados, bits de parada e paridade. Com a comunicação serial configurada corretamente, o *host* inicia o envio de dados ao dispositivo de destino;
- c) **Transferência de Dados via Serial:** a transferência do firmware ocorre enviando o arquivo binário em pacotes de dados sequenciais. Cada pacote contém uma série de bytes que representam uma parte do firmware. Os pacotes de dados são transmitidos de maneira ordenada e são recebidos pelo dispositivo de destino. Esses pacotes são enviados um após o outro, até que todo o arquivo de firmware tenha sido transferido para o dispositivo;
- d) **Armazenamento Temporário dos Dados no Dispositivo:** ao receber os pacotes de dados, o dispositivo de destino não atualiza diretamente sua memória principal. Em vez disso, os dados são temporariamente armazenados em uma área reservada da memória, como uma Random Access Memory (RAM) ou uma região específica da memória dedicada à atualização de firmware. Esse armazenamento temporário permite que o dispositivo receba todo o firmware antes de sobrescrever o código existente;
- e) **Verificação de Consistência dos Dados:** depois de toda a transferência de firmware, o dispositivo de destino realiza uma verificação básica da consistência dos dados recebidos. A verificação pode envolver a conferência do tamanho total dos dados recebidos, comparando-o com o tamanho esperado, ou conferindo a integridade básica dos pacotes recebidos;
- f) **Reprogramação da Memória do Dispositivo:** após verificar que os dados foram recebidos corretamente, o dispositivo de destino inicia a reprogramação de sua memória. Essa etapa envolve substituir o firmware existente pelo novo código. Durante esse processo, o dispositivo pode desativar temporariamente algumas de suas funcionalidades para garantir que a reprogramação ocorra sem interrupções. Esse processo pode ser gerenciado por um *bootloader*, que é responsável por garantir que o dispositivo possa ser restaurado em caso de falha. Após isso, o dispositivo é reiniciado.

2.4.3 Linguagem de Programação C

A linguagem de programação C (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2020) desempenha um papel fundamental na implementação da lógica de atualização do firmware do medidor inteligente. Originado no Bell Labs/Lucent Technologies (1972) no início dos anos 1970, C é considerado uma linguagem de uso geral projetada para ser eficiente e portátil em múltiplas plataformas. Seu principal objetivo é fornecer um equilíbrio entre flexibilidade e desempenho, permitindo que os programadores escrevam códigos eficientes e de fácil manutenção. A linguagem C tem sido amplamente adotada devido à sua capacidade de criar programas eficientes e de alto desempenho, desde sistemas operacionais (WALLS, 2012c) até aplicativos embarcados.

As principais aplicações da programação em C incluem a criação de sistemas e softwares que requerem controle direto de hardware e a necessidade de otimização de recursos (WALLS, 2012a). A linguagem C é particularmente adequada para o desenvolvimento de firmware, sistemas embarcados e drivers de dispositivos porque fornece acesso direto aos recursos de hardware e fornece controle de alto nível sobre como o sistema funciona. A abordagem de programação C concentra-se na eficiência e na simplicidade, tornando-a uma escolha valiosa para projetos que buscam alto desempenho e baixo consumo de recursos (WALLS, 2012b).

2.4.4 Padrão de Codificação BARR

O uso de códigos de programação padronizados é essencial para garantir a compatibilidade entre diferentes sistemas. Ter um conjunto comum de padrões de codificação permite a integração perfeita entre diferentes programas de software, tornando mais fácil para os desenvolvedores trabalharem juntos e para os usuários alternarem entre diferentes sistemas. Por exemplo, o padrão de codificação Barr-C (BARR, 2018) é um conjunto de regras básicas projetadas para evitar erros no firmware (BRITTON, 2022).

A eficácia da compatibilidade entre Barr-C e outros padrões de codificação, particularmente MISRA C, foi extensivamente pesquisada e demonstrada (BAGNARA *et al.*, 2020). MISRA C é um outro padrão para desenvolvimento de software em linguagem de programação C e recebe esse nome pois foi elaborado pela *Motor Industry Software Reliability Association*. Práticas de codificação padronizadas facilitam o desenvolvimento mais eficiente e econômico de programas de software, o que, em última análise, beneficia tanto os desenvolvedores quanto os

usuários finais.

Tendo em vista as vantagens de padronização do código, o desenvolvimento realizado na implementação da atualização de firmware do medidor foi padronizado com base no Barr-C.

3 METODOLOGIA

Este capítulo detalha a abordagem feita na pesquisa e na implementação da solução que visa atualizar o *firmware* de medidores de energia corretamente e com segurança.

Contextualizando este capítulo, foi realizado inicialmente um estudo aprofundado do protocolo DLMS, com foco nas comunicações e nas etapas envolvidas no processo de *Image Transfer*. Esse estudo foi fundamental para compreender a estrutura de comunicação, as mensagens trocadas e as especificações técnicas que regulam as operações de atualização de *firmware*.

Em seguida, foi utilizado o software proprietário fornecido pelo fabricante do medidor, o qual continha a funcionalidade de atualização de *firmware*. Esse software permitiu uma análise detalhada das trocas de mensagens entre o dispositivo cliente (software) e o medidor de energia. Cada pacote de comunicação registrado em arquivos de log foi cuidadosamente analisado para entender o comportamento do medidor durante o processo de atualização de *firmware*. Essas informações foram essenciais para criar uma base sólida de conhecimento prático, além da teoria estudada.

Com o entendimento das etapas do processo e o comportamento esperado do medidor durante a atualização, iniciou-se o desenvolvimento dos pré-requisitos necessários para o processo de atualização de *firmware*. Esses pré-requisitos envolvem aspectos técnicos, como a configuração de parâmetros, segurança das comunicações, entre outros detalhes fundamentais para garantir que o processo ocorra de forma segura e eficaz.

Na sequência, foi iniciado o planejamento do design do *firmware* do dispositivo, cuja funcionalidade principal era possibilitar a atualização remota do *firmware* do medidor de energia. O design foi estruturado de forma a atender todas as especificações do protocolo DLMS, os requisitos de segurança inerentes ao processo de transferência de imagem e cobrir uma série de possíveis problemas que poderiam ocorrer durante o processo de atualização do *firmware*.

Após o planejamento do *firmware*, prosseguiu-se para a etapa de implementação, onde as funcionalidades planejadas foram codificadas e integradas ao dispositivo. A implementação seguiu uma abordagem modular, o que facilitou a identificação e resolução de problemas durante o desenvolvimento.

3.1 Estudo do Padrão DLMS/COSEM e Objeto Image Transfer

No que diz respeito a metodologia utilizada para pesquisar sobre o padrão DLMS/COSEM foi reunido parte do conhecimento contido nos livros DLMS User Association (2020a) e DLMS User Association (2020b) com o conhecimento da biblioteca DLMS/COSEM implementada na linguagem C desenvolvida pelo Núcleo de Estudo e Pesquisas do Norte e Nordeste (NEPEN).

Para que o processo de atualização fosse feito da melhor maneira possível, foi estudado o Objeto Image Transfer presente na documentação do DLMS/COSEM. Este objeto é responsável unicamente pela atualização de *firmware* do medidor. A partir desse estudo foi retirado o seguinte passo a passo:

- a) **Pré-requisito:** atributo *image_transfer_enabled* definido como *true*;
- b) **Etapa 1:** realizar uma GetRequest ao atributo *image_block_size* para saber qual tamanho de bloco suportado pelo medidor;
- c) **Etapa 2:** O cliente inicia o processo de transferência de imagem invocando o método *image_transfer_initiate*. O parâmetro de chamada do método contém o identificador e o tamanho da Image a ser transferida. O servidor disponibilizará o espaço de memória necessário para acomodar a Imagem. Após o sucesso na iniciação, o valor do atributo *image_transfer_status* é (1). O atributo *image_transferred_blocks_status* deve ser resetado, o valor do atributo *image_first_not_transferred_block_number* deve ser definido como (0) e o valor do atributo *image_to_activate_info* deve ser redefinido. O processo de transferência de imagem é iniciado e o servidor COSEM está preparado para aceitar ImageBlocks;
- d) **Etapa 3:** O cliente consegue transferir ImageBlocks para o servidor por meio do método *image_block_transfer*. Os parâmetros de invocação do método incluem ImageBlockNumber e um ImageBlock. Os ImageBlocks são aceitos apenas pelos COSEM servers em que o processo de Image Transfer foi iniciado corretamente. Outros servidores descartam silenciosamente quaisquer ImageBlocks recebidos;
- e) **Etapa 4:** O cliente checa , individualmente com cada servidor, a completude da Image transferida. Se a Imagem não estiver completa, ele transfere os ImageBlocks não transferidos ainda. Isso é um processo iterativo, continua até que toda Image seja transferida com sucesso. Para identificar e transferir Ima-

geBlocks não transferidos, o cliente pode recuperar o ImageBlockNumber do primeiro bloco não transferido. Isso é realizado recuperando o valor do atributo *image_first_not_transferred_block_number*. O cliente transfere então o ImageBlock ainda não transferido depois disso, o cliente checa de novo a completude da Image;

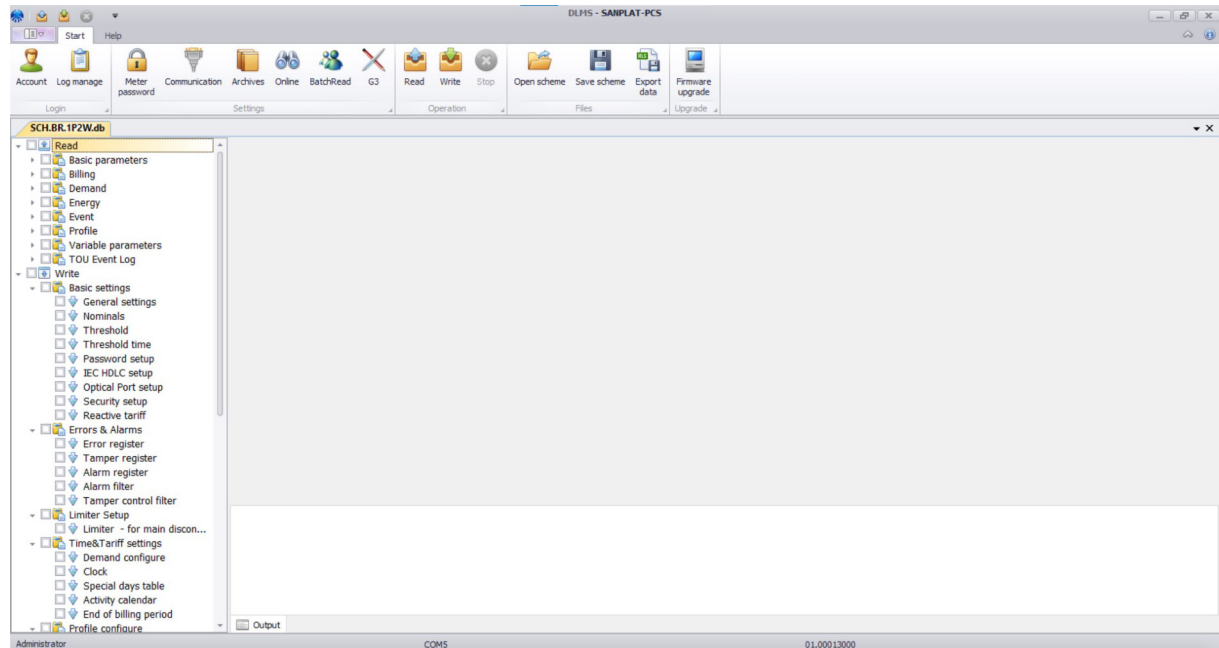
- f) **Etapa 5:** A Imagem é verificada pelo servidor. Esta etapa pode ser iniciada pelo cliente invocando o método *image_verify*. O resultado pode ser: **success**, se a verificação puder ser concluída; **temporary-failure**, se a verificação não foi completa; **other-reason**, se a verificação falhou;
- g) **Etapa 6:** O resultado da verificação pode ser verificado pelo cliente recuperando o atributo *image_transfer_status*. O valor deste atributo é atualizado como resultado da verificação da Imagem. Uma Imagem transferida pode conter uma ou mais Imagens a serem ativadas. Para cada Imagem ser ativada, o atributo *image_to_activate_info* contém os parâmetros: *image_to_activate_size*, *image_to_activate_identification*, *image_to_activate_signature* como mostra a Figura 14. Se essas informações não forem as informações esperadas, o cliente poderá reiniciar a transferência da imagem;
- h) **Etapa 7:** A Imagem é ativada pelo servidor. Esta etapa pode ser iniciada invocando o método *image_activate* pelo cliente. Se a ativação foi concluída sem verificação prévia, a verificação é feita implicitamente como parte da ativação. O resultado da invocação do método da Imagem pode retornar: **success**, se a ativação da Imagem foi iniciada com sucesso; **temporary-failure**, se a verificação/ativação não foi completa; **other-reason**, se a ativação falhou. No caso de sucesso, o server realiza a ativação da nova Imagem. Durante esse processo, ele estará inacessível. Depois da Imagem ser ativada, o resultado pode ser checado pelo cliente recuperando o valor do atributo *image_transfer_status* ou pela leitura dos objetos COSEM apropriados que guardam o identificador, versão e assinatura digital do *firmware* ativo.

3.2 Instalação do Sanplat e Estudo da Funcionalidade de Atualização de Firmware

Nesta etapa foi requisitado à fabricante do medidor o *software* de configuração e de leitura dos dados do medidor de energia e a licença para uso no **Desktop**. O programa utilizado

chama-se Sanplat e tem a seguinte interface gráfica:

Figura 17 – Interface gráfica do programa Sanplat do medidor de energia NSX P213i



Fonte: Elaborado pelo autor.

Percebe-se no canto superior da Figura 17 a funcionalidade chamada **Firmware Upgrade** na qual é disponibilizada pela fabricante o serviço de atualização da imagem do medidor de energia por meio da comunicação serial. Além disso, este serviço também armazena dentro da pasta local do programa um arquivo que contém o histórico dos pacotes de comunicação trocados entre cliente(Computador) e servidor(medidor NSX P213i). A partir disso, foi possível entender de maneira prática o processo geral e particularidades do medidor NSX P213i. Segue as principais propriedades do medidor de energia utilizado:

1. O atributo *image_block_size*(suportado) tem valor igual a 128 bytes;
2. O parâmetro que está contido na estrutura do atributo *image_to_activate_info* chamado de *image_to_activate_identification* tem o tamanho de 27 bytes e segue o seguinte formato de exemplo **S13S12 R1-01.BR.3025.V4.4.3**;
3. O parâmetro que está contido na estrutura do atributo *image_to_activate_info* chamado de *image_to_activate_signature* tem o tamanho de 12 bytes

3.3 Definição dos Pré-requisitos para Elaboração da Implementação

Em primeiro plano, para assegurar que a implementação feita neste trabalho contenha o mínimo de brechas possíveis para fraude, foi necessário estabelecer alguns pré-requisitos antes

da implementação do trabalho proposto. Além disso, o próprio processo de transferência de imagem entre dispositivo cliente e servidor necessita a estruturação de um ambiente para que o processo ocorra da melhor maneira possível. Dessa maneira, estabeleceu-se os seguintes requisitos:

1. **Transferência via Long Range Wide Area Network (LoRaWAN):** O dispositivo cliente apresentado na Figura 4 possui conectividade LoRaWAN, o que permite a transferência da imagem por meio de redes de longa distância de baixa potência. Neste cenário, o *firmware* do medidor é carregado inicialmente em um servidor central e, em seguida, transferido para o dispositivo cliente através da rede LoRaWAN. Esse processo foi escolhido devido à sua capacidade de cobrir grandes áreas com baixo consumo de energia, o que o torna ideal para locais remotos onde a conectividade direta com o medidor seria difícil ou inviável. A transferência via LoRaWAN permite que o *firmware* seja transferido para o dispositivo cliente e que a implementação possa ser ativada remotamente, sem a necessidade de proximidade física entre o dispositivo cliente e o medidor, garantindo flexibilidade e eficiência no processo de atualização;
2. **Assinatura do *Firmware*:** Para assegurar que a imagem a ser transferida não sofra alterações e garantir a autenticidade do mesmo, foi realizada a assinatura digital do arquivo de *firmware* antes de enviá-lo ao dispositivo cliente. Este procedimento é essencial para garantir que o *firmware* recebido e posteriormente transferido ao medidor de energia seja legítimo, evitando a instalação de código não autorizado ou corrompido no medidor de energia. A verificação dessa assinatura acontece após o processo de transferência completa do *firmware* ao dispositivo cliente, assegurando a integridade do *firmware* antes da execução final da atualização;
3. **Transferência para Memória Não Volátil:** A transferência do *firmware* do medidor de energia para o dispositivo cliente é uma etapa crucial, pois envolve o armazenamento seguro do arquivo em uma região específica da memória não volátil. Este processo garante que o *firmware* esteja prontamente disponível para o dispositivo realizar o processo de atualização conforme necessário. O uso da memória não volátil assegura que os dados do *firmware* sejam preservados, mesmo em casos de falha de energia, garantindo a continuidade da atualização após um reinício.

3.4 Definição da Arquitetura de Firmware da Implementação

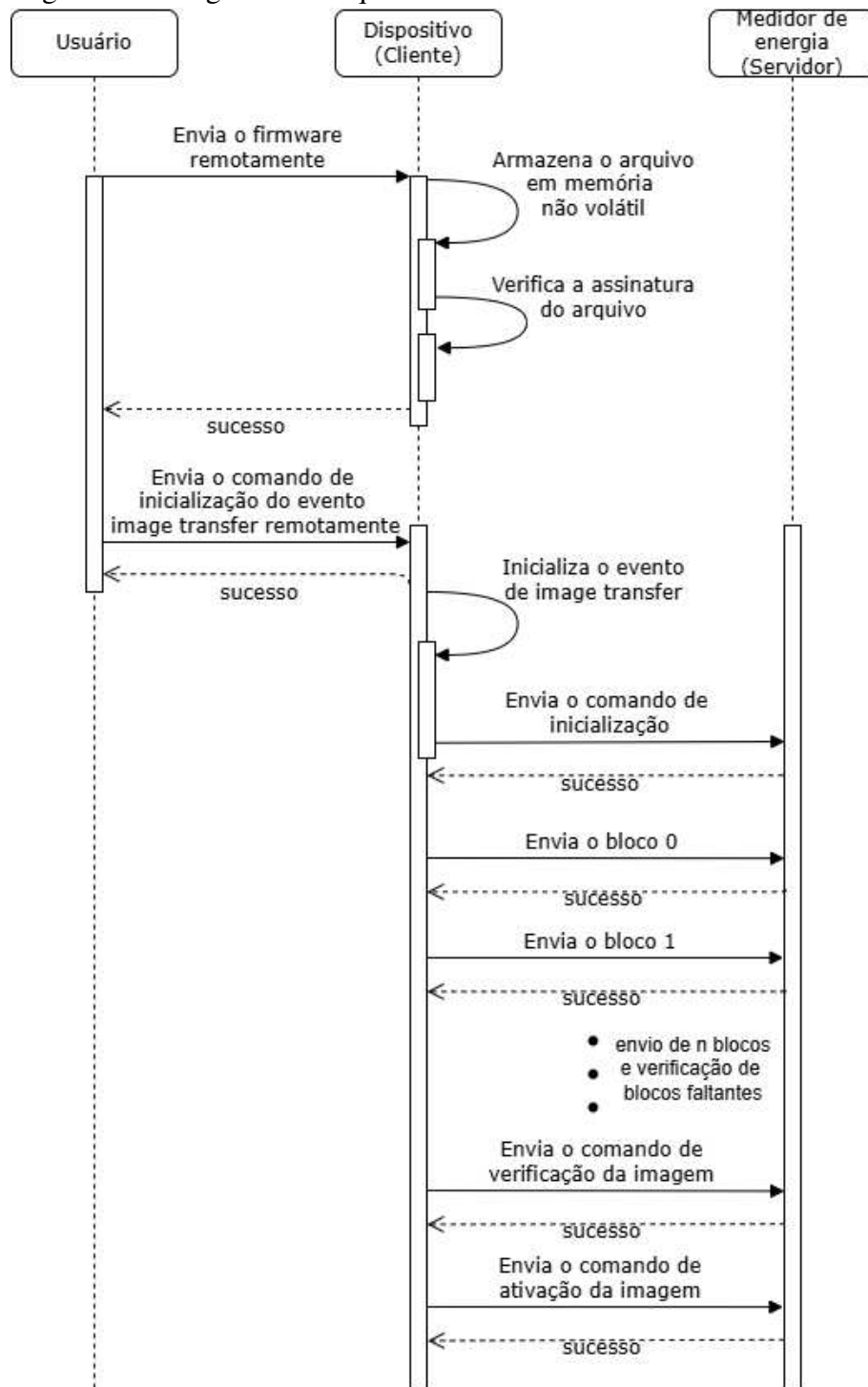
Na implementação da funcionalidade de atualização de *firmware* no dispositivo acoplado ao medidor de energia, adotou-se uma abordagem baseada em eventos, utilizando a linguagem C. O desenvolvimento da arquitetura de *firmware* seguiu um fluxo contínuo de comunicação com o medidor, sempre assegurando a confiabilidade e robustez do processo de atualização, essencial para um cenário de dispositivos de campo.

A arquitetura foi planejada para garantir que o processo de atualização ocorresse de forma automatizada, sem a necessidade de intervenção manual constante. O escopo geral da funcionalidade foi estruturado em três pontos principais:

1. **Transferência do *Firmware* Assinado via LoRaWAN:** como dito na Seção 3.3, o *firmware* assinado deve ser transferido para o dispositivo cliente via LoRaWAN para armazenamento em memória não volátil. Dessa forma, precisou-se definir no *firmware* um endereço base específico e um tamanho máximo dedicado exclusivamente para o armazenamento do *firmware*. Isso garante que o arquivo seja armazenado de maneira organizada e segura dentro da memória do dispositivo, e que o espaço destinado ao *firmware* não seja ocupado por outros dados. A configuração dessa região na memória flash foi essencial para garantir que o processo de atualização ocorra sem problemas e que o *firmware* seja armazenado integralmente. Logo após a transferência completa há a verificação da assinatura do *firmware*. O processo total de transferência dura em torno de 24 horas.
2. **Envio do Comando de Ativação da Atualização:** Após a transferência completa do *firmware*, é necessário enviar o comando de ativação remotamente via LoRaWAN para o dispositivo cliente acoplado ao medidor. Esse comando sinaliza para o dispositivo que o processo de transferência de imagem ao medidor deve ser iniciado.
3. **Disparo do Evento `image_transfer_client`:** Após o recebimento do comando de ativação, é disparado a execução do evento `image_transfer_client`, pois toda a lógica de controle do processo para lidar com a comunicação serial contínua entre o dispositivo e o medidor é gerenciada por este evento. Este evento é responsável por coordenar a atualização, desde a transferência do *firmware* até a ativação e a conclusão do processo.

Para melhor demonstrar visualmente a arquitetura genérica da implementação feita neste trabalho, a Figura 18 demonstra a sequência de configuração e a sequência de execução do Image Transfer.

Figura 18 – Diagrama de sequência



3.5 Definição do Design de Firmware da Implementação

Para o design da implementação, foi escolhida a programação orientada a eventos. Como descrito anteriormente, todo controle do processo da atualização de *firmware* é gerenciado pelo evento *image_transfer_client* que foi projetado para rodar continuamente, monitorando o

estado da atualização e interagindo com o medidor via comunicação serial conforme necessário.

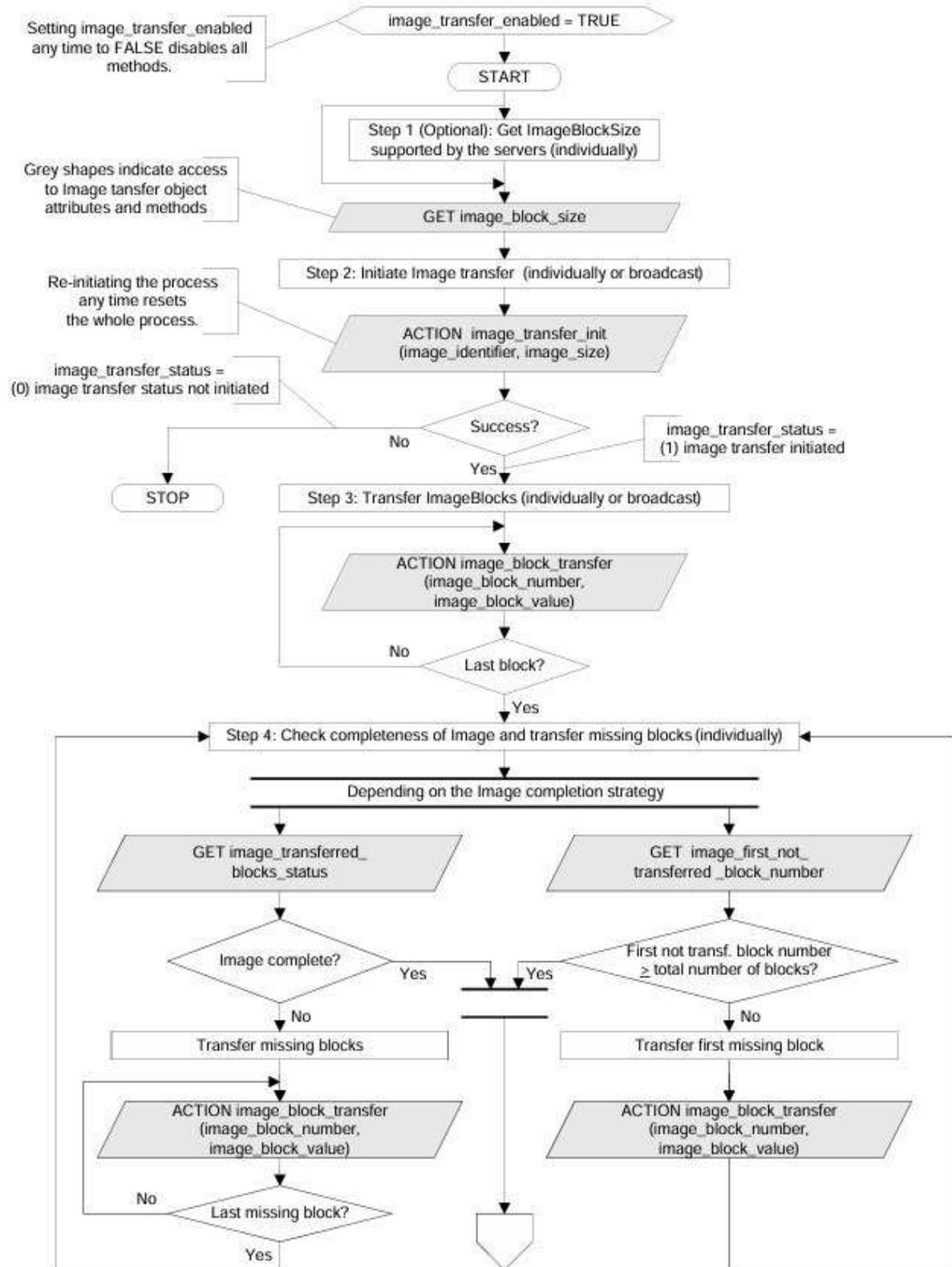
O fluxo de controle do evento inclui verificações periódicas da conexão com o medidor. Caso houvesse perda de comunicação, o evento entra em um estado de espera, aguardando um tempo antes de tentar restabelecer a comunicação. Esse comportamento foi essencial para lidar com as incertezas da comunicação e garantir que a atualização ocorresse de maneira confiável, mesmo em cenários onde a conexão pudesse ser intermitente.

O evento também lida com diferentes estados do processo de atualização, como a parte de iniciação do processo, transferência dos blocos de *firmware*, verificação de integridade e a ativação do *firmware*. Cada um desses estados foi implementado como sub-rotinas dentro do evento principal, permitindo um controle preciso e eficiente do processo de atualização. A Figura 19 e Figura 20 descrevem o diagrama de fluxo do evento implementado baseado no fluxo sugerido pelo livro DLMS User Association (2020a).

Para esclarecer melhor o fluxo implementado em comparação com o sugerido pelo livro, a sequência escolhida neste trabalho segue todos os passos obrigatórios com as seguintes particularidades:

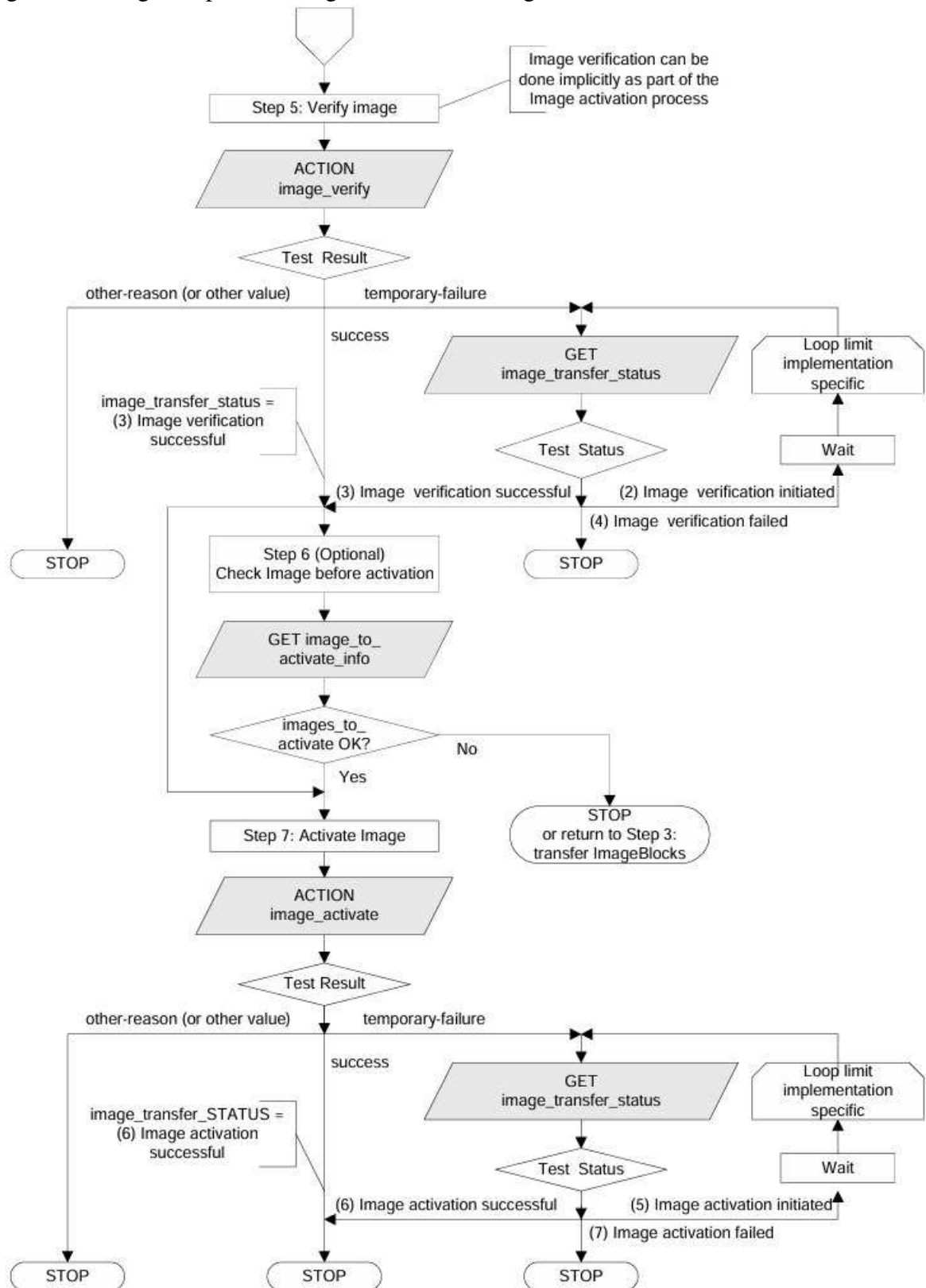
- a) A etapa 1 que é opcional foi implementada e retornou o tamanho de bloco suportado pelo medidor de energia da Figura 3 : 128 bytes;
- b) A etapa 2 foi feita individualmente;
- c) A etapa 3 foi feita individualmente;
- d) A etapa 4 adotada a estratégia de verificar a presença de blocos faltantes no medidor pelo número do primeiro bloco não transferido;
- e) A etapa 6 que é opcional não foi adotada.

Figura 19 – Primeira parte do diagrama de fluxo Image Transfer



Fonte: DLMS User Association (2020a).

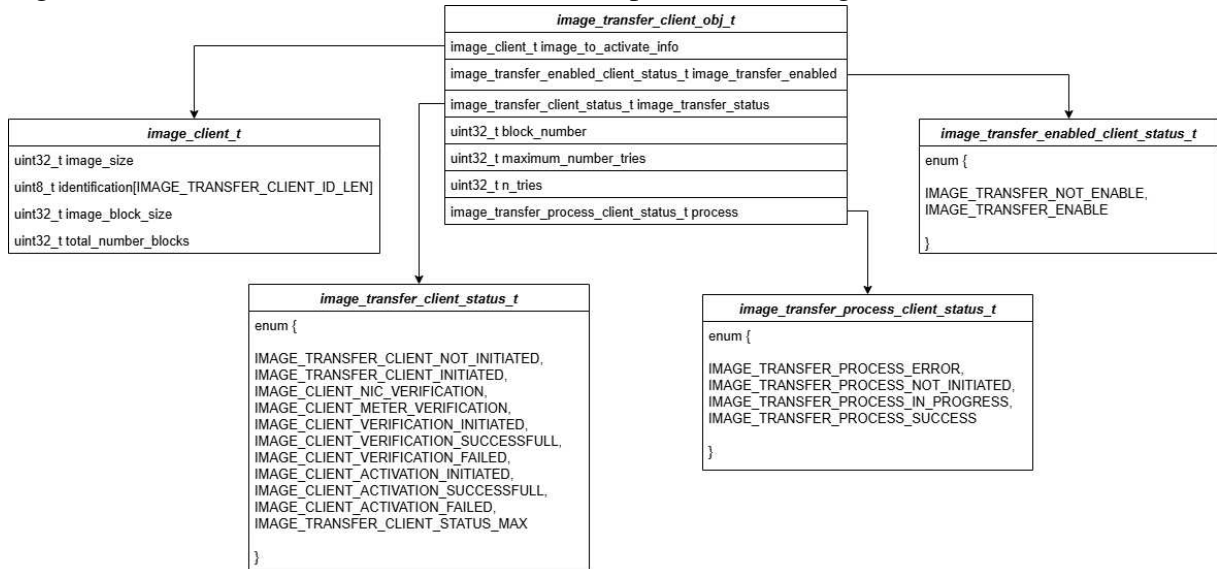
Figura 20 – Segunda parte do diagrama de fluxo Image Transfer



Fonte: DLMS User Association (2020a).

Para cumprir a proposta de gerenciamento dos estados do processo, foi instânciado na implementação um parâmetro que armazena as principais informações do processo. Essa variável de controle está representada na Figura 21.

Figura 21 – Instância da variável de controle do processo de Image Transfer



Fonte: Elaborado pelo autor.

3.6 Gerenciamento de Estados da Implementação

Este trabalho contém a implementação de um serviço de atualização de *firmware* desenvolvido para empresa NEPEN, a qual detém a propriedade intelectual do código fonte. Esse serviço é integrado a um produto comercializado pela empresa e foi desenvolvido utilizando a linguagem de programação C. Em virtude da proteção de propriedade intelectual, o código fonte completo da implementação não pode ser divulgado neste trabalho.

Entretanto, para transmitir a ideia central e a lógica por trás da implementação, é descrito e representado de maneira ilustrativa o pseudocódigo da máquina de estados responsável por gerenciar todo o processo de atualização de *firmware*. Este pseudocódigo reflete as principais transições de estados e decisões tomadas pela implementação durante a execução do serviço, abordando desde a inicialização do processo até a ativação da Imagem para garantir a conclusão bem-sucedida do processo.

Dessa forma, o Algoritmo 1 apresentado demonstra a implementação da recuperação e configuração dos parâmetros de pré-inicialização e de inicialização. Em seguida, o Algoritmo 2 é referente a lógica da transferência e verificação de blocos com recuperação de blocos pendentes.

Por último, o Algoritmo 3 retrata a lógica da verificação e ativação da Imagem transferida ao medidor de energia.

A implementação do fluxo sugerido por DLMS User Association (2020a), representado nas Figuras 19 e 20, foi adaptada de acordo com as modificações descritas na Seção 3.5. Essas adaptações foram incorporadas nos algoritmos desenvolvidos, resultando em um processo capaz de atualizar o *firmware* de medidores de energia utilizando o padrão DLMS/COSEM.

Algoritmo 1: Pseudocódigo da pré-inicialização e inicialização do Image Transfer

Parte 1: Pré-Inicialização

```

get-request(image_block_size);
// leitura da memória não volátil
read(image_size);
read(image_identification);
// Calcular o número total de blocos
total_number_blocks = (image_size – SIGNATURE_SIZE)/image_block_size;
// Montar a estrutura data
build_data_param(data, image_size, image_identification);

```

Parte 2: Inicialização

```

// Iniciar o processo de Image Transfer
action-request: image_transfer_init(data);
if get-request(image_transfer_status) == IMAGE_TRANSFER_CLIENT_INITIATED
  then
  | create_block_transfer(image_block, image_block_number);
  | // Seguir para etapa de transferência dos blocos
else
  | // Desabilitar o evento
end

```

Algoritmo 2: Pseudocódigo da transferência e verificação de blocos

Parte 3: Transferência de blocos

```
// transferência de todos os blocos
while image_block_number < total_number_blocks do
  | action-request:image_block_transfer(image_block);
  | create_block_transfer(image_block, image_block_number);
end
```

Parte 4: Verificação de blocos pendentes

```
// requisita o primeiro bloco não transferido
image_first_not_transferred_block_number =
  get-request(image_first_not_transferred_block_number);
while image_first_not_transferred_block_number < total_number_blocks do
  | create_block_transfer(image_block, image_first_not_transferred_block_number);
  | action-request:image_block_transfer(image_block);
  | image_first_not_transferred_block_number =
  | get-request(image_first_not_transferred_block_number);
end

// Seguir para etapa de verificação da Imagem
```

Algoritmo 3: Pseudocódigo da verificação e ativação da imagem

Parte 5: Verificação da Imagem

```
if action-request:image_verify(data) == SUCCESS then
  | // segue para etapa de ativação
else
  | // desabilita o evento
end
```

Parte 6: Ativação da Imagem

```
if action-request:image_activate(data) == SUCCESS then
  | // imagem ativada com sucesso
else
  | // falha ao ativar a imagem
end

// desabilita o evento
```

4 RESULTADOS

Neste capítulo, serão apresentados os resultados gerais obtidos durante o processo de desenvolvimento, resultados qualitativos e quantitativo. Serão discutidos os resultados quantitativos, como o tempo de duração da atualização da implementação proposta em comparação ao teste da implementação do software proprietário do medidor de energia, como os qualitativos, que englobam funcionalidade, segurança e usabilidade. Os resultados se baseiam em testes comparativos realizados sob as mesmas condições para garantir uma análise justa e precisa.

4.1 Descrição dos Cenários Testados

Foram definidos dois cenários principais de teste: o primeiro envolve a atualização de *firmware* utilizando o software proprietário fornecido pelo fabricante do medidor de energia, enquanto o segundo cenário utiliza a implementação própria desenvolvida durante este trabalho. Ambos os cenários foram testados nas seguintes condições:

1. Medidor utilizado: NANSEN NSX P213i;
2. Versão de *firmware* do medidor antes da atualização: 4.4.3;
3. Versão de *firmware* transmitida: 4.5.6;
4. Comunicação serial: RS485.

4.2 Resultados Quantitativos: Comparação de Tempos de Atualização

A Tabela 3 apresenta os tempos de duração da atualização de *firmware* nos dois cenários testados.

Tabela 3 – Tempo de duração do processo de atualização do *firmware* do medidor de energia

Cenário de atualização	Tempo de duração (em minutos)
Software Proprietário do Medidor	10
Implementação Desenvolvida	12

Fonte: Elaborado pelo autor.

O principal objetivo da implementação desenvolvida foi garantir que a funcionalidade de atualização de *firmware* fosse realizada com sucesso. Diferente do software proprietário, o foco inicial desta implementação não foi otimizar o tempo de execução do processo, o que abre espaço para melhorias futuras.

Embora o tempo total de atualização utilizando a implementação própria tenha sido de 12 minutos, comparado aos 10 minutos do software proprietário, essa diferença é justificada por algumas decisões de design importantes. Uma delas foi a introdução de um tempo de espera adicional após o comando de ativação do *firmware*. Durante a etapa de ativação, o medidor de energia passa por um período em que ele não responde à comunicação, enquanto processa a atualização do seu *firmware*. Para evitar sobrecarga no dispositivo cliente, que precisa continuar executando outras tarefas durante esse período, foi incluído um tempo de espera antes de o dispositivo verificar novamente o status do medidor.

Essa abordagem foi adotada para garantir a continuidade das outras operações do dispositivo cliente, sem interromper suas funcionalidades enquanto o medidor realiza a atualização do *firmware*. No futuro, melhorias podem ser feitas para otimizar esse processo e reduzir o tempo total de atualização, mantendo a integridade do sistema e a eficiência nas operações.

Portanto, a diferença de dois minutos entre os dois cenários pode ser atribuída a essa escolha de design, que priorizou a robustez e a manutenção das demais funcionalidades do dispositivo cliente durante o processo de atualização.

Além disso, o processo completo de atualização proposto neste trabalho segue uma sequência específica. Inicialmente, o *firmware* é enviado via rede LoRaWAN para o dispositivo cliente, um processo que pode durar em torno de 24 horas, dependendo das condições da rede e do volume de dados transferidos. Após a conclusão da transferência do *firmware*, o comando de inicialização do evento é enviado ao dispositivo cliente, que, em seguida, inicia o processo de atualização do medidor. Esta etapa final, que consiste na atualização propriamente dita e citada anteriormente, tem uma duração de aproximadamente 12 minutos.

Portanto, o tempo total para a conclusão da atualização do *firmware*, desde o início da transferência do arquivo até a conclusão da instalação, é de aproximadamente 24 horas e 12 minutos. Apesar do tempo elevado devido à transferência via LoRaWAN, o processo geral ainda tem benefícios em comparação com a atualização de *firmware* presencial e manual feita pela fabricante.

4.3 Resultados Qualitativos

Nesta seção, são apresentados os resultados qualitativos relacionados à segurança, confiabilidade, redução de custos e eficiência do processo de atualização de *firmware* em

medidores de energia utilizando o padrão DLMS/COSEM.

Em vez de depender de métodos tradicionais de atualização que frequentemente exigem intervenções manuais e contato físico com o dispositivo, a adoção dessa implementação permite que o processo seja realizado de forma automatizada e remota, pois o envio do arquivo binário de *firmware* do medidor é enviado remotamente via LoRaWAN e ativação do processo de atualização pelo dispositivo cliente acoplado ao medidor também pode ser ativado remotamente via LoRaWAN. A partir deste ponto, todo processo ocorre em meio físico entre dispositivo coletor de dados e medidor inteligente de energia.

Uma das principais vantagens é a redução de esforços manuais. Tradicionalmente, atualizações de *firmware* em dispositivos espalhados por grandes áreas geográficas, como medidores de energia em cidades ou zonas rurais, requerem a visita de técnicos ao local de instalação. Isso envolve custos com transporte, mão de obra e o tempo de indisponibilidade dos dispositivos durante o procedimento. Com a implementação de atualizações remotas via DLMS/COSEM, essas visitas se tornam desnecessárias, permitindo que o *firmware* seja atualizado de forma centralizada a partir de uma sala de controle ou de uma plataforma de gestão remota.

Essa capacidade de atualização remota também leva a uma significativa redução de custos operacionais. Empresas de serviços públicos podem atualizar simultaneamente milhares de medidores de energia sem incorrer nos custos associados a deslocamentos e interrupções. Além disso, as atualizações podem ser programadas para horários de menor demanda, minimizando o impacto nas operações diárias e garantindo a continuidade do serviço.

Outro aspecto importante é o aumento da eficiência no processo de manutenção e atualização dos dispositivos. Através do objeto Image Transfer, o protocolo DLMS/COSEM permite que todo o processo de transferência de imagem seja gerenciado automaticamente. Isso inclui desde a transferência do arquivo de *firmware* até a ativação do novo *firmware* no medidor. O processo pode ser monitorado e verificado remotamente, permitindo que qualquer falha seja identificada rapidamente e corrigida sem necessidade de intervenção física.

Além disso, a automação do processo reduz a possibilidade de erros humanos, que podem ocorrer durante uma atualização manual. Isso é especialmente importante quando consideramos que um erro de *firmware* pode deixar o medidor inoperante, resultando em custos adicionais para sua recuperação. Com a automação oferecida pela implementação, as atualizações seguem um procedimento padronizado e testado, reduzindo consideravelmente os riscos associados.

Outros resultados notáveis estão relacionados aos mecanismos de verificação da integridade do *firmware* e as camadas de proteção criptográfica aplicadas durante a transferência e aplicação da atualização.

Após a transferência do *firmware* via LoRaWAN para o dispositivo cliente foi implementado um rigoroso processo de verificação de assinatura digital. O *firmware* contém uma assinatura digital nos últimos 64 bytes do arquivo, e o dispositivo cliente é responsável por realizar a checagem da integridade desse *firmware* antes de transmiti-lo ao medidor de energia. O processo de verificação é realizado da seguinte forma:

- a) O dispositivo cliente calcula a hash do arquivo de *firmware*, excluindo os últimos 64 bytes que contêm a assinatura digital (ou seja, o cálculo da hash é feito sobre o conteúdo total do arquivo, exceto pela assinatura digital).
- b) Após calcular a hash do *firmware*, o dispositivo cliente extrai a assinatura digital dos últimos 64 bytes do arquivo e a compara com a hash previamente calculada.
- c) Se a assinatura calculada não corresponder à assinatura digital extraída, o *firmware* é rejeitado, garantindo que apenas um *firmware* íntegro e não modificado seja aceito pelo medidor.

Esse mecanismo de verificação de integridade foi testado em diversas condições, e os resultados demonstraram sua eficácia na identificação de *firmware* corrompido ou adulterado, pois se houver uma alteração de tamanho ou conteúdo do arquivo a assinatura transmitida não será igual a calculada pelo dispositivo cliente. A rejeição automática de arquivos que não passaram na verificação evita a instalação de *firmware* malicioso, reforçando a confiabilidade do sistema e assegurando a segurança operacional dos medidores de energia.

Primeiramente, para exemplificar um caso de sucesso, o arquivo binário de *firmware* do medidor de energia foi enviado via LoRaWAN para o dispositivo cliente assinado corretamente. A Figura 22 corresponde à janela de depuração do código na qual foi utilizado a ferramenta da *SEGGER* chamada de *JLink RTT Client*. Nessa figura, percebe-se que o processo terminou com sucesso e, portanto, a imagem do medidor foi gravada com sucesso na região de memória reservada do dispositivo cliente.

Por outro lado, para exemplificar um caso de erro, foi retirado um byte do arquivo binário de *firmware* do medidor de energia. Após isso, foi transferido o arquivo via LoRaWAN para o dispositivo cliente. Com a alteração do arquivo de *firmware* do medidor, a Figura 23 demonstra que, ao final do processo de transferência desse arquivo, a assinatura é invalidada e o

Figura 22 – Depuração do processo de transferência de imagem do medidor original

```

J-Link RTT Client V7.94d
###RTT Client: *****
###RTT Client: *          SEGGER Microcontroller GmbH          *
###RTT Client: * Solutions for real time microcontroller applications *
###RTT Client: *****
###RTT Client: *
###RTT Client: *      (c) 2012 - 2016 SEGGER Microcontroller GmbH      *
###RTT Client: *
###RTT Client: *      www.segger.com      Support: support@segger.com      *
###RTT Client: *
###RTT Client: *****
###RTT Client: *
###RTT Client: * SEGGER J-Link RTT Client   Compiled Jan 11 2024 10:54:31 *
###RTT Client: *
###RTT Client: *****
###RTT Client: -----
###RTT Client: Connecting to J-Link RTT Server via localhost:19021 ...
###RTT Client: Connected.

SEGGER J-Link V7.94h - Real time terminal output
SEGGER J-Link EDU V11.0, SN=261010376
Process: JLinkGDBServerCL.exe
Initiate.
IMAGE_SIZE: 221275 bytes.
Initiate Block Transfer.
Valid signature.
Process completed successfully.

```

Fonte: Elaborado pelo autor.

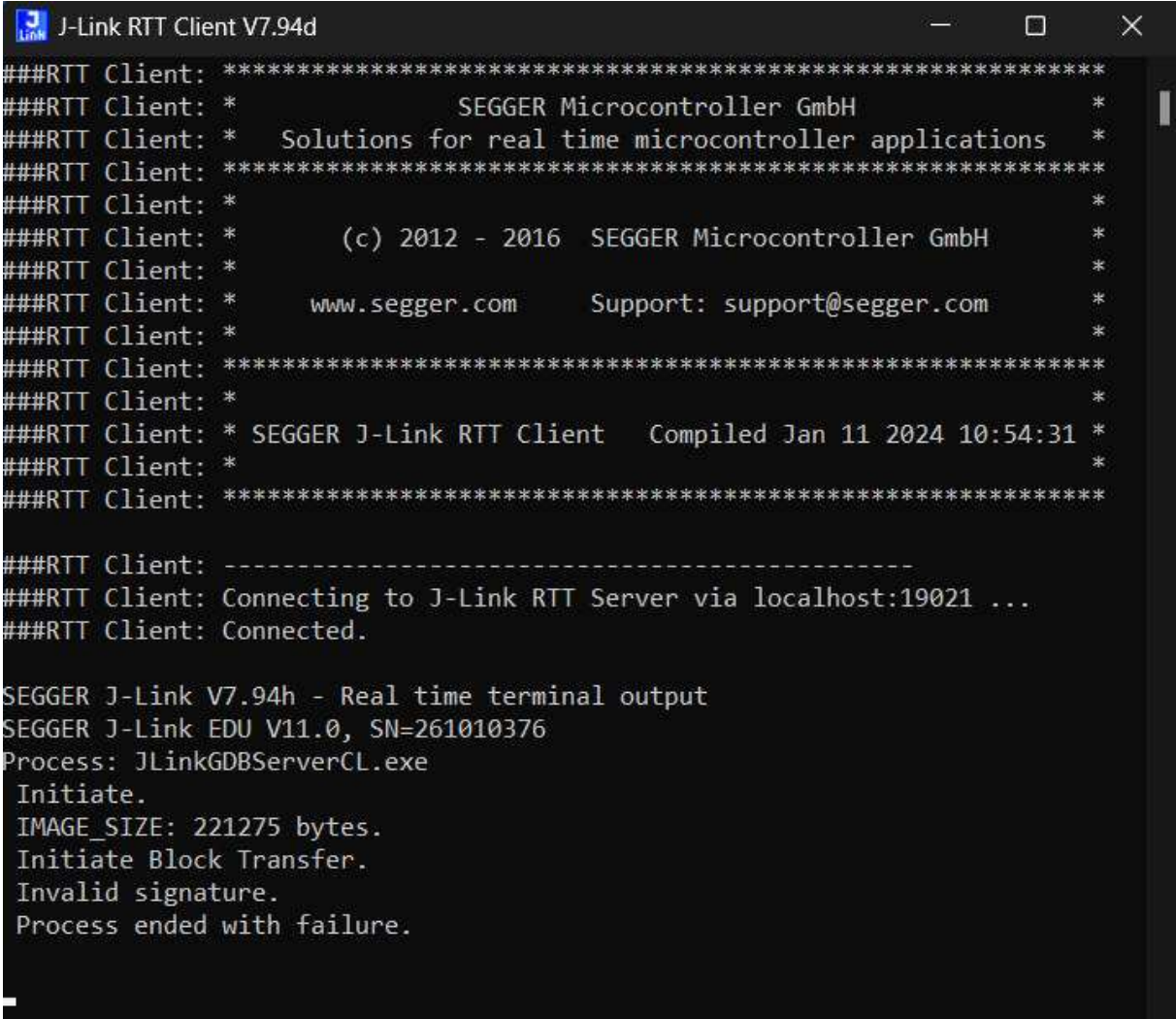
processo termina com falha. Dessa forma, a verificação da assinatura garante a autenticidade, integridade e não repúdio.

Além da verificação de integridade, foi implementada a criptografia dos pacotes de dados no padrão DLMS/COSEM utilizando o nível de segurança HLS com criptografia *HighGMAC*. Esse método de criptografia foi escolhido por sua capacidade de proteger os dados transmitidos entre o dispositivo cliente e o medidor de energia.

O processo de criptografia depende de uma chave compartilhada, conhecida apenas pelo dispositivo cliente e pelo medidor de energia. Essa chave pode ser única para cada dispositivo, o que adiciona uma camada extra de segurança, isolando eventuais compromissos de segurança a dispositivos específicos, sem comprometer toda a rede.

Por último, a implementação desenvolvida apresenta robustez e resiliência em cenários de falha de energia durante o processo de atualização. Em caso de interrupção no fornecimento de energia, o dispositivo cliente é capaz de detectar a ocorrência da queda e salvar os parâmetros que controlam o progresso do evento *image_transfer_client* em uma região de memória não volátil. Com isso, quando o dispositivo é religado, o processo de atualização é

Figura 23 – Depuração do processo de transferência de imagem do medidor alterada



```

J-Link RTT Client V7.94d
###RTT Client: *****
###RTT Client: *                SEGGER Microcontroller GmbH                *
###RTT Client: *   Solutions for real time microcontroller applications   *
###RTT Client: *****
###RTT Client: *
###RTT Client: *           (c) 2012 - 2016  SEGGER Microcontroller GmbH           *
###RTT Client: *
###RTT Client: *   www.segger.com      Support: support@segger.com      *
###RTT Client: *
###RTT Client: *****
###RTT Client: *
###RTT Client: *   SEGGER J-Link RTT Client    Compiled Jan 11 2024 10:54:31   *
###RTT Client: *
###RTT Client: *****

###RTT Client: -----
###RTT Client: Connecting to J-Link RTT Server via localhost:19021 ...
###RTT Client: Connected.

SEGGER J-Link V7.94h - Real time terminal output
SEGGER J-Link EDU V11.0, SN=261010376
Process: JLinkGDBServerCL.exe
Initiate.
IMAGE_SIZE: 221275 bytes.
Initiate Block Transfer.
Invalid signature.
Process ended with failure.

```

Fonte: Elaborado pelo autor.

retomado a partir do ponto exato onde foi interrompido, garantindo a continuidade e a integridade da atualização, sem a necessidade de reiniciar o procedimento.

5 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou a implementação de um sistema de atualização segura de *firmware* para medidores inteligentes de energia elétrica, baseado no padrão DLMS/COSEM e na utilização do objeto *Image Transfer*. O objetivo principal foi garantir a segurança, autenticidade e integridade das atualizações de *firmware* realizadas remotamente, atendendo às necessidades de confiabilidade e conformidade com as regulamentações do setor de energia elétrica no Brasil.

Os resultados quantitativos demonstraram que a implementação desenvolvida, embora não tenha sido otimizada para reduzir o tempo de atualização, apresentou um desempenho competitivo em relação ao *software* proprietário do medidor. O tempo de atualização do *firmware* foi de 12 minutos para a implementação desenvolvida, comparado aos 10 minutos do *software* proprietário. Mesmo com um foco maior na implementação da funcionalidade e não na otimização de tempo, a solução apresentou um desempenho favorável.

Além disso, os resultados qualitativos mostraram aspectos importantes da segurança e confiabilidade do sistema. A comunicação serial entre o dispositivo cliente e o servidor foi criptografada utilizando o nível de segurança HLS do DLMS, o que protege as informações trocadas durante o processo de atualização. Além disso, a verificação da assinatura digital do arquivo de *firmware* antes da atualização garantiu a autenticidade e integridade do código, prevenindo possíveis ataques que poderiam comprometer a segurança das medições e das operações do medidor. Este processo também garantiu o não repúdio, assegurando que a imagem transferida fosse legítima e não adulterada.

Ademais, a implementação do armazenamento do estado do processo após detecção de queda de energia também impacta positivamente o processo de atualização de imagem, pois esse resultado demonstra a robustez e confiabilidade da implementação desenvolvida, uma vez que o mecanismo de recuperação automática após quedas de energia evita a perda de progresso e previne a necessidade de intervenção manual para reiniciar o processo. Isso assegura que as atualizações possam ser realizadas com maior segurança e eficiência, reduzindo riscos de falhas e indisponibilidade do sistema, além de minimizar custos operacionais. A capacidade de retomar o evento *image_transfer_client* de maneira automática reforça a resiliência do sistema, tornando-o adequado para ambientes críticos de infraestrutura elétrica.

Apesar dos resultados positivos, algumas limitações foram observadas. A implementação desenvolvida poderia ser otimizada para reduzir ainda mais o tempo de atualização, o que seria benéfico em cenários de larga escala, onde múltiplos medidores precisam ser atualizados

simultaneamente. Essa otimização pode ser uma linha futura de pesquisa para garantir maior eficiência operacional.

O impacto dessa implementação no setor de energia elétrica no Brasil é significativo. A AMI é um pilar essencial das Redes Inteligentes (Smart Grids) e a atualização segura de *firmware* é uma necessidade crítica para garantir que os medidores inteligentes estejam sempre em conformidade com as regulamentações e protegidos contra ameaças de segurança. A adoção de uma solução automatizada e segura para atualizações remotas de *firmware* reduz significativamente a necessidade de intervenções manuais, resultando em uma economia de custos operacionais para as concessionárias de energia e em uma maior confiabilidade do sistema elétrico como um todo.

Além disso, essa implementação tem um impacto positivo no aumento da segurança cibernética do setor. Com a crescente digitalização das redes elétricas, as ameaças cibernéticas se tornam cada vez mais presentes, e a capacidade de garantir que o *firmware* dos medidores seja atualizado de forma segura e autêntica é fundamental para mitigar riscos de ataques que possam comprometer a integridade das operações e das medições.

Por fim, a contribuição deste trabalho reside na demonstração prática de que é possível implementar uma solução de atualização de *firmware* robusta e segura utilizando o padrão DLMS/COSEM, alinhada às necessidades do setor elétrico brasileiro. A solução desenvolvida pode ser expandida e adaptada para diferentes cenários, oferecendo uma base sólida para futuras implementações e melhorias.

REFERÊNCIAS

- BAGNARA, R.; BARR, M.; HILL, P. M. **BARR-C:2018 and MISRA C:2012: Synergy Between the Two Most Widely Used C Coding Standards**. 2020.
- BARAI, G. R.; KRISHNAN, S.; VENKATESH, B. Smart metering and functionalities of smart meters in smart grid - a review. In: **2015 IEEE Electrical Power and Energy Conference (EPEC)**. [S. l.: s. n.], 2015. p. 138–145.
- BARR, M. **Embedded C Coding Standard**. Germantown: Barr Group, 2018.
- BELL LABS/LUCENT TECHNOLOGIES. **O Desenvolvimento da Linguagem C***. 1972. Disponível em: <https://www.bell-labs.com/usr/dmr/www/chistPT.html>. Acesso em: 26 ago. 2023.
- BRITTON, J. **What Is Barr-C?** Minneapolis: Perforce Software, 2022. Disponível em: <https://www.perforce.com/blog/qac/what-is-barr-c>. Acesso em: 23 ago. 2023.
- DLMS USER ASSOCIATION. **COSEM Interface Classes and OBIS Object Identification System**. [S. l.]: DLMS UA, 2020.
- DLMS USER ASSOCIATION. **DLMS/COSEM Architecture and Protocols**. [S. l.]: DLMS UA, 2020.
- DLMS USER ASSOCIATION. **The smart link between consumption and supply**: Dlms user association is a leading voice globally in interoperable and secure data exchange, providing a foundation for innovation and growth to support strategic energy and water management. Switzerland: DLMS UA, 2024. Disponível em: <https://www.dlms.com/>. Acesso em: 24 set. 2024.
- DUSA, P.; NOVAC, C.; PURICE, E.; DODUN, O.; SLĂTINEANU, L. Configuration a meter data management system using axiomatic design. **Procedia CIRP**, v. 34, p. 174–179, 2015. ISSN 2212-8271. 9th International Conference on Axiomatic Design (ICAD 2015). Disponível em: <https://www.sciencedirect.com/science/article/pii/S221282711500801X>.
- EL-HAWARY, M. E. The smart grid—state-of-the-art and future trends. **Electric Power Components and Systems**, Taylor Francis, v. 42, n. 3-4, p. 239–250, 2014. Disponível em: <https://doi.org/10.1080/15325008.2013.868558>.
- GUNGOR, V. C.; SAHIN, D.; KOCAK, T.; ERGUT, S.; BUCCELLA, C.; CECATI, C.; HANCKE, G. P. Smart grid technologies: Communication technologies and standards. **IEEE Transactions on Industrial Informatics**, v. 7, n. 4, p. 529–539, 2011.
- INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 9899:2018**: Information technology — Programming languages — C. ISO, 2020. Disponível em: <https://www.iso.org/standard/74528.html>. Acesso em: 26 ago. 2023.
- KEBOTOGETSE, O.; SAMIKANNU, R.; YAHYA, A. Review of key management techniques for advanced metering infrastructure. **International Journal of Distributed Sensor Networks**, v. 17, n. 8, p. 15501477211041541, 2021. Disponível em: <https://doi.org/10.1177/15501477211041541>.

NANSEN INSTRUMENTOS DE PRECISÃO. **NSXi**: Medidores inteligentes de energia elétrica com operação remota. Contagem: NANSEN, 2023. Disponível em: <http://nansen.com.br/medidores/nsxi/#ficha-tecnica>. Acesso em: 22 ago. 2023.

NATIONAL ENERGY TECHNOLOGY LABORATORY. **NETL Modern Grid Strategy Powering our 21st-Century Economy**: Advanced metering infrastructure. South Park: U.S. Department of Energy, 2008.

OTUOZE, A. O.; MUSTAFA, M. W.; LARIK, R. M. Smart grids security challenges: Classification by sources of threats. **Journal of Electrical Systems and Information Technology**, v. 5, n. 3, p. 468–483, 2018. ISSN 2314-7172. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2314717218300163>.

SMART ENERGY INTERNATIONAL. **Advanced metering infrastructure and smart grid**. Maarsse: SEI, 2022. Disponível em: <https://www.smart-energy.com/regional-news/asia/advanced-metering-infrastructure-and-smart-grid/>. Acesso em: 18 ago. 2023.

WALLS, C. Chapter 1 - embedded software. In: WALLS, C. (Ed.). **Embedded Software (Second Edition)**. Second edition. Oxford: Newnes, 2012. p. 1–45. ISBN 978-0-12-415822-1. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780124158221000015>.

WALLS, C. Chapter 4 - c language. In: WALLS, C. (Ed.). **Embedded Software (Second Edition)**. Second edition. Oxford: Newnes, 2012. p. 125–175. ISBN 978-0-12-415822-1. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780124158221000040>.

WALLS, C. Chapter 7 - real-time operating systems. In: WALLS, C. (Ed.). **Embedded Software (Second Edition)**. Second edition. Oxford: Newnes, 2012. p. 243–286. ISBN 978-0-12-415822-1. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780124158221000076>.