Segurança em Redes IOT: Desenvolvimento de um Protocolo Sobre TCP/IP com Criptografia Integrada

Rafael da Silva Dagostim
Centro Universitário Satc
Criciúma, Brasil
rafaeldagostim.pessoal@gmail.com

Cleber Lourenço Izidoro
Centro Universitário Satc
Criciúma, Brasil
cleber.izidoro@satc.edu.br

Ramon Venson
Centro Universitário Satc
Criciúma, Brasil
ramon.venson@satc.edu.br

Rodrigo Cesar Nunes Maciel
Centro Universitário Satc
Criciúma, Brasil
rodrigo.maciel@satc.edu.br

***Abstract*—This work proposes the development of a secure communication protocol for Internet of Things (IoT) networks, focusing on data protection in resource-constrained devices. Inspired by protocols such as MQTT, the system implements AES encryption in CBC mode to ensure the confidentiality and integrity of transmitted messages. The adopted methodology includes the development of a broker in Node.js, responsible for managing communication and cryptographic keys, and the implementation of a client in C++ for ESP32 devices using the Arduino IDE. Validation tests were conducted in a controlled environment, simulating communication scenarios between multiple devices. The results showed that the protocol efficiently ensures secure data transmission with suitable performance for IoT applications. It is concluded that the proposed solution is feasible and contributes to more secure communication in smart device networks.**

***Keywords*—Internet of Things; Information Security; AES Encryption; ESP32; Node.js; Communication Protocol; MQTT; IoT Networks; Cybersecurity; Embedded Devices.**

***Resumo*—Este trabalho propõe o desenvolvimento de um protocolo de comunicação seguro para redes de Internet das Coisas (IoT), com foco na proteção de dados em dispositivos com recursos computacionais limitados. Inspirado em protocolos como o MQTT, o sistema implementa criptografia ECC (Elliptic Curve Cryptography) e AES (Advanced Encryption Standard) para garantir a confidencialidade e integridade das mensagens transmitidas. A metodologia adotada inclui a construção de um broker em Node.js, responsável pela mediação da comunicação e gerenciamento das chaves criptográficas, e a implementação de um client em C++ para dispositivos ESP32. Os testes de validação foram realizados em ambiente controlado, simulando cenários de comunicação entre múltiplos dispositivos. Os resultados demonstraram que o protocolo é eficiente na transmissão segura de dados, com desempenho adequado para aplicações IoT. Conclui-se que a proposta é viável e pode contribuir com soluções mais seguras para a comunicação em redes de dispositivos inteligentes.**

***Palavras-chave*—Internet das Coisas; Segurança da Informação; Criptografia AES; ESP32; Node.js; Protocolo de Comunicação; MQTT; Redes IoT; Cibersegurança; dispositivos embarcados.**

I. Introdução

A crescente adoção de dispositivos inteligentes em residências, empresas e ambientes industriais tem promovido avanços significativos em termos de conveniência, automação e eficiência. Contudo, essa expansão também amplia a superfície de ataque disponível para cibercriminosos, expondo vulnerabilidades críticas relacionadas à segurança da informação. Dispositivos como câmeras de vigilância, termostatos e assistentes virtuais, frequentemente conectados à mesma rede, tornam-se alvos potenciais de ataques que podem resultar em invasões, roubo de dados e controle remoto indevido [1].

Estudo realizado pela TGT Consult, em parceria com a Associação Brasileira de Internet das Coisas (ABINC), projeta que, até 2025, haverá mais de 27 bilhões de dispositivos conectados em operação em todo o mundo. Esse crescimento exponencial reforça a urgência por soluções de segurança adaptadas à realidade desses dispositivos, geralmente limitados em capacidade de processamento e energia [2].

Embora existam diversos protocolos de comunicação empregados em ecossistemas IoT, como HTTP (Hypertext Transfer Protocol), WebSocket, CoAP (Constrained Application Protocol) e MQTT (Message Queuing Telemetry Transport), muitos deles não oferecem criptografia robusta. Quando a oferecem, frequentemente sobrecarregam os recursos dos dispositivos, comprometendo sua eficiência. Nesse contexto, torna-se essencial o desenvolvimento de soluções que conciliem segurança e desempenho.

Dessa forma, este trabalho tem como objetivo geral o desenvolvimento de um protocolo de comunicação experimental baseado na pilha TCP/IP, voltado especificamente para dispositivos IoT. O protocolo deverá integrar mecanismos de criptografia de forma eficiente, utilizando tecnologias leves e compatíveis com as limitações desses dispositivos. Busca-se, assim, garantir a segurança na troca de dados sem comprometer o desempenho, contribuindo para a padronização de métodos seguros de transmissão em ambientes embarcados.

II. REVISÃO BIBLIOGRÁFICA

Esta revisão bibliográfica aborda os principais fundamentos que sustentam o trabalho, com foco no conceito de Internet das Coisas, nos desafios da segurança da informação e nos protocolos de comunicação em dispositivos IoT.

A. Internet das Coisas (IOT)

Não existe uma definição única para Internet das Coisas (IoT). O termo foi cunhado por Kevin Ashton em 1999 e, em 2005, passou a ser formalmente reconhecido pela União Internacional de Telecomunicações (ITU) [3]. Uma definição para IoT seria uma rede de dispositivos interconectados que compartilham dados com o objetivo de alcançar uma finalidade comum [4]. Uma segunda definição afirma que os principais componentes de uma rede IoT incluem: **sensores**, que coletam dados; **identificadores**, que indicam sua origem; **software**, responsável pela análise das informações; e a **rede de comunicação**, que conecta os dispositivos entre si e com os usuários [3].

Esses elementos juntos formam o conceito de IoT: dispositivos com inteligência proporcionada por software, capazes de interagir com o ambiente e de se comunicar através de uma rede. Em última análise, a IoT busca conectar tudo e todos, permitindo que os dados coletados sejam processados e devolvidos ao ambiente de forma interativa [3]. A figura abaixo demonstra a tríade citada que define o que é IOT:

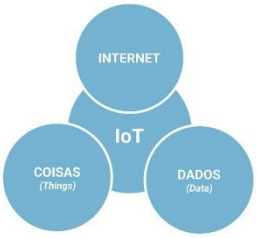

Fig. 1 Definição de IOT em um simples formato

A expansão acelerada de dispositivos inteligentes no cotidiano representa uma transformação tecnológica impulsionada por múltiplos fatores convergentes. Este crescimento exponencial é alimentado principalmente pela digitalização industrial (Indústria 4.0), redução dos custos de sensores e componentes eletrônicos, expansão das redes de conectividade 4G/5G, e pela crescente demanda por automação e eficiência operacional em diversos setores [1].

A pandemia de COVID-19 atuou como catalisador adicional, acelerando a adoção de tecnologias IoT para operações remotas, monitoramento de saúde e otimização de cadeias de suprimentos [5].

Com essa expansão, a maior parte do tráfego na internet e em outros meios de comunicação tende a ser dominada pela troca de dados máquina a máquina (M2M) [1]. A Figura 2 ilustra essa tendência de crescimento, demonstrando como os dispositivos passam a se comunicar cada vez mais entre si em detrimento da interação com usuários humanos, além de necessitarem progressivamente menos de servidores centralizados.

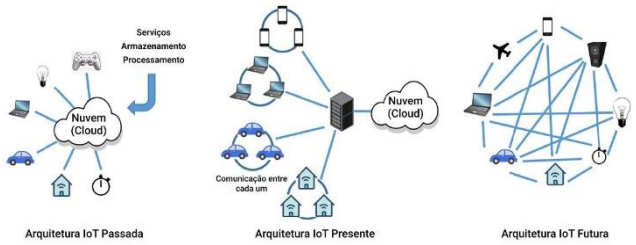

Fig. 2 Passado, presente e futuro da arquitetura IOT

Essa tendência global reflete-se em projeções que indicam a conexão de bilhões de dispositivos nos próximos anos, fundamentando a necessidade crítica de protocolos de comunicação seguros e eficientes para suportar essa infraestrutura digital emergente.

1) Camadas de Uma Arquitetura IOT

A arquitetura de sistemas IoT costuma ser dividida em camadas, cada uma com funções e desafios específicos, especialmente no que diz respeito à segurança. Embora não exista um padrão único, três camadas principais são amplamente reconhecidas [4]:

- **Camada de Hardware:** formada pelos dispositivos físicos que coletam dados do ambiente.
- **Camada de Comunicação:** trata da conexão e transmissão de dados entre dispositivos, por meio de tecnologias como rádio, 3G, 4G, LoRaWAN, entre outras.
- **Camada de Aplicação:** composta por servidores e sistemas que processam os dados coletados.

Outros autores propõem variações. [6], por exemplo, une as camadas de comunicação e aplicação em uma única camada de software e acrescenta a camada de **peopleware**, que representa o usuário como parte integrante do sistema, um ponto importante ao se considerar aspectos de segurança e interação.

B. Segurança da Informação

No meio corporativo, é irrevogável dizer que informação atualmente é um dos principais ativos de uma empresa, é com base no controle e análise destes dados que gestores e setores estratégicos tomam as decisões que acarretam o sucesso da empresa (ou o seu fracasso) [7].

Proteger esses dados contra perdas ou roubos é um trabalho constate das empresas que desejam manter-se no mercado, é dessa necessidade que surge o ramo da segurança da informação, visando encontrar os melhores métodos de mitigar danos e maximizando o retorno desse investimento

1) Princípios Fundamentais da Segurança da Informação

A gestão de segurança conta com três princípios fundamentais: aconfidencialidade, integridade e disponibilidade. Estes podem ser por diferentes mecanismos e sua ausência pode afetar negativamente um ambiente [8].

- **Confidencialidade:** garante que o acesso à informação seja restrito a indivíduos autorizados, assegurando que possuem o nível de permissão necessário para manipular os dados de forma segura.
- **Integridade:** assegura a consistência e precisão dos dados ao longo de seu ciclo de vida, incluindo armazenamento, manipulação, alteração e entrega. Qualquer modificação não autorizada constitui uma violação da integridade.
- **Disponibilidade:** assegura que os dados estejam acessíveis sempre que requisitados, o que envolve sua gestão em locais seguros e a existência de mecanismos de recuperação, como backups, para lidar com situações excepcionais, especialmente no caso de dados digitais.

[7] complementa os princípios da segurança da informação ao incluir dois aspectos importantes: **autenticidade**, que assegura a legitimidade das informações e das partes envolvidas, e **irrevogabilidade** (não repúdio), que garante a rastreabilidade das ações, impedindo que o autor negue a realização de uma transação.

2) Desafios da Segurança da Informação em IOT

Com o crescimento da Internet das Coisas (IoT) e seu uso no dia a dia, aumenta também o risco de acesso não autorizado a esses dispositivos. Garantir a segurança dos dados em dispositivos com pouca capacidade computacional é um dos maiores desafios da IoT. Isso fica visível ao consultar no site *IEEE Xplore* (pesquisa feita em 15/11/2024), que revela mais de 28 mil resultados para *"IoT security"*. A maioria são conferências (20.556) e periódicos (6.331), mas apenas 32 materiais tratam de normas e padrões, ou seja, aproximadamente 0,12% de todo o conteúdo encontrado.

Como o foco deste trabalho está na camada de comunicação em ecossistemas IoT, destacam-se alguns ataques comuns:

- **Man-in-the-Middle (MitM):** o invasor intercepta a comunicação entre dois dispositivos, podendo ler, alterar ou redirecionar os dados trocados [9].
- **Falsa Injeção de Dados:** após invadir a rede ou um dispositivo, o atacante envia dados falsos para obter informações ou causar falhas no sistema [1].
- **Interceptação de Tráfego (Sniffing):** uso de ferramentas para capturar dados transmitidos pela rede, com o objetivo de acessar informações sensíveis [6].

A segurança da camada de comunicação é vital em ecossistemas IoT, pois essa camada serve como elo para a troca de dados entre dispositivos e sistemas. A falha em proteger essa etapa pode comprometer não apenas as informações transmitidas, mas também todo o ecossistema conectado.

C. Protocolos de Comunicação

Protocolo pode ser entendido como a "linguagem" usada entre computadores para conversarem pela rede, nesse processo vários protocolos são usados, cada um com uma finalidade diferente [10].

Para protocolos de comunicação, a principal forma de organização é estabelecida em camadas. Cada camada desempenha um papel específico, desde a transmissão dos bits no nível mais básico até os protocolos mais complexos, responsáveis por identificar dispositivos e estabelecer padrões de comunicação.

A estrutura de camadas mais conhecida é a definida pelo modelo OSI *(Open Systems Interconnection)*, que divide a comunicação em sete camadas lógicas [11]. A figura abaixo apresenta a sequência dessas camadas, juntamente com as responsabilidades atribuídas a cada uma delas:

Fig. 3 Modelo de 7 camadas especificadas no padrão OSI

Dentro da camada de aplicação do modelo TCP/IP, destacam-se diversos protocolos utilizados em ecossistemas IoT, entre eles: **HTTP**, **WebSocket**, **AMQP** e **MQTT**. A escolha do protocolo de comunicação depende das necessidades específicas de cada aplicação, como confiabilidade, consumo de energia, latência, largura de banda e segurança. O uso de protocolos otimizados e bem implementados é fundamental para garantir que os dispositivos IoT possam se comunicar de maneira eficaz, segura e escalável, contribuindo para o crescimento e a confiabilidade das redes inteligentes.

D. Fundamento de Criptografia

A criptografia constitui um conjunto de técnicas matemáticas e computacionais destinadas a proteger informações através da transformação de dados legíveis em formato cifrado, garantindo confidencialidade, integridade e autenticidade durante o armazenamento e transmissão [12]. No contexto de sistemas IoT, onde dispositivos com recursos limitados necessitam comunicar-se de forma segura, a escolha adequada dos algoritmos criptográficos torna-se fundamental para equilibrar segurança e desempenho.

1) Criptografia Assimétrica

A criptografia assimétrica, também conhecida como criptografia de chave pública, baseia-se no uso de pares de chaves matematicamente relacionadas: uma chave pública, que pode ser compartilhada livremente, e uma chave privada, mantida em sigilo pelo proprietário [13]. Este modelo resolve

o problema fundamental da distribuição segura de chaves, permitindo que duas partes estabeleçam comunicação segura sem necessidade de compartilhamento prévio de segredos.

O algoritmo RSA (Rivest-Shamir-Adleman) representa um dos sistemas criptográficos assimétricos mais amplamente utilizados, baseando sua segurança na dificuldade computacional de fatorar números inteiros grandes [14]. Contudo, para dispositivos IoT, o RSA apresenta limitações significativas em termos de consumo de recursos computacionais e tamanho de chaves necessárias para garantir níveis adequados de segurança.

A Criptografia de Curva Elíptica (ECC) emerge como alternativa mais eficiente para ambientes com recursos limitados, oferecendo níveis de segurança equivalentes ao RSA com chaves substancialmente menores [15]. Por exemplo, uma chave ECC de 256 bits proporciona segurança comparável a uma chave RSA de 3072 bits, resultando em operações mais rápidas e menor consumo de memória.

2) Criptografia Simétrica

Na criptografia simétrica, a mesma chave é utilizada tanto para cifrar quanto para decifrar informações, exigindo que emissor e receptor compartilhem previamente esta chave secreta [12]. Este modelo oferece desempenho superior em comparação aos algoritmos assimétricos, sendo adequado para criptografia de grandes volumes de dados.

O Advanced Encryption Standard (AES) constitui o padrão de criptografia simétrica adotado pelo governo americano e amplamente utilizado em aplicações comerciais [16]. O AES opera com chaves de 128, 192 ou 256 bits e suporta diferentes modos de operação, sendo o modo CBC (Cipher Block Chaining) particularmente relevante por proporcionar proteção adicional contra padrões nos dados cifrados através do uso de um vetor de inicialização.

3) Transport Layer Security (TLS)

O Transport Layer Security (TLS) representa um protocolo criptográfico amplamente empregado para estabelecer comunicações seguras sobre redes não confiáveis, operando na camada de transporte do modelo TCP/IP [17]. O TLS combina criptografia assimétrica para estabelecimento seguro de sessão com criptografia simétrica para proteção eficiente dos dados transmitidos.

O processo de handshake TLS envolve múltiplas etapas, incluindo negociação de algoritmos criptográficos, verificação de certificados digitais e estabelecimento de chaves de sessão [18]. Embora efetivo para garantir segurança em comunicações web, o overhead computacional e de rede do TLS pode representar limitações significativas para dispositivos IoT com recursos restritos, especialmente considerando o consumo adicional de energia e largura de banda necessários para o handshake completo.

III. Procedimento Experimental

Este capítulo descreve o processo de concepção, pesquisa, desenvolvimento, dificuldades enfrentadas e validação do protocolo SMQP (Secure-based Message Queuing Protocol).

A. Concepção do Protocolo SMQP

O protocolo SMQP foi concebido como uma resposta às limitações de segurança identificadas nos protocolos de comunicação IoT existentes. Inspirado na eficiência e simplicidade do MQTT, o SMQP foi projetado com o objetivo de integrar criptografia nativa diretamente no protocolo, eliminando a dependência de camadas externas de segurança como o TLS.

A proposta inicial baseava-se em uma arquitetura broker-cliente, onde o broker centralizaria a comunicação e gerenciaria a distribuição segura de chaves criptográficas entre os dispositivos conectados. A visão era desenvolver um sistema que combinasse diferentes técnicas criptográficas para garantir tanto a segurança na troca de chaves quanto a eficiência na proteção das mensagens transmitidas.

O protocolo foi idealizado para implementar segurança granular por tópico, permitindo que cada canal de comunicação mantivesse seu próprio contexto de segurança independente. Esta abordagem visava proporcionar isolamento entre diferentes fluxos de dados e facilitar o controle de acesso em sistemas multi-usuário.

Desde o início, o desenvolvimento foi orientado pela necessidade de compatibilidade com dispositivos embarcados como o ESP32, reconhecendo as severas limitações de memória, processamento e energia desses ambientes. O desafio central consistia em criar um protocolo que oferecesse robusta proteção criptográfica sem comprometer a viabilidade operacional em dispositivos com recursos computacionais restritos.

B. Desafios e Problemas Encontados

Durante o desenvolvimento do protocolo SMQP, diversos desafios técnicos e conceituais foram enfrentados. Desde a definição da arquitetura criptográfica até a implementação em dispositivos com recursos limitados, cada etapa exigiu adaptações importantes para garantir segurança, desempenho e viabilidade prática. Os tópicos a seguir detalham os principais problemas encontrados e as soluções adotadas ao longo do projeto.

1) Limitações de Desempenho do RSA e Alternativa Com ECC

Durante a fase inicial, a criptografia RSA foi utilizada para troca de chaves. No entanto, foi rapidamente identificada como inadequada para o ESP32 devido ao tempo de processamento (200-500ms) e ao uso excessivo de memória (512 bytes por chave). Isso comprometia a responsividade do sistema e o consumo de energia.

Como solução, foi adotado o uso de criptografia de curva elíptica (ECC), especificamente a curva **secp256k1**, que fornece segurança equivalente ao RSA-2048 com desempenho muito superior:

- Operações até 10x mais rápidas
- Uso de memória 8x menor
- Suporte nativo à aceleração de hardware no ESP32 (via mbedTLS)

ECC é uma forma moderna de criptografia assimétrica que permite trocas seguras de chaves usando curvas matemáticas eficientes. No protocolo, ela é utilizada para criptografar as chaves AES dos tópicos.

2) Troca de Chaves Vulneráveis

No design inicial, a chave AES do tópico era enviada em texto simples no comando SUBACK, deixando o processo de distribuição de chaves exposto a ataques do tipo Man-in-the-Middle. Para resolver isso, o cliente passou a gerar um par de chaves ECC (uma pública e uma privada).

A chave pública é enviada ao broker durante o CONNECT, e o broker a armazena temporariamente. Quando o cliente se inscreve em um tópico (SUBSCRIBE), o broker utiliza essa chave pública ECC para criptografar a chave AESdo tópico antes de enviá-la ao cliente. Dessa forma, somente o cliente que possui a chave privada correspondente poderá descriptografar a chave do tópico, garantindo sigilo e autenticidade.

3) Modelo de Publicação e Assinatura

Inicialmente não estava claro como os publicadores obteriam a chave do tópico. O protocolo foi então unificado: todos os clientes devem primeiro fazer SUBSCRIBE ao tópico desejado, mesmo que o objetivo seja apenas publicar. O broker retorna a chave criptografada com a ECC do cliente. Essa chave é então usada tanto para publicar quanto para assinar mensagens.

C. Melhorias e Arquitetura Final

Após os ajustes, o protocolo foi reestruturado com as seguintes características principais:

- Cabeçalho fixo de 20 bytes: compatível com sistemas embarcados.
- ECC-256 (secp256k1) para troca de chaves.
- AES-256-CBC para criptografia de mensagens.
- Chaves por tópico geradas e gerenciadas pelo broker.
- Verificação de integridade com CRC-16.
- Modelo subscribe-to-publish com segurança unificada.

1) Implementação da Rotação de Chaves

A rotação automática de chaves a cada 3 minutos é implementada através de um timer interno no broker que monitora o tempo de vida das chaves AES por tópico. Quando o intervalo é atingido, o broker executa automaticamente os seguintes passos:

1. Geração de nova chave AES-256 para o tópico específico.
2. Criptografia da nova chave com a chave pública ECC de cada cliente inscrito.
3. Envio do comando ROTATEKEY (0x0B) para todos os clientes do tópico.
4. Confirmação via KEYACK (0x0C) pelos clientes após validação da nova chave.
5. Atualização do cache interno do broker com a nova chave.

Durante a transição, o broker mantém temporariamente ambas as chaves (antiga e nova) por 30 segundos para processar mensagens em trânsito. Após este período, a chave antiga é descartada, implementando efetivamente forward secrecy no protocolo.

D. Fluxograma de Funcionamento

Para facilitar a compreensão visual da comunicação entre os componentes, os fluxos principais estão representados nos fluxogramas a seguir:

1) Processo de Conexão

Este fluxo representa o processo de conexão inicial entre o cliente (ESP32) e o broker. O cliente gera seu par de chaves ECC e envia a chave pública no pacote CONNECT. O broker salva a chave e responde com um CONNACK confirmando a conexão:

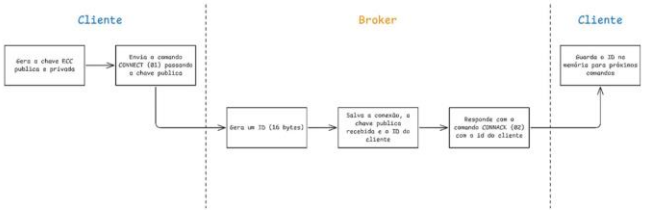

Fig. 4 Fluxo de conexão entre cliente e broker

2) Processo de Inscrição em Tópico

Neste fluxo, o cliente solicita inscrição em um tópico através do comando SUBSCRIBE. O broker gera uma chave AES exclusiva para o tópico, criptografa-a com a chave pública do cliente e responde com SUBACK. O cliente então descriptografa a chave e a armazena localmente.

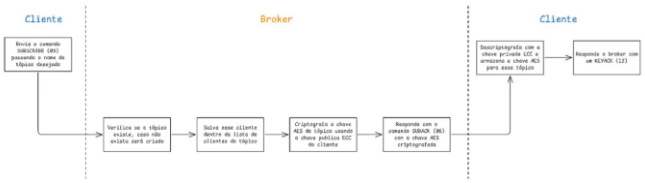

Fig. 5 Fluxo de inscrição no tópico

3) Processo de Publicação de Mensagem

Uma vez com a chave AES do tópico armazenada, o cliente pode publicar mensagens criptografadas. O broker repassa essas mensagens para os assinantes do tópico, que usam sua própria chave para descriptografar o conteúdo.

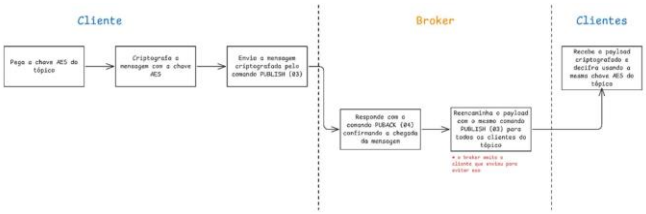

Fig. 6 Fluxo de publicação de mensagem

E. Estrutura dos Pacotes

Cada comando possui sua estrutura padronizada para envio dos dados, para que a comunicação entre cliente-broker ocorra, é preciso que todos estejam alinhados com os dados a serem enviados.

1) Estrutura do Cabeçalho

Todo pacote transmitido entre cliente e broker utiliza um cabeçalho fixo de 20 bytes. Esse cabeçalho garante a identificação do comando, o cliente responsável e o tamanho do conteúdo transmitido no payload.

O cabeçalho é composto por:

- **Byte 0 –** Versão: versão do protocolo, atualmente fixa em 0x00.
- **Byte 1 –** Codificação + Código do comando: os dois bits mais significativos definem o tipo de codificação (00 para UTF-8, 01 para Base64), e os seis bits menos significativos representam o código do comando.
- **Bytes 2 a 17 –** ID do cliente: identificador único de 16 bytes (gerado a partir do MAC + aleatoriedade).
- **Bytes 18 e 19 –** Tamanho do payload: valor em Big Endian (16 bits), indicando a quantidade de bytes do corpo da mensagem.

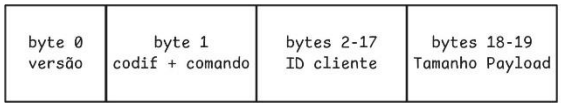

Fig. 7 Estrutura do Cabeçalho

2) Comando CONNECT (0x01)

Função: solicitar a conexão com o broker, enviando um tempo de keep-alive e a chave pública ECC do cliente.

Estrutura do payload:

- Primeiros 4 bytes: intervalo de keep-alive em segundos (formato Big Endian).
- Bytes restantes: chave pública ECC no formato comprimido, com 66 caracteres hexadecimais (33 bytes).

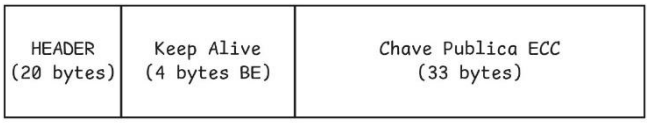

Fig. 8 Estrutura do Comando CONNECT

3) Comando CONNACK (0x02)

Função: confirmar que a conexão foi aceita pelo broker.

Estrutura do payload:

- Contém apenas o ID do cliente, como uma string hexadecimal de 32 caracteres (16 bytes brutos).

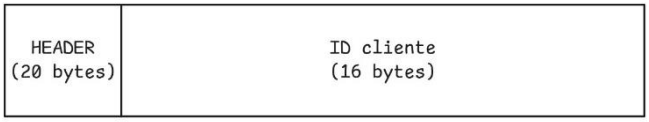

Fig. 9 Estrutura do Comando CONNACK

4) Comando PUBLISH (0x03)

Função: enviar uma mensagem criptografada para um tópico específico.

Estrutura do payload:

- 2 bytes iniciais: tamanho do nome do tópico (Big Endian).
- Nome do tópico em seguida, codificado em UTF-8.
- 16 bytes: ID do cliente remetente (brutos).
- 2 bytes: CRC-16 da mensagem original.
- 16 bytes: vetor de inicialização (IV) utilizado na criptografia AES.
- Restante: conteúdo criptografado com AES-256-CBC.

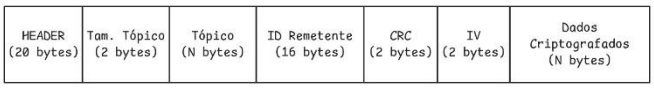

Fig. 10 Estrutura do Comando PUBLISH

5) Comando SUBSCRIBE (0x05)

Função: solicitar a inscrição em um tópico.

Estrutura do payload:

- Contém apenas o nome do tópico desejado, codificado em UTF-8 (sem prefixo de tamanho).

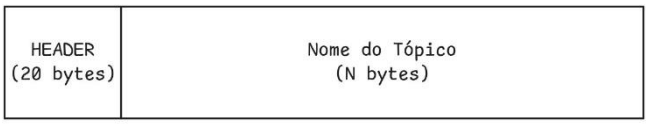

Fig. 11 Estrutura do Comando SUBSCRIBE

6) Comando SUBACK (0x06) e ROTATEKEY (0x0B)

Função do SUBACK: confirmar a inscrição em um tópico, fornecendo a chave AES criptografada.

Função do ROTATEKEY: informar uma nova chave AES para o tópico, garantindo sigilo futuro.

Estrutura do payload (para ambos):

- Nome do tópico, codificado em UTF-8.
- Símbolo separador (|).
- Chave AES criptografada utilizando ECIES (criptografia assimétrica com ECC), representada como texto hexadecimal.

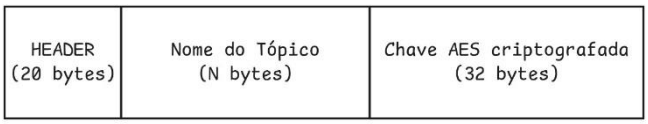

Fig. 12 Estrutura do Comando SUBACK e ROTATEKEY

7) Comando UNSUBSCRIBE (0x07)

Função: solicitar o cancelamento da inscrição em um tópico.

Estrutura do payload:

- Nome do tópico, codificado em UTF-8.

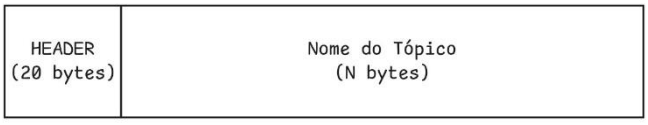

Fig. 13 Estrutura do Comando UNSUBSCRIBE

8) Comando KEYACK (0x0C)

Função: confirmar que a chave AES do tópico foi validada com sucesso após descriptografar um teste.

Estrutura do payload:

- 2 bytes: comprimento do nome do tópico (Big Endian).

- Nome do tópico, codificado em UTF-8.
- 4 bytes: CRC-32 da mensagem criptografada.
- Resto: texto "OK" criptografada com a chave AES do tópico e codificada em Base64.

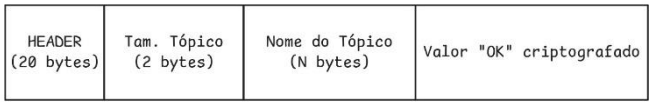

Fig. 14 Estrutura do Comando KEYACK

9) Comando DISCONNECT (0x0D)

Função: encerrar a conexão com o broker de forma elegante.

Estrutura do payload:

- Mensagem de texto opcional informando o motivo da desconexão, codificada em UTF-8.

Fig. 15 Estrutura do Comando DISCONNECT

10) Comandos Sem Payload

Alguns comandos utilizam apenas o cabeçalho, sem corpo de mensagem. Nestes casos, o campo de tamanho do payload é igual a zero (0x0000). São eles:

- **PUBACK (0x04):** confirmação de recebimento da mensagem.
- **PINGREQ (0x09):** ping de keep-alive, enviado periodicamente pelo cliente.
- **PINGRESP (0x0A):** resposta do broker ao ping.
- **UNSUBACK (0x08):** confirmação de cancelamento da inscrição em um tópico.

IV. Resultados e Discussão

O desenvolvimento do protocolo SMQP (Secure-based Message Queuing Protocol) resultou em uma implementação completa e funcional que demonstra significativos avanços em relação aos protocolos IoT tradicionais. A arquitetura final implementada apresenta características únicas que combinam segurança nativa, eficiência computacional e compatibilidade com dispositivos de recursos limitados

A. Arquitetura Final Implementada

A versão final do protocolo SMQP é composta por três componentes integrados: o broker em Node.js e um cliente (Typescript) e outro para ESP32 em C++. O broker centraliza a comunicação, roteia mensagens e distribui chaves criptográficas com eficiência linear O(n) no número de conexões, assegurando escalabilidade em ambientes produtivos.

O cliente TypeScript disponibiliza uma API similar à do MQTT, com suporte nativo a ECC e AES, além de funcionalidades como reconexão automática, gerenciamento de tópicos e tratamento robusto de erros, facilitando o uso por desenvolvedores.

Já o cliente ESP32, escrito em C++, foi otimizado para sistemas embarcados e utiliza aceleração de hardware para operações criptográficas. Durante os testes, apresentou estabilidade e uso de memória inferior a 25% da RAM total (254KB livres de 320KB), confirmando sua viabilidade para dispositivos com recursos limitados.

B. Características Técnicas Alcançadas

A evolução do protocolo consolidou decisões arquitetônicas que equilibram segurança e desempenho em dispositivos embarcados. O cabeçalho fixo de 20 bytes, embora maior que o do MQTT em mensagens pequenas, simplifica o parsing e elimina a necessidade de interpretar headers variáveis, sendo ideal para ambientes com restrições de processamento.

A criptografia ECC-256 baseada na curva secp256k1 aproveita a aceleração de hardware do ESP32 e garante compatibilidade com padrões consolidados. Para os dados, utiliza-se AES-256-CBC combinado com CRC-16, formando uma base robusta de segurança adequada a aplicações IoT críticas.

O uso de chaves AES por tópico oferece isolamento granular entre canais, facilitando o controle de acesso em sistemas multi-tenant e permitindo revogação seletiva sem afetar todo o sistema. Além disso, a rotação automática de chaves a cada 3 minutos implementa forward secrecy, protegendo mensagens passadas mesmo em caso de comprometimento de chaves futuras.

C. Análise de Desempenho

Uma das descobertas mais significativas do projeto relaciona-se à dramática melhoria de desempenho obtida através da migração de RSA para ECC. Durante as fases iniciais de implementação, as operações RSA-2048 apresentavam tempos de processamento entre 200-500ms no ESP32, valores que se mostraram incompatíveis com aplicações em tempo real e dispositivos alimentados por bateria. A transição para ECC-256 resultou em tempos de operação entre 20-50ms, representando uma melhoria de performance de ordem de magnitude que viabiliza aplicações anteriormente impraticáveis.

A redução no uso de memória de 512 bytes para 64 bytes por conjunto de chaves demonstrou ser crucial para dispositivos com severas limitações de RAM. No contexto do ESP32, com seus 320KB de RAM disponível, esta otimização possibilita o suporte simultâneo a significativamente mais tópicos criptografados, expandindo as possibilidades de aplicação do protocolo.

O aproveitamento da aceleração de hardware mbedTLS do ESP32 revelou-se um fator multiplicador de eficiência. As medições indicam que as operações criptográficas consomem menos de 5% do tempo de CPU disponível, deixando recursos substanciais para lógica de aplicação. Esta característica emergiu como fundamental para a viabilidade comercial da solução, permitindo que desenvolvedores implementem funcionalidades complexas sem comprometer a segurança.

D. Eficiência do Protocolo em Operação

A análise detalhada do overhead do protocolo revelou características específicas que impactam diretamente a eficiência da transmissão em redes IoT. O protocolo SMQP

utiliza um cabeçalho fixo de 20 bytes, valor superior aos 2-14 bytes variáveis do MQTT, porém esta decisão arquitetural facilita significativamente o parsing em dispositivos embarcados e elimina a necessidade de processamento complexo de headers de tamanho variável.

A encriptação AES adiciona entre 16-32 bytes por mensagem devido ao padding PKCS7 (Public-Key Cryptography Standards #7), overhead que deve ser considerado no contexto das mensagens IoT típicas. Considerando que mensagens IoT usuais possuem entre 50-200 bytes de payload, este overhead representa aproximadamente 8-16% do tamanho total da mensagem, valor que se mostra aceitável considerando os benefícios substanciais de segurança obtidos.

Durante os testes de latência realizados em redes locais, o protocolo demonstrou latência consistente inferior a 10ms, valor comparável ao MQTT padrão em cenários similares. Esta performance indica que a adição de criptografia nativa não introduz penalidades significativas de latência quando comparada a protocolos não criptografados, contradizendo preocupações iniciais sobre overhead de processamento.

E. Escalabilidade e Arquitetura do Broke

O design arquitetural do broker com chaves compartilhadas por tópico demonstrou vantagens significativas em termos de escalabilidade. Testes com múltiplos clientes simultâneos revelaram que o broker mantém operações criptográficas O(1) por mensagem, uma vez que não necessita descriptografar e re-criptografar mensagens durante o processo de roteamento. Esta característica permite que o sistema escale linearmente com o número de clientes sem degradação exponencial de performance.

A implementação de tópicos com chaves dedicadas revelou-se não apenas benéfica do ponto de vista de segurança, mas também eficiente em termos computacionais. O broker mantém um cache de chaves ativas em memória, evitando operações criptográficas redundantes e permitindo roteamento eficiente mesmo com centenas de tópicos simultâneos ativos.

F. Análise Comparativa Com Protocolos Estabelecidos

Para avaliar a eficácia do protocolo SMQP, é relevante compará-lo com soluções consolidadas no ecossistema IoT. Protocolos como MQTT e HTTP/HTTPS são amplamente utilizados, mas apresentam limitações em termos de segurança nativa e eficiência para dispositivos com recursos restritos. Os subtópicos a seguir exploram essas diferenças, considerando aspectos de desempenho, arquitetura de comunicação e resiliência a ataques.

1) SMQP versus MQTT: Segurança e Eficiência

A comparação entre SMQP e MQTT revela trocas fundamentais entre segurança nativa e compatibilidade com infraestrutura existente. O MQTT tradicional depende de TLS externo para segurança, introduzindo overhead significativo de handshake (traduzido como aperto de mão, é forma para descrever o reconhecimento dos dispositivos e início da comunicação) TLS que tipicamente consome 1-2KB de dados e requer múltiplos round-trips (idas e vindas, nesse caso a troca de confirmações) para estabelecimento de conexão segura. Em contraste, o SMQP integra segurança diretamente no protocolo, eliminando a complexidade de configuração TLS e permitindo controle granular de criptografia por tópico.

O estabelecimento de conexão também difere substancialmente entre os protocolos. MQTT com TLS requer handshake TCP tradicional de 3 round-trips seguido por handshake TLS de 2-3 round-trips adicionais, totalizando 5-6 round-trips para estabelecimento de conexão segura. O SMQP simplifica este processo combinando handshake TCP de 3 round-trips com troca de chaves ECC em apenas 1 round-trip adicional, reduzindo o overhead de estabelecimento de conexão.

Em relação a eficiência do cabeçalho, MQTT utiliza headers variáveis de 2-14 bytes dependendo do tipo de mensagem e tamanho de identificadores, enquanto SMQP emprega header fixo de 20 bytes. Embora o SMQP apresente overhead ligeiramente maior para mensagens pequenas, oferece parsing significativamente mais eficiente em dispositivos embarcados, eliminando a necessidade de processamento complexo de headers variáveis.

2) SMQP versus HTTP/HTTPS: Paradigmas de Comunicação

A comparação com HTTP revela diferenças fundamentais nos paradigmas de comunicação adequados para IoT. HTTP utiliza headers textuais que tipicamente consomem 200-800 bytes por requisição, representando overhead substancial comparado aos 20 bytes do header binário SMQP. Esta diferença torna-se particularmente relevante em dispositivos com limitações severas de largura de banda ou custos de transmissão por byte.

O modelo de comunicação também difere radicalmente entre os protocolos. HTTP opera em paradigma request-response síncrono que se mostra inadequado para notificações push e comunicação M2M em tempo real, características fundamentais em aplicações IoT modernas. O SMQP implementa comunicação publish-subscribe assíncrona que se alinha naturalmente com os padrões de comunicação IoT, onde dispositivos frequentemente precisam reagir a eventos externos em tempo real.

Em termos de uso de recursos no ESP32, bibliotecas HTTP/HTTPS típicas consomem 40-60KB de RAM devido à complexidade do parsing de headers textuais e gestão de estado HTTP. A implementação customizada do SMQP consome aproximadamente 30KB de RAM, liberando recursos valiosos para lógica de aplicação. Esta eficiência de recursos torna-se crítica em dispositivos com apenas 320KB de RAM total disponível.

3) Análise de Segurança em Contexto

O protocolo SMQP oferece vantagens específicas em relação ao MQTT tradicional, especialmente em ambientes IoT. Ele protege contra ataques Man-in-the-Middle por meio de criptografia ECIES e validação de chaves públicas, eliminando a necessidade de infraestrutura externa como o TLS. Em ataques de análise de tráfego, o SMQP se destaca ao criptografar mensagens por tópico, dificultando a correlação entre comunicações. Além disso, sua granularidade permite revogar acessos individualmente sem

comprometer todo o sistema, oferecendo uma camada extra de segurança mesmo em comparação ao MQTT com TLS.

G. Validação em Ambiente Operacional

A implementação no ESP32 representa a validação mais rigorosa do protocolo, testando todos os componentes em ambiente com restrições reais de recursos. A geração de chaves ECC foi verificada utilizando aceleração de hardware mbedTLS, demonstrando compatibilidade com o formato de chaves comprimidas requerido pelo broker TypeScript.

A gestão de memória durante operação normal revelou uso eficiente de recursos, com o ESP32 mantendo consistentemente 254KB de heap livre, representando 79% da capacidade total disponível. Esta eficiência permite que aplicações complexas operem simultaneamente com o cliente de comunicação segura, característica fundamental para viabilidade comercial da solução.

H. Limitações e Desafios Técnicos

Esta seção explora as principais limitações encontradas durante o desenvolvimento do protocolo, os desafios técnicos que foram superados no processo e as oportunidades identificadas para aprimoramentos futuros. Esses aspectos são essenciais para compreender os limites atuais da solução e seu potencial de evolução.

1) Limitações Arquiteturais Identificadas

Durante o desenvolvimento e operação do sistema, emergiram limitações inerentes à abordagem escolhida que merecem discussão detalhada. A incompatibilidade com infraestrutura MQTT existente representa a limitação mais significativa para adoção em larga escala, exigindo migração completa de sistemas implementados e retreinamento de equipes técnicas familiarizadas com MQTT.

O overhead do cabeçalho fixo de 20 bytes torna-se desvantajoso para mensagens extremamente pequenas, comuns em sensores IoT simples que transmitem apenas valores numéricos. Para mensagens inferiores a 50 bytes, o overhead relativo pode exceder 40%, valor significativo em aplicações com severas restrições de largura de banda ou custos de transmissão por byte.

2) Desafios Técnicos Superados

O desenvolvimento revelou desafios técnicos substanciais que exigiram soluções inovadoras e iterações múltiplas. A compatibilidade criptográfica cross-platform emergiu como o desafio mais complexo, exigindo padronização cuidadosa entre implementações JavaScript e C++ de algoritmos ECIES. Diferenças sutis em formatos de chave, algoritmos de derivação e sequências de dados para HMAC exigiram múltiplas iterações de refinamento até alcançar compatibilidade completa.

A gestão de memória no ESP32 para operações criptográficas complexas exigiu otimizações específicas que incluíram uso de buffers compartilhados, limpeza cuidadosa de contextos criptográficos e implementação de estratégias de yield para evitar timeouts de watchdog. Durante o desenvolvimento inicial, operações criptográficas executadas em construtores causavam boot loops que foram resolvidos através da divisão da inicialização em etapas gerenciáveis.

3) Oportunidades Para Evolução Futura

A arquitetura atual oferece fundação sólida para extensões futuras que poderiam endereçar limitações identificadas. A implementação de autenticação mútua broker-cliente através de certificados digitais ou Pre-Shared Keys eliminaria vulnerabilidades residuais relacionadas à autenticação unidirecional atual.

A adição de níveis QoS (Quality of Service) similares aos do MQTT proporcionaria garantias de entrega configuráveis, característica importante para aplicações críticas que requerem confirmação de recebimento. A implementação de compressão de dados para mensagens grandes poderia reduzir overhead de rede, especialmente relevante para dispositivos com conexões limitadas ou custos de transmissão elevados.

V. CONCLUSÃO

O projeto atingiu com sucesso seu objetivo de desenvolver um protocolo de comunicação seguro e eficiente para dispositivos IoT com recursos computacionais limitados. A proposta de criptografia nativa, baseada na troca de chaves via ECC e na criptografia de mensagens com AES-256-CBC, mostrou-se eficaz ao superar as limitações típicas de protocolos que dependem do uso de TLS.

A implementação completa do SMQP, composta por broker em TypeScript, cliente TypeScript e cliente ESP32 em C++, validou a viabilidade da solução em ambientes heterogêneos. O foco na otimização de recursos, especialmente no ESP32, permitiu manter elevados níveis de segurança com baixo consumo de memória, CPU e energia.

As principais decisões arquitetônicas, como a substituição do RSA pela ECC, o modelo unificado de publicação/assinatura (publish/subscribe) e o gerenciamento centralizado das chaves por tópico, resultaram em um protocolo que equilibra segurança, desempenho e usabilidade. A adoção de práticas como a perfect forward secrecy, obtida por meio da rotação periódica de chaves, eleva o nível de proteção oferecido.

A proposta representa uma contribuição relevante para o campo da segurança IoT, oferecendo não apenas uma solução funcional, mas também um conjunto de boas práticas e aprendizados aplicáveis a outras implementações seguras. O código-fonte e a documentação desenvolvidos se configuram como recursos úteis para pesquisadores e desenvolvedores da área.

O protocolo apresenta aplicabilidade concreta em diferentes contextos. Em ambientes de smart home, possibilita comunicação segura entre dispositivos sem exigir infraestrutura de segurança complexa. Em cenários industriais, oferece transmissão confiável de dados sensoriais mesmo em dispositivos com severas restrições de energia e conectividade. Em aplicações de edge computing, seu baixo uso de recursos o torna adequado às exigências de processamento distribuído com foco em segurança.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus e à minha família, em especial aos meus pais Marília e Sandro, à minha irmã Larissa e aos meus avós Teresa e Hilário, pelo imenso apoio, incentivo aos estudos e presença constante em todos os momentos da vida. Agradeço também aos meus amigos Eduardo e Tairine, que tive o prazer de conhecer durante esta jornada acadêmica e que me motivaram, ajudaram e apoiaram ao longo do caminho. Sou grato ainda aos professores, pelos ensinamentos, companheirismo e pela dedicação que contribuíram diretamente para a conclusão deste trabalho com excelência. Por fim, agradeço a todos que, de alguma forma, fizeram parte dessa jornada, direta ou indiretamente. Cada gesto, palavra e apoio teve um papel importante nesta etapa da minha vida.

REFERÊNCIAS

[1] V. Hassija , et al., "A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures," jun. 20, 2019.

[2] L. G. Pacete, "IoT: até 2025, mais de 27 bilhões de dispositivos estarão conectados," forbes.com.br, 11 ago. 2022, disponível em: https://forbes.com.br/forbes-tech/2022/08/iot-ate-2025-mais-de-27-bilhoes-de dispositivos-estarao-conectados/, acesso em: 02 nov. 2024.

[3] A. Rayes and S. Salam, Internet of Things From Hype to Reality: The Road to Digitization, [S.l.]: Springer, 2018.

[4] M. Adam et al., "A Survey on Security, Privacy, Trust, and Architectural Challenges in IoT Systems," mar. 28, 2024.

[5] Ubisense, "Ubisense," 2025, disponível em: https://ubisense.com/a-rapid-increase-in-iot-adoption-manufacturing-iot-in-2023/, acesso em: 24 jun. 2025.

[6] R. G. Berlanda, "Guia de segurança da informação para a conectividade de dispositivos IoT," IFSC – Instituto Federal de Santa Catarina, pp. 22–29, 2021.

[7] M. D. C. Galvão, Fundamentos em segurança da informação, 1ª ed., São Paulo: Pearson, 2015.

[8] J. Hintzbergen. et al., Fundamentos de Segurança da Informação: com Base na ISO 27001 e na ISO 27002, 1ª ed., [S.l.]: Brasport, 2018.

[9] D. Q. D. Andrade and G. C. S. dos Santos, "Ataque de Homem do Meio em Aplicações de Realidade Virtual," UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO, jul. 2018.

[10] G. Torres, Rede de Computadores, 2ª ed., [S.l.]: Clube do Hardware, 2018.

[11] K. R. Fall and W. R. Stevens, TCP/IP Illustrated, Volume 1: The Protocols, 1ª ed., [S.l.]: Addison-Wesley Professional, v. 1, 2012.

[12] W. Stallings, Cryptography and Network Security: Principles and Practice, [S.l.]: Pearson, 2013.

[13] A. J. Menezes, P. C. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, [S.l.]: CRC Press, 1996.

[14] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120–126, 1978.

[15] D. Hankerson, A. Menezes, and S. Vanstone, Guide to Elliptic Curve Cryptography, New York: Springer-Verlag, 2004.

[16] J. Daemen and V. Rijmen, The Design of Rijndael: AES The Advanced Encryption Standard, [S.l.]: Springer, 2002.

[17] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, ago. 2018, disponível em: https://www.rfc-editor.org/info/rfc8446.

[18] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, ago. 2008, disponível em: https://www.rfc-editor.org/info/rfc5246.