Implementação e avaliação de protótipo de sistema de monitoramento de baixo custo utilizando dispositivos sem fio IoT, criptografia simétrica AES e protocolo de comunicação MQTT

Daniel Gleison Moreira Lira
Mestrado Acadêmico em Ciência da Computação - Universidade Estadual do Ceará (UECE)
Fortaleza, CE – Brasil
daniel.gleison@aluno.uece.br

Resumo:
Atualmente, em decorrência da facilidade de comunicação entre diversos equipamentos utilizando a internet, vimos o rápido crescimento da oferta de sistemas de monitoramento, especialmente em smartphones, onde dados sensíveis de usuários e empresas trafegam pela rede sem fio, muitas vezes, sem a garantia de disponibilidade, integridade e confidencialidade, que são pilares da segurança da informação. Um sistema de monitoramento IoT, além das premissas de baixo custo e arquitetura aberta, deve ser capaz de garantir a comunicação segura e eficiente entre os clientes e servidores, baseados em algoritmos criptográficos robustos e protocolos confiáveis. A proposta desse trabalho consiste na implementação e avaliação de protótipo de sistema de monitoramento web utilizando dispositivos sem fio Raspberry e Esp8266, criptografia simétrica AES (*Advanced Encryption Standard*) e protocolo de comunicação MQTT (*Message Queue Telemetry Transport*). No protótipo desenvolvido foram avaliados o tempo de processamento de encriptação e decriptação no Esp8266 utilizando o algoritmo AES, tempo de transmissão via MQTT para o Raspberry, bem como o consumo de memória de programa e de dados do Esp8266. Para monitoramento em tempo real dos dados, foram utilizados os aplicativos MQTT Box (extensão Chrome), MQTTool (aplicativo Android/iOS) e Elipse E3 (sistema supervisório SCADA). De acordo com os resultados, o tempo total de encriptação, decriptação e publicação do ESP8266 não representa impacto significativo no tráfego de dado, tão pouco o processamento exigido para o Esp8266. Diante do exposto, reforça-se a viabilidade de utilização de sistema de monitoramento IoT de baixo custo em aplicações que requerem segurança, disponibilidade e confiabilidade.
Repositório do projeto: https://github.com/danielgleison/IoT

Palavras-chave: IoT, Internet das Coisas, criptografia, AES, MQTT, Raspberry, Esp8266.

1. INTRODUÇÃO

Atualmente, em decorrência da facilidade de comunicação entre diversos equipamentos utilizando a internet, vimos o rápido crescimento da oferta de sistemas de monitoramento, especialmente em smartphones, onde dados sensíveis de usuários e empresas trafegam pela rede sem fio, muitas vezes, sem a garantia de disponibilidade, integridade e confidencialidade, que são pilares da segurança da informação.

Destaca-se que a evolução desses equipamentos é muito mais rápida do que a capacidade de domínio da solução, o que dificultam o estudo, documentação e implementação de mecanismos para detecção de ameaças e correção de defeitos. Nesse momento, é importante registrar que todos os equipamentos, inclusive apenas alterações de hardware, com conectividade em rede pública devem ser homologados pela ANATEL para uso em território brasileiro.

Nos últimos tempos, o principal obstáculo para a acessibilidade da automação em ambientes domésticos, comerciais e industriais foi o alto custo de soluções complexas, majoritariamente de origem importada, bem como a carência de mão de obra qualificada. Entretanto, com o advento da Internet da Coisas ou IoT (*Internet of Things*), tornou-se notório o interesse de pessoas comuns e pequenas empresas em utilizar essas tecnologias, antes de acesso restrito a grandes corporações.

Ademais, a disseminação dos conceitos de inteligência artificial, aprendizado de máquina e ciência de dados contribuem sobremaneira para a tendência de aumento da Internet das Coisas nos próximos anos, ampliando a possibilidade de utilização em vários de domínios de aplicação, a exemplo de: saúde, segurança, agronomia, meteorologia, educação e indústrias de modo geral.

Nesse cenário existe a preocupação da comunidade científica com a velocidade e confiabilidade no tráfego das informações, visto que os dispositivos IoT vêm sendo utilizados em diversas aplicações, desde simples automação residencial até o monitoramento de sinais vitais de pacientes clínicos.

Um sistema de monitoramento IoT, além das premissas de baixo custo e arquitetura aberta, deve ser capaz de garantir a comunicação segura e eficiente entre os clientes e servidores, baseados em algoritmos criptográficos robustos e protocolos confiáveis.

Hoje, estamos percebendo uma convergência de tecnologias antigas e tradicionais com soluções relativamente novas, das quais podemos destacar o Raspberry que pode desempenhar o papel de servidor, além do Esp8266 que são largamente utilizando como clientes

sem fio para envio de dados sensoriais. Esses dispositivos têm suporte para vários algoritmos de criptografia (AES, DES, ARC4), e protocolos de comunicação de padrão industrial, como (MQTT, Modbus, SNMP), o que reforça a necessidade de pesquisa para avaliação de segurança e confiabilidade em sistemas IoT.

2. APRESENTAÇÃO DA PROPOSTA

A proposta desse trabalho consiste na implementação e avaliação de protótipo de sistema de monitoramento web utilizando dispositivos sem fio Raspberry e Esp8266, criptografia simétrica AES (*Advanced Encryption Standard*) e protocolo de comunicação MQTT (*Message Queue Telemetry Transport*).

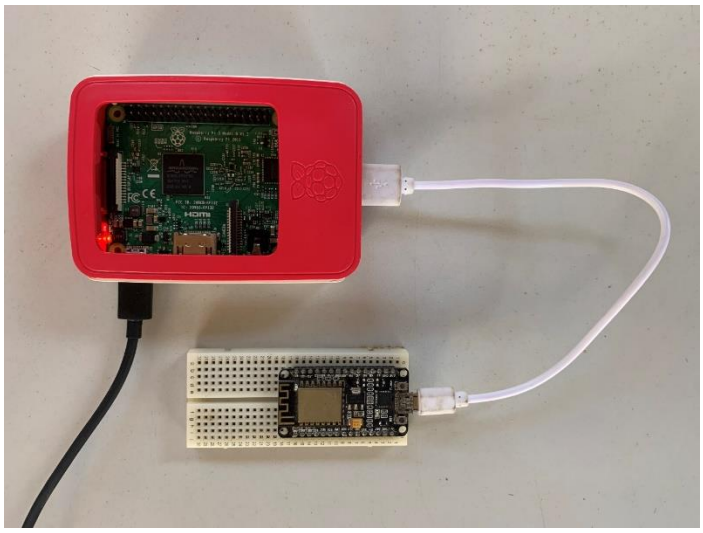

Figura 1 - Raspberry (superior) e Esp8266 NodeMCU (inferior)

O Esp8266 foi utilizado como cliente MQTT para coleta e envio de sinal analógico, que foram previamente criptografados com base no algoritmo AES.

A função de broker (servidor) MQTT foi desempenhada diretamente pelo Raspberry, dispensando a utilização de brokers externos, conforme arquitetura da Figura 2.

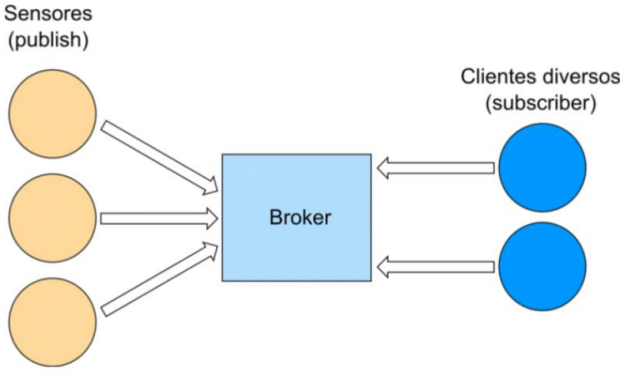

Figura 2 - Arquitetura do MQTT

A comunicação entre o Esp8266 (cliente) e o Raspberry (servidor) é realizada pela rede Wi-Fi. O código fonte do Esp8266 foi desenvolvido em C++ utilizando a IDE Arduino versão 1.8.13, enquanto o código do Raspberry, em Python, na IDE Thonny Python.

O monitoramento da aplicação pode ser realizado, em tempo real, a partir qualquer cliente assinante dos tópicos MQTT, a exemplo do MQTT Box (extensão Chrome), MQTool (aplicativo Android/iOS) e Elipse E3 (plataforma SCADA).

2.1. METODOLOGIA DE AVALIAÇÃO

No protótipo desenvolvido foram avaliados os tempos de processamento de encriptação e decriptação no Esp8266 utilizando o algoritmo AES, bem como os tempos de transmissão do sinal analógico via MQTT para o Raspberry. Também serão avaliados o consumo de memória de programa e de dados do Esp8266.

Os tempos foram computados diretamente nos códigos com precisão de microssegundos. Para garantir a acurácia dos dados, as rotinas de contagem foram repetidas 20 (vinte) vezes em intervalos de 1(um) segundo.

Para melhorar a amostragem do sinal analógico, utilizou-se uma variável randômica de 0 a 50 no Esp8266 com envio periódico, após cada ciclo de contagem de tempo.

As mensagens codificadas e decodificadas correspondentes ao valor atual do sinal analógico do Esp8266 são enviadas para os tópicos “MACC/IoT/Esp8266/Encrypted” e “MACC/IoT/Esp8266/Decrypted”, respectivamente. No Raspberry os dados decriptografados são coletados de forma automática e armazenados juntamente com a estampa de tempo em formato CSV. Alternativamente, o Raspberry poderia coletar os dados criptogrados e realizar internamente a decriptação, sendo inclusive a melhor de maior segurança, visto que o valor aberto (decriptografado) não estaria disponível no tópico assinado por brokers externos.

3. IMPLEMENTAÇÃO E ANÁLISES

No Esp8266 foram utilizadas as bibliotecas “ESP8266WiFi.h”, “AESLib.h” e “PubSubClient.h”, conforme código C++ a seguir.

```
1   #include "AESLib.h"
2   #include <PubSubClient.h>
```

```
3   #include <ESP8266WiFi.h>
4
5   AESLib aesLib;
6
7   const char* ssid = "ssid";
8   const char* password = "password";
9   const char* mqtt_server = "raspberrypi";
10  const int mqtt_port = 1883;
11
12  WiFiClient espClient;
13  PubSubClient client(espClient);
14
15  unsigned long lastMsg = 0;
16  #define MSG_BUFFER_SIZE  (50)
17  char msg[MSG_BUFFER_SIZE];
18
19  int time_count = 20;
20  unsigned long time_sum_enc = 0;
21  unsigned long time_sum_dec = 0;
22
23  String plaintext = "HELLO WORLD!";
24
25  char cleartext[256];
26  char ciphertext[512];
27  String encrypted, decrypted;
28
29  // AES Encryption Key
30  byte aes_key[] = { 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
0x30, 0x30, 0x30, 0x30, 0x30, 0x30 };
31
32  // General initialization vector (you must use your own IV's in production for
full security!!!)
33  byte aes_iv[N_BLOCK] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
34
35  // Generate IV (once)
36  void aes_init() {
37    Serial.println("gen_iv()");
38    aesLib.gen_iv(aes_iv);
39    Serial.println("encrypt()");
40    Serial.println(encrypt(strdup(plaintext.c_str()), plaintext.length(),
aes_iv));
41  }
42
43  String encrypt(char * msg, uint16_t msgLen, byte iv[]) {
44
```

```
45    int cipherlength = aesLib.get_cipher_length(msgLen);
46    char encrypted[cipherlength]; // AHA! needs to be large, 2x is not enough
47    aesLib.encrypt64(msg, msgLen, encrypted, aes_key, sizeof(aes_key), iv);
48    return String(encrypted);
49  }
50
51  String decrypt(char * msg, uint16_t msgLen, byte iv[]) {
52
53    char decrypted[msgLen];
54    aesLib.decrypt64(msg, msgLen, decrypted, aes_key, sizeof(aes_key), iv);
55    return String(decrypted);
56  }
57
58  void setup_wifi() {
59
60    delay(10);
61    // We start by connecting to a WiFi network
62    Serial.println();
63    Serial.print("Connecting to ");
64    Serial.println(ssid);
65
66    WiFi.mode(WIFI_STA);
67    WiFi.begin(ssid, password);
68
69      while (WiFi.status() != WL_CONNECTED) {
70        delay(500);
71        Serial.print(".");
72      }
73
74      //randomSeed(micros());
75
76      Serial.println("");
77      Serial.println("WiFi connected");
78      Serial.println("IP address: ");
79      Serial.println(WiFi.localIP());
80  }
81
82
83  void callback(char* topic, byte* payload, unsigned int length) {
84    Serial.print("Message arrived [");
85    Serial.print(topic);
86    Serial.print("] ");
87    for (int i = 0; i < length; i++) {
```

```
88        Serial.print((char)payload[i]);
89      }
90      Serial.println();
91
92     }
93
94    void reconnect() {
95      // Loop until we're reconnected
96      while (!client.connected()) {
97        Serial.print("Attempting MQTT connection...");
98        // Create a random client ID
99        String clientId = "ESP8266Client-";
100       clientId += String(random(0xffff), HEX);
101       // Attempt to connect
102       if (client.connect(clientId.c_str())) {
103         Serial.println("connected");
104         // Once connected, publish an announcement...
105         //client.publish("MACC/IoT/Esp8266/", "teste MQTT Esp8266");
106         // ... and resubscribe
107         //client.subscribe("MACC/IoT/Esp8266/");
108       } else {
109         Serial.print("failed, rc=");
110         Serial.print(client.state());
111         Serial.println(" try again in 5 seconds");
112         // Wait 5 seconds before retrying
113         delay(5000);
114       }
115     }
116   }
117
118
119   void setup() {
120     Serial.begin(9600);
121     while (!Serial); // wait for serial port
122     delay(2000);
123     aes_init();
124     aesLib.set_paddingmode(paddingMode::Array);
125
126     setup_wifi();
127
128     client.setServer(mqtt_server, mqtt_port);
129     client.setCallback(callback);
130
```

```
131     char b64in[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
132
133     char b64out[base64_enc_len(sizeof(aes_iv))];
134     base64_encode(b64out, b64in, 16);
135
136     char b64enc[base64_enc_len(10)];
137     base64_encode(b64enc, (char*) "0123456789", 10);
138
139     char b64dec[ base64_dec_len(b64enc, sizeof(b64enc))];
140     base64_decode(b64dec, b64enc, sizeof(b64enc));
141
142   }
143
144   /* non-blocking wait function */
145   void wait(unsigned long milliseconds) {
146     unsigned long timeout = millis() + milliseconds;
147     while (millis() < timeout) {
148       yield();
149     }
150   }
151
152
153   byte enc_iv[N_BLOCK] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }; //
iv_block gets written to, provide own fresh copy...
154   byte dec_iv[N_BLOCK] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
155
156   void MQTT_Publish (String encrypted, String decrypted) {
157
158     if (!client.connected()) {
159       reconnect();
160     }
161     client.loop();
162
163     snprintf (msg, MSG_BUFFER_SIZE, encrypted.c_str());
164     //Serial.print("Publish message: "); Serial.println(msg);
165     client.publish("MACC/IoT/Esp8266/Encrypted", msg);
166
167     snprintf (msg, MSG_BUFFER_SIZE, decrypted.c_str());
168     //Serial.print("Publish message: "); Serial.println(msg);
169     client.publish("MACC/IoT/Esp8266/Decrypted", msg);
170   }
171
172   void loop() {
```

```
173
174   unsigned long lastSend;
175
176     String readBuffer = String(random(50));
177
178     Serial.println("INPUT:" + readBuffer);
179
180     unsigned long loopcount = 0;
181     unsigned long time_sum_enc = 0;
182     unsigned long time_sum_dec = 0;
183
184     for (int i = 0; i < time_count; i++) {
185
186       loopcount++; Serial.println(loopcount); // entry counter
187
188       sprintf(cleartext, "%s", readBuffer.c_str()); // must not exceed 255
bytes; may contain a newline
189
190       // Encrypt
191       Serial.println("ENCRYPT");
192       unsigned long time_start_enc = micros();
193       uint16_t clen = String(cleartext).length();
194       encrypted = encrypt(cleartext, clen, enc_iv);
195       unsigned long time_aux_enc = micros() - time_start_enc;
196       time_sum_enc = time_sum_enc + time_aux_enc;
197
198       sprintf(ciphertext, "%s", encrypted.c_str());
199       //Serial.print("Ciphertext: "); Serial.println(encrypted);
200       Serial.print("Encryption time: "); Serial.print(time_aux_enc);
Serial.println("us");
201       delay(500);
202
203       // Decrypt
204       Serial.println("DECRYPT");
205       unsigned long time_start_dec = micros();
206       uint16_t dlen = encrypted.length();
207       decrypted = decrypt(ciphertext, dlen, dec_iv);
208       unsigned long time_aux_dec = micros() - time_start_dec;
209       time_sum_dec = time_sum_dec + time_aux_dec;
210
211       //Serial.print("Cleartext: "); Serial.println(decrypted);
212       Serial.print("Decryption time: "); Serial.print(time_aux_dec);
Serial.println("us");
213       delay(500);
214
215       // Validation
```

```
216       if (decrypted.equals(cleartext)) {
217         Serial.println("SUCCES");
218       }
219       else
220       {
221         Serial.println("FAILURE");
222
223       }
224       for (int i = 0; i < 16; i++) {
225         enc_iv[i] = 0;
226         dec_iv[i] = 0;
227       }
228     }
229     unsigned long time_avg_enc = time_sum_enc / time_count;
230     unsigned long time_avg_dec = time_sum_dec / time_count;
231     Serial.println("--------------------------");
232     Serial.print("Encryption time average: "); Serial.print(time_avg_enc);
Serial.println("us");
233     Serial.print("Decryption time average: "); Serial.print(time_avg_dec);
Serial.println("us");
234     Serial.println("--------------------------");
235     Serial.print("Encrypted message: "); Serial.println(encrypted);
236     Serial.print("Decrypted message: "); Serial.println(decrypted);
237     Serial.println("--------------------------");
238
239     //MQTT Publish
240
241     Serial.println("INPUT:" + readBuffer);
242
243     loopcount = 0;
244     unsigned long time_sum_pub = 0;
245     unsigned long time_sum_sub = 0;
246
247     for (int i = 0; i < time_count; i++) {
248
249       loopcount++; Serial.println(loopcount); // entry counter
250
251       unsigned long time_start_pub = micros();
252       MQTT_Publish (encrypted, decrypted);
253       unsigned long time_aux_pub = micros() - time_start_pub;
254       time_sum_pub = time_sum_pub + time_aux_pub;
255       Serial.print("MQTT publish time: "); Serial.print(time_aux_pub);
Serial.println("us");
256       delay(500);
257
```

```
258     }
259
260     unsigned long time_avg_pub = time_sum_pub / time_count;
261
262     Serial.println("--------------------------");
263     Serial.print("MQTT publish time average: "); Serial.print(time_avg_pub);
Serial.println("us");
264     Serial.println("--------------------------");
265     Serial.print("Published message: "); Serial.println(encrypted);
266     Serial.print("Published message: "); Serial.println(decrypted);
267
268    }
```

Nas linhas 45 a 55 do código acima são processadas as funções de encriptação e decriptação do sinal analógico representado pelo valor randômico (0 a 50) gerado na linha 176, as quais são repetidas 20 vezes e, ao final, as médias armazenadas nas variáveis “time_avg_enc” (linha 229) e “time_avg_dec” (linha 230). Os referidos tempos são publicados nos tópicos "MACC/IoT/Esp8266/Encrypted" e "MACC/IoT/Esp8266/Decrypted" (linhas 156 a 170).

No processo de criptografia foi utilizado o algoritmo simétrico AES com camada adicional Base64 e chave de 16 bytes, especificada na linha 30. No log abaixo estão discriminados os tempos de 20 rotinas de encriptação e decriptação para o valor de entrada “26”, a partir dos quais foram extraídos para análise os respectivos valores médios.

```
12:48:09.472 -> INPUT:26
12:48:09.472 -> 1
12:48:09.472 -> ENCRYPT
12:48:09.519 -> Encryption time: 780us
12:48:09.896 -> DECRYPT
12:48:09.896 -> Decryption time: 916us
12:48:10.410 -> SUCCES
12:48:10.410 -> 2
12:48:10.410 -> ENCRYPT
12:48:10.410 -> Encryption time: 860us
12:48:10.876 -> DECRYPT
12:48:10.923 -> Decryption time: 916us
12:48:11.393 -> SUCCES
12:48:11.393 -> 3
12:48:11.393 -> ENCRYPT
12:48:11.393 -> Encryption time: 863us
12:48:11.904 -> DECRYPT
12:48:11.904 -> Decryption time: 929us
12:48:12.419 -> SUCCES
```

```
12:48:12.419 -> 4
12:48:12.419 -> ENCRYPT
12:48:12.419 -> Encryption time: 863us
12:48:12.889 -> DECRYPT
12:48:12.889 -> Decryption time: 917us
12:48:13.401 -> SUCCES
12:48:13.401 -> 5
12:48:13.401 -> ENCRYPT
12:48:13.401 -> Encryption time: 871us
12:48:13.918 -> DECRYPT
12:48:13.918 -> Decryption time: 917us
12:48:14.387 -> SUCCES
12:48:14.433 -> 6
12:48:14.433 -> ENCRYPT
12:48:14.433 -> Encryption time: 871us
12:48:14.900 -> DECRYPT
12:48:14.900 -> Decryption time: 913us
12:48:15.416 -> SUCCES
12:48:15.416 -> 7
12:48:15.416 -> ENCRYPT
12:48:15.416 -> Encryption time: 858us
12:48:15.890 -> DECRYPT
12:48:15.933 -> Decryption time: 927us
12:48:16.402 -> SUCCES
12:48:16.402 -> 8
12:48:16.402 -> ENCRYPT
12:48:16.448 -> Encryption time: 863us
12:48:16.917 -> DECRYPT
12:48:16.917 -> Decryption time: 917us
12:48:17.429 -> SUCCES
12:48:17.429 -> 9
12:48:17.429 -> ENCRYPT
12:48:17.429 -> Encryption time: 868us
12:48:17.897 -> DECRYPT
12:48:17.946 -> Decryption time: 916us
12:48:18.413 -> SUCCES
12:48:18.413 -> 10
12:48:18.413 -> ENCRYPT
12:48:18.459 -> Encryption time: 863us
12:48:18.927 -> DECRYPT
12:48:18.927 -> Decryption time: 912us
12:48:19.398 -> SUCCES
12:48:19.445 -> 11
12:48:19.445 -> ENCRYPT
12:48:19.445 -> Encryption time: 859us
12:48:19.917 -> DECRYPT
12:48:19.917 -> Decryption time: 912us
12:48:20.433 -> SUCCES
12:48:20.433 -> 12
12:48:20.433 -> ENCRYPT
```

```
12:48:20.433 -> Encryption time: 859us
12:48:20.906 -> DECRYPT
12:48:20.951 -> Decryption time: 912us
12:48:21.421 -> SUCCES
12:48:21.421 -> 13
12:48:21.421 -> ENCRYPT
12:48:21.421 -> Encryption time: 873us
12:48:21.937 -> DECRYPT
12:48:21.937 -> Decryption time: 917us
12:48:22.406 -> SUCCES
12:48:22.453 -> 14
12:48:22.453 -> ENCRYPT
12:48:22.453 -> Encryption time: 854us
12:48:22.920 -> DECRYPT
12:48:22.920 -> Decryption time: 912us
12:48:23.436 -> SUCCES
12:48:23.436 -> 15
12:48:23.436 -> ENCRYPT
12:48:23.436 -> Encryption time: 863us
12:48:23.951 -> DECRYPT
12:48:23.951 -> Decryption time: 912us
12:48:24.420 -> SUCCES
12:48:24.420 -> 16
12:48:24.420 -> ENCRYPT
12:48:24.468 -> Encryption time: 859us
12:48:24.938 -> DECRYPT
12:48:24.938 -> Decryption time: 913us
12:48:25.454 -> SUCCES
12:48:25.454 -> 17
12:48:25.454 -> ENCRYPT
12:48:25.454 -> Encryption time: 858us
12:48:25.922 -> DECRYPT
12:48:25.922 -> Decryption time: 926us
12:48:26.433 -> SUCCES
12:48:26.433 -> 18
12:48:26.433 -> ENCRYPT
12:48:26.480 -> Encryption time: 876us
12:48:26.949 -> DECRYPT
12:48:26.949 -> Decryption time: 912us
12:48:27.461 -> SUCCES
12:48:27.461 -> 19
12:48:27.461 -> ENCRYPT
12:48:27.461 -> Encryption time: 854us
12:48:27.931 -> DECRYPT
12:48:27.931 -> Decryption time: 913us
12:48:28.444 -> SUCCES
12:48:28.444 -> 20
12:48:28.444 -> ENCRYPT
12:48:28.444 -> Encryption time: 867us
12:48:28.958 -> DECRYPT
```

```
12:48:28.958 -> Decryption time: 914us
12:48:29.430 -> SUCCES
12:48:29.477 -> -------------------------
12:48:29.477 -> Encryption time average: 859us
12:48:29.523 -> Decryption time average: 916us
12:48:29.570 -> -------------------------
12:48:29.570 -> Encrypted message: kVtDhLJ9JOeP3OClrHyQJQ==
12:48:29.618 -> Decrypted message: 26
12:48:29.665 -> -------------------------
```

Para a transmissão da mensagem do Esp866 para o Raspberry foi utilizado o protocolo MQTT, com publicação nos tópicos “MACC/IoT/Esp8299/Encryted” e “MACC/IoT/Es8266/Decrypted”. No log abaixo estão discriminadas os tempos de 20 publicações para o valor de entrada “26”, a partir dos quais foi extraído para análise o valor médio, armazenado na variável “time_avg_pub” (linha 260).

```
12:48:29.665 -> INPUT:26
12:48:29.712 -> 1
12:48:29.712 -> MQTT publish time: 3637us
12:48:30.089 -> 2
12:48:30.089 -> MQTT publish time: 1995us
12:48:30.603 -> 3
12:48:30.603 -> MQTT publish time: 1883us
12:48:31.073 -> 4
12:48:31.073 -> MQTT publish time: 1894us
12:48:31.584 -> 5
12:48:31.584 -> MQTT publish time: 953us
12:48:32.100 -> 6
12:48:32.100 -> MQTT publish time: 949us
12:48:32.613 -> 7
12:48:32.613 -> MQTT publish time: 1898us
12:48:33.080 -> 8
12:48:33.080 -> MQTT publish time: 2026us
12:48:33.592 -> 9
12:48:33.592 -> MQTT publish time: 1909us
12:48:34.108 -> 10
12:48:34.108 -> MQTT publish time: 1860us
12:48:34.620 -> 11
12:48:34.620 -> MQTT publish time: 1879us
12:48:35.086 -> 12
12:48:35.134 -> MQTT publish time: 1898us
12:48:35.600 -> 13
12:48:35.600 -> MQTT publish time: 1873us
12:48:36.112 -> 14
12:48:36.112 -> MQTT publish time: 1873us
12:48:36.626 -> 15
12:48:36.626 -> MQTT publish time: 1871us
```

```
12:48:37.115 -> 16
12:48:37.115 -> MQTT publish time: 1867us
12:48:37.605 -> 17
12:48:37.605 -> MQTT publish time: 1861us
12:48:38.118 -> 18
12:48:38.118 -> MQTT publish time: 1890us
12:48:38.638 -> 19
12:48:38.638 -> MQTT publish time: 2175us
12:48:39.105 -> 20
12:48:39.151 -> MQTT publish time: 1932us
12:48:39.609 -> --------------------------
12:48:39.620 -> MQTT publish time average: 1906us
12:48:39.667 -> --------------------------
12:48:39.714 -> Published message: kVtDhLJ9JOeP3OClrHyQJQ==
12:48:39.760 -> Published message: 26
```

Com relação à performance do Esp8266 para uso com criptografia AES e protocolo MQTT, temos os seguintes resultados.

```
Executable segment sizes:
IROM   : 360844          - code in flash         (default or ICACHE_FLASH_ATTR)
IRAM   : 27288   / 32768 - code in IRAM          (ICACHE_RAM_ATTR, ISRs...)
DATA   : 1508  )         - initialized variables (global, static) in RAM/HEAP
RODATA : 3472  ) / 81920 - constants             (global, static) in RAM/HEAP
BSS    : 31648 )         - zeroed variables      (global, static) in RAM/HEAP
O sketch usa 393112 bytes (41%) de espaço de armazenamento para programas. O máximo são 958448 bytes.
Variáveis globais usam 36628 bytes (44%) de memória dinâmica, deixando 45292 bytes para variáveis locais.
O máximo são 81920 bytes.
```

No Raspberry Pi 3 foram utilizadas as bibliotecas “paho.mqtt.client”, “pandas” e “datetime”, conforme código Python a seguir.

```python
import paho.mqtt.client as mqtt
import pandas as pd
from datetime import datetime


mqtt_topic = "MACC/IoT/Esp8266/Decrypted"
mqtt_server = "raspberrypi"
mqtt_port = 1883

client = mqtt.Client()


data = []

```

```
13          def on_connect(client, userdata, flags, rc):
14
15              print ("Connected!")
16
17              client.subscribe(mqtt_topic)
18
19          def on_message(client, userdata, msg):
20
21              now = datetime.now().strftime('%d/%m/%Y %H:%M:%S')
22              temp = msg.payload.decode('utf-8')
23
24              data.append([now, temp])
25              df = pd.DataFrame(data, columns = ['Time','Decrypted'])
26              df.to_csv('dataset_IoT.csv', mode = 'a', header = False)
27              print(df)
28          client.on_connect = on_connect
29          client.on_message = on_message
30          client.connect(mqtt_server, mqtt_port)
31          client.loop_forever()
32          client.disconnect()
```

No Raspberry foram realizadas as etapas de coleta automática de dados do tópico assinante “MACC/IoT/Esp8266/Decrypted” (linha 17), transformação das séries temporais em dataframe Pandas (linhas 21 a 25) e armazenamento local em arquivo estruturado CSV (linha 26), o qual pode ser para exploração e visualização em ferramentas externas (Jupyter Notebook e Visual Studio Code, por exemplo). Na Figura 3 encontra-se o dataset gerado com 2 colunas e 3.000 linhas.

3001 lines (3001 sloc) | 66.7 KB

Search this file...

	TimeStamp	Decrypted
2	28/11/2020 19:22:57	43
3	28/11/2020 19:22:57	43
4	28/11/2020 19:23:44	38
5	28/11/2020 19:22:57	43
6	28/11/2020 19:23:44	38
7	28/11/2020 19:24:04	11
8	28/11/2020 19:24:45	36
9	28/11/2020 19:24:45	36
10	28/11/2020 19:25:05	21

Figura 3 - Dataset CSV

A visualização dos dados pode ser realizada no próprio Raspberry utilizando as bibliotecas gráficas do Python. Também é possível o monitoramento remoto a partir de qualquer cliente MQTT na web, sendo recomendada, nesses casos, a utilização de autenticação de acesso (usuário e senha). Na Figura 4 os dados estão sendo monitorados em tempo real pelo aplicativo MQTT Box, disponível como extensão do navegador Chrome.

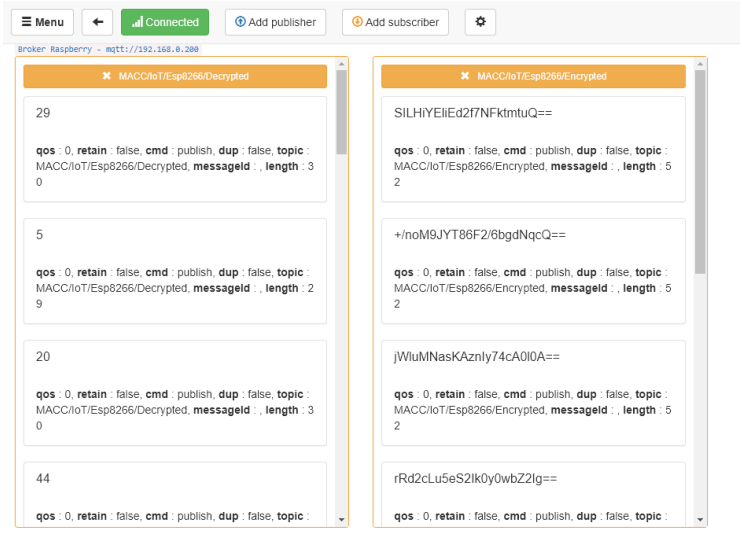

Figura 4 - Aplicativo MQTT Box - Chrome

Na figura abaixo, os dados estão sendo visualizado via smartphone pelo aplicativo MQTTool, disponível para Android/iOS.

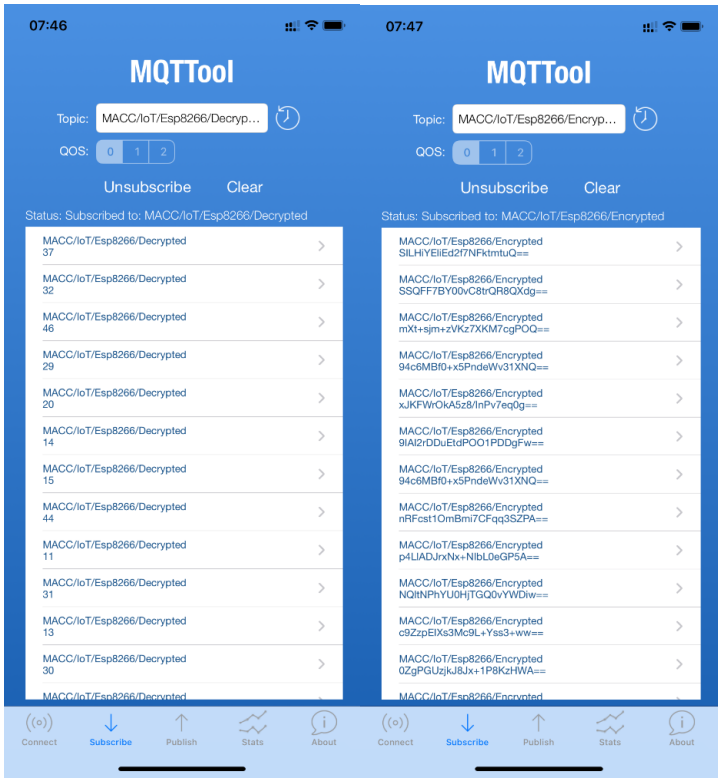

Figura 5 Aplicativo MQTT Box - iOS

Também é possível a utilização de sistemas supervisórios (SCADA) como clientes MQTT. Para tanto, foi desenvolvido projeto no Elipse E3, utilizando driver de comunicação "MQTT.dll" para monitoramento dos tópicos MQTT do broker Raspberry, conforme tela do Elipse E3 Viewer da Figura 6.

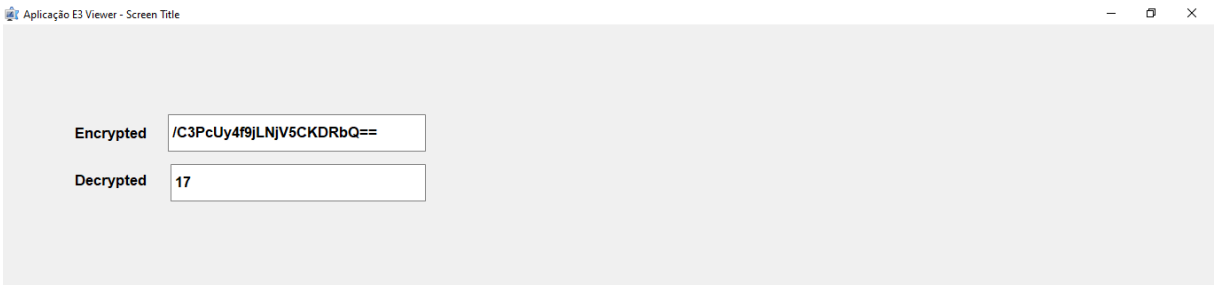

Figura 6 - Elipse E3 Viewer

3.1. RESULTADOS

Na tabela abaixo encontram-se os valores obtidos durante a implementação do protótipo do sistema IoT utilizando criptografia AES, protocolo de comunicação MQTT e conectividade Wi-Fi.

O tempo médio de encriptação e decriptação foram de 0,859ms e 0,916ms, enquanto o tempo de publicação no tópico MQTT foi de 1,906ms, perfazendo o tempo total de 3,681ms. O programa e as variáveis utilizadas no Esp8266 consumiram 41% de memória EEPROM e 44% de memória SRAM. Não foi realizada a medição dos tempos de processamento e consumo do Raspberry, pois a demanda da aplicação é irrelevante.

ESP8266					
Tempo de Encriptação AES (ms)	Tempo de Decriptação AES (ms)	Tempo de Publicação MQTT (ms)	Tempo Total (ms)	Uso de Memória EEPROM	Uso de Memória SRAM
0,859	0,916	1,906	3,681	41%	44%

4. CONCLUSÕES E TRABALHOS FUTUROS

4.1. CONCLUSÕES

De acordo com os resultados, o tempo total de encriptação, decriptação e publicação do ESP8266 não representa impacto significativo no tráfego de dados. As bibliotecas utilizadas

mostraram-se eficientes, sendo que o consumo de memória EEPROM e SRAM não afetam a disponibilidade do recurso. Dessa forma, o emprego de criptografia AES e comunicação MQTT tornam-se fortemente recomendados em dispositivos IoT ESP8266 e similares.

O Raspberry funcionou adequadamente como servidor MQTT, dispensando a utilização de brokers externos, seja públicos ou pagos, e ferramentas externas de visualização de dados. Também mostrou notória eficiência na coleta, análise e geração do dataset utilizando as bibliotecas Python.

Diante dos resultados obtidos, podemos concluir que a performance dos dispositivos IoT ESP8266 e Raspberry foram consideradas satisfatória durante o uso com criptografia AES, protocolo de comunicação MQTT e conectividade sem fio.

Ademais, a solução desenvolvida é considerada de baixo custo, pois podem substituir controladores lógicos programáveis (CLP), servidores, drivers de comunicação e sistemas proprietários de supervisão por dispositivos de baixo custo e softwares de uso livre.

Diante do exposto, reforça-se a viabilidade técnica e científica de utilização de sistema de monitoramento IoT de baixo custo em aplicações que requerem segurança, disponibilidade e confiabilidade.

4.2. TRABALHOS FUTUROS

Com base no conhecimento adquirido na presente pesquisa, vimos propor os trabalhos futuros relacionados a seguir:

- Adequar o sistema para uso em indústria (IIoT);
- Integrar outros algoritmos de criptografia simétrica e assimétrica;
- Integrar outros dispositivos IoT;
- Utilizar o dataset gerado para análise preditiva utilizando *Machine Learning*;
- Utilizar sensores para medição do consumo de energia dos dispositivos IoT;
- Utilizar o banco de dados SQL disponível no Raspberry para armazenamento dos dados;

Ademais, o protótipo de sistema IoT de baixo custo pode ser utilizado em vários domínios de aplicação, quais sejam:

- Monitoramento de sistemas de segurança de agências bancárias;
- Monitoramento de produtividade OEE de plantas industriais;
- Monitoramento de sistemas hospitalares;
- Monitoramento de sistemas meteorológicos.

APÊNDICE

Repositório do projeto: https://github.com/danielgleison/IoT

- Código C++ - Esp8266 NodeMCU;
- Código Python - Raspberry Pi 3
- Projeto Elipse E3 - MQTT Client;
- Dataset CSV;
- Logs dos resultados;
- Fotos.