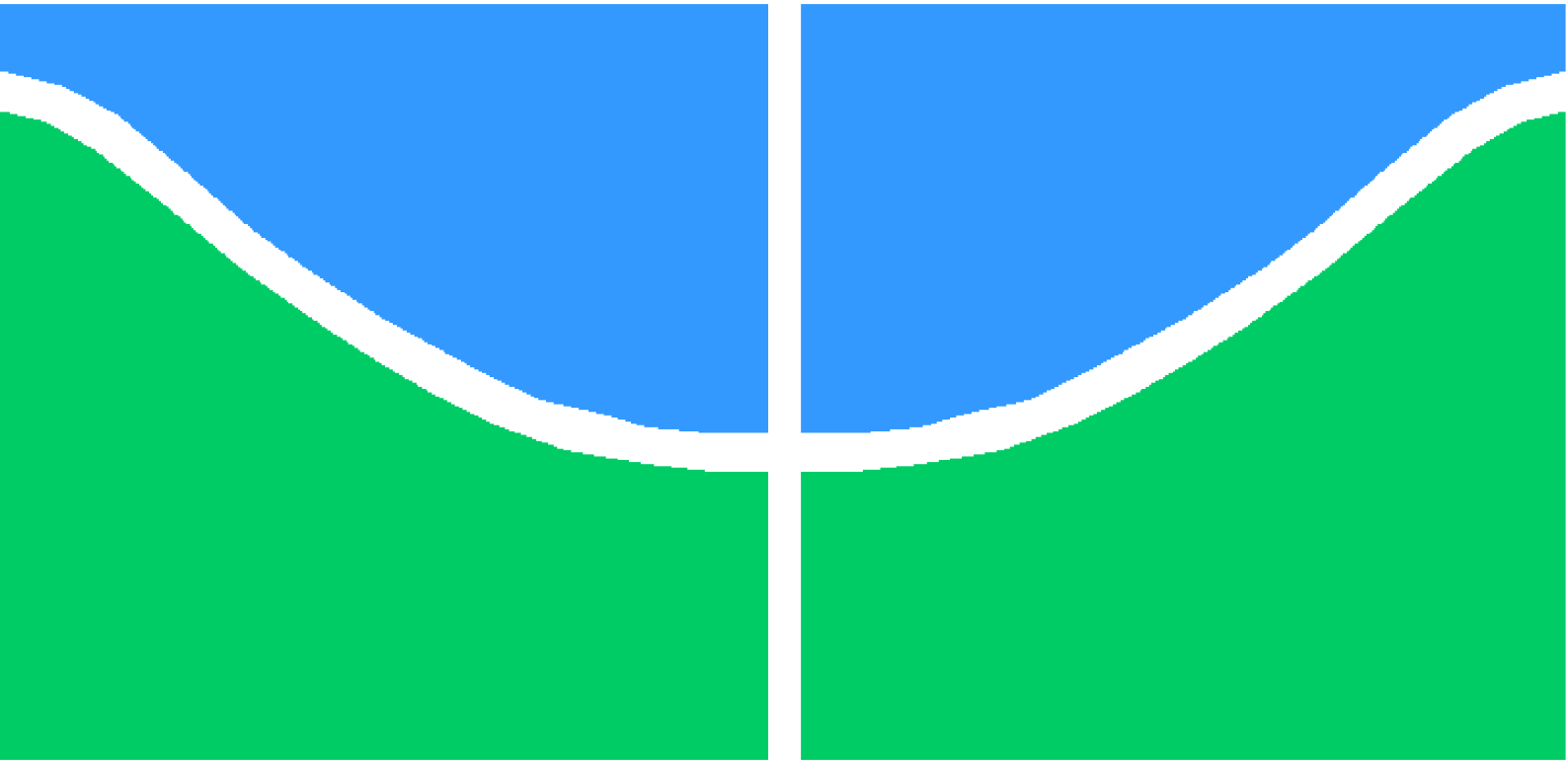

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia Eletrônica

Protocolo para segurança da informação em Gateways IoT

Autor: Aubani Júnio Teixeira Cândido
Orientador: Prof. Dr. Tiago Alves da Fonseca

Brasília, DF
2019

Aubani Júnio Teixeira Cândido

Protocolo para segurança da informação em Gateways IoT

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Tiago Alves da Fonseca

Brasília, DF

2019

Aubani Júnio Teixeira Cândido
Protocolo para segurança da informação em Gateways IoT/ Aubani Júnio Teixeira Cândido. – Brasília, DF, 2019-
46 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Tiago Alves da Fonseca

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2019.

1. IoT. 2. Segurança IoT. I. Prof. Dr. Tiago Alves da Fonseca. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Protocolo para segurança da informação em Gateways IoT

CDU 02:141:005.6

Aubani Júnio Teixeira Cândido

Protocolo para segurança da informação em Gateways IoT

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Trabalho aprovado. Brasília, DF, Julho de 2019:

Prof. Dr. Tiago Alves da Fonseca
Orientador

Prof. Dr. Luiz Augusto Fontes Laranjeira
Convidado 1

Dr. José Antônio Carrijo Barbosa
Convidado 2

Brasília, DF
2019

Agradecimentos

Agradeço aos meus amigos e, principalmente, à minha família pelo apoio a conquistar meus sonhos. Desejo felicidade e sucesso nas conquistas da minha irmã, Kálida. :)

Resumo

A Internet das Coisas traz diversas possibilidades promissoras à humanidade, permitindo que os mais variados dispositivos se conectem à Internet para tráfego de dados. Esses dispositivos são sensores de diversas grandezas físicas, atuadores para várias funções, sistemas embarcados designados para controles de sistemas e etc. Entretanto esses dispositivos por serem, em geral, simples e de baixo consumo, não usam algoritmos de segurança já consolidados por questões que é mais conveniente para o mercado encurtar o tempo de evolução em detrimento de segurança. O presente trabalho busca algoritmos viáveis que sejam mais simples e proporcionem um bom nível de segurança. Portanto, é criado e implementado um protocolo que atenda serviços básicos de segurança da informação objetivando afetar minimamente as funções que os dispositivos IoT foram designados.

Palavras-chaves: iot. internet das coisas. segurança da informação. protocolos simples.

Abstract

The Internet of Things brings a number of promising possibilities to mankind, allowing various devices to connect to the Internet for data traffic. These devices are sensors, actuators, embedded systems designed to control other systems and so on. However, because these devices are generally simple and demand low power, they do not use consolidated security algorithms due to hardware limitations and narrow schedule to deliver the product to the market. The present work looks for feasible algorithms that are lighter, simpler and provide a good level of security. Therefore, a protocol was created to provide basic services of information security while minimally affecting the functions that the IoT devices have been assigned.

Key-words: iot. internet of things. information security. simple protocols

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AES	Advanced Encryption Standard
DSA	Digital Signature Algorithm
ECC	Elliptic-curve cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
RSA	Rivest-Shamir-Adleman
RTT	Round Trip Time

Sumário

1 Introdução

1.1 Contextualização e Justificativa

O termo Internet das Coisas ou *Internet of Things* (IoT), usado pela primeira vez em 1999 por Kevin Ashton, foi utilizado para descrever dispositivos físicos que se conectavam à Internet (ROSE; ELDRIDGE; CHAPIN, 2015, p. 12). Hoje é um assunto em grande ascensão que se refere a sensores, componentes, carros e outros objetos que se conectam à Internet. Desta forma esses objetos são capazes de gerar quantidades significativas de dados que podem ser utilizadas para análises e processamento. Sendo assim a proposta do IoT é transformar a forma como a sociedade trabalha e vive. (ROSE; ELDRIDGE; CHAPIN, 2015, p. 4).

Estima-se que mais de 100 bilhões de dispositivos IoT estejam conectados à Internet e um grande impacto na economia mundial com $ 11 trilhões até 2025, afirma Rose, Eldridge e Chapin (2015, p. 4) no *White paper*.

Em termos de dimensão digital, segundo a Cisco® Visual Networking Index™ (2017, p. 2), em 2016 o tráfego *Internet Protocol* (IP) global anual foi de 96 EB, sendo 1 EB equivalente a 1 bilhão de GB. Em 2021 deve chegar próximo de 278 EB, representando um aumento de, aproximadamente, 189,58% em cinco anos. Ainda nesse intervalo, em 2016 os computadores pessoais, *Personal Computer* (PC), representaram cerca de 46% de todo o tráfego IP, porém em 2021 espera-se um valor próximo de 25% apenas, ou seja, 75% dos dispositivos conectados à Internet não serão computadores pessoais.

A dimensão digital citada mostra o quanto diversificados serão os tipos de dispositivos conectados à Internet trafegando os mais diferentes tipos de dados e/ou informações. Isso mostra a interoperabilidade desses dispositivos na rede e, entretanto, a diferença de sistemas traz um desafio no estabelecimento de um patamar mínimo de segurança operacional. Sendo assim, qualquer dispositivo conectado à rede pode ser um potencial elo fraco na segurança e comprometer a rede.

Com ascensão do IoT, vem também o crescimento de questões relacionadas à segurança, dada a quantidade de dados que são trocados na rede. Para a utilização de novas tecnologias propiciadas pelo IoT é necessário, no mínimo, a garantia da segurança da informação para os usuários. É preciso ter confiança e segurança na utilização dos dispositivos IoT. Caso não implementado os serviços de segurança, uma série de possibilidades com advento da tecnologia poderia ser inviabilizada por relutância dos usuários, decorrente da divulgação das consequências de episódios de falhas de segurança. (ROSE; ELDRIDGE; CHAPIN, 2015, p. 32).

Cria-se um dilema bastante discutido atualmente que se manifesta como um desafio para grandes empresas que atuam na área de tecnologia de IoT como Intel®, Cisco ®, IBM®, Microsoft: a busca por soluções IoT simples, com baixo consumo de energia e baixo custo, mas garantindo segurança para os usuários. Para tanto, são necessários algoritmos específicos que acompanhem os requisitos de hardware para dispositivos IoT que usem o mínimo de processamento e energia, mas garantindo a segurança das informações. (GERBER, 2017, Challenge #1).

1.2 Objetivos

1.2.1 Objetivo geral

Com base no exposto, são explorados neste trabalho propostas para garantir a segurança de dispositivos IoT, mais especificamente a criação de um protocolo que garanta o sigilo, a autenticidade e a integridade das informações considerando o baixo poder computacional dos dispositivos envolvidos através de um *gateway* específico para redes IoT.

Com base nesse propósito, o presente trabalho avalia o desempenho do protocolo desenvolvido frente ao SSL, ambos sendo usados em um *gateway* IoT. Para isso foram definidos objetivos específicos listados abaixo.

1.2.2 Objetivos específicos

- Definir tipos de hardwares disponíveis no mercado que possam atuar como *gateway* e avaliar sua capacidade frente à todo o processamento necessário dos dispositivos IoT e o quanto a rede estará sobrecarregada em processamento. Ou seja, o quanto de processamento estará concentrado no *gateway* e o quanto estará distribuído para os dispositivos IoT.
- Estabelecer um tipo protocolo e definir um algoritmo que garanta a autenticidade, integridade e o sigilo das mensagens trocadas entre o dispositivo IoT e o *gateway*.
- Definir e implementar algoritmos que garantam a autenticidade entre o dispositivo IoT e o *gateway*, com base no protocolo aqui proposto.
- Criar um protocolo criptográfico que permita realizar o acordo de chaves para criptografia entre o dispositivo IoT e o *gateway*.
- No protocolo de acordo de chaves, uma vez definidas as chaves simétricas, utilizar algoritmo criptográfico simétrico para garantir o sigilo entre o dispositivo IoT e o *gateway*.

2 Fundamentação Teórica

2.1 Internet das Coisas

A Internet das Coisas parte da premissa da conexão entre diversos tipos de dispositivos à Internet. Esses dispositivos para IoT são chamados de *Things*. Cada *Thing* tem um propósito diferenciado, isto é, alguns são sensores de temperatura, outros de umidade, entre outras grandezas físicas. Mas *Things* não se limitam somente a sensores, abrangem objetos mais complexos como TVs, carros, casa e até mesmo objetos de iluminação pública.

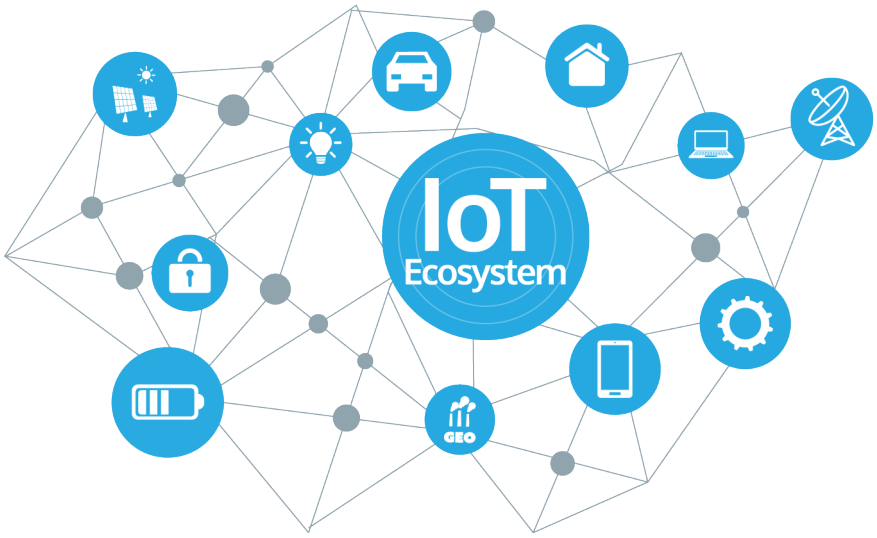

Figura 1 – Ecossistema IoT (Integral PLM Experts, 2016).

Com advento do IoT, surgiram conceitos de cidades inteligentes que possuem diversos mecanismos para seus habitantes com automação e obtenção de informações graças à Internet das Coisas.

Uma forma de abstração simples de uma rede IoT é descrita pelos *Things*, *Gateway* fazendo a ligação com o *Cloud* através da Internet, como mostra a Fig. 2.

Entretanto existe uma grande complexidade nas redes IoT que variam a forma de comunicação desde a camada física até a camada de aplicação, usando o modelo OSI (Cisco Systems, Inc., 2006). Há uma variedade de conexões como RFID, WiFi, LoRa™, Ethernet, 3G, 4G e outros. Em camadas superiores, há diferentes protocolos de comunicação como MQTT, HTTPS, REST e etc.

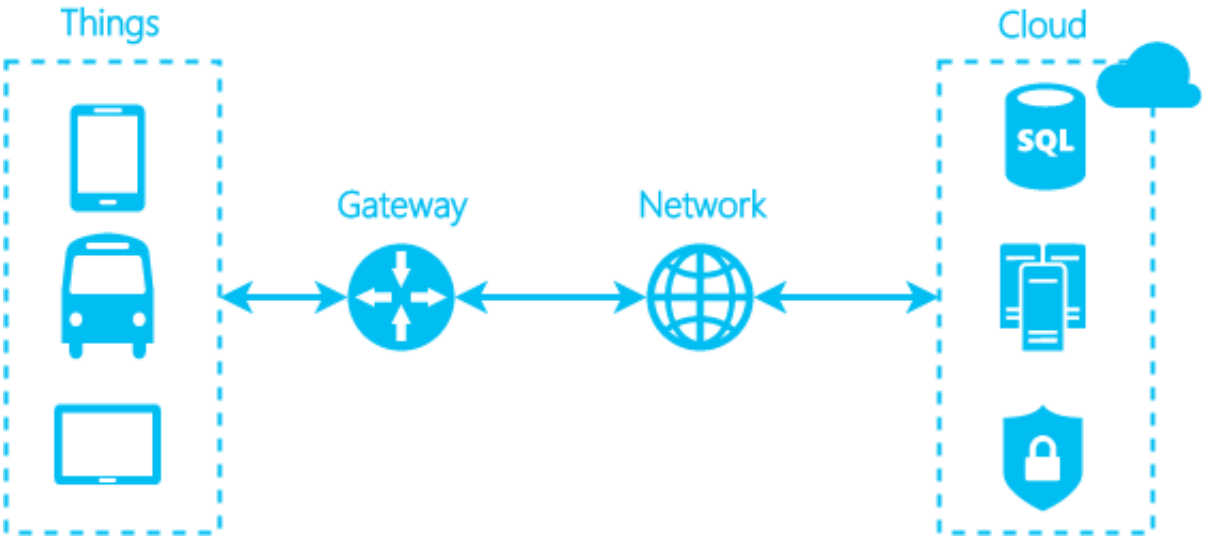

Figura 2 – Diagrama de conexão IoT.

A Figura 3 mostra a diversidade e complexidade de redes IoT utilizadas pela Intel®, com vários sensores, atuadores, objetos, denominados *Things*. Com a presença de algoritmos de segurança, fluxo de dados e controle desse fluxo.

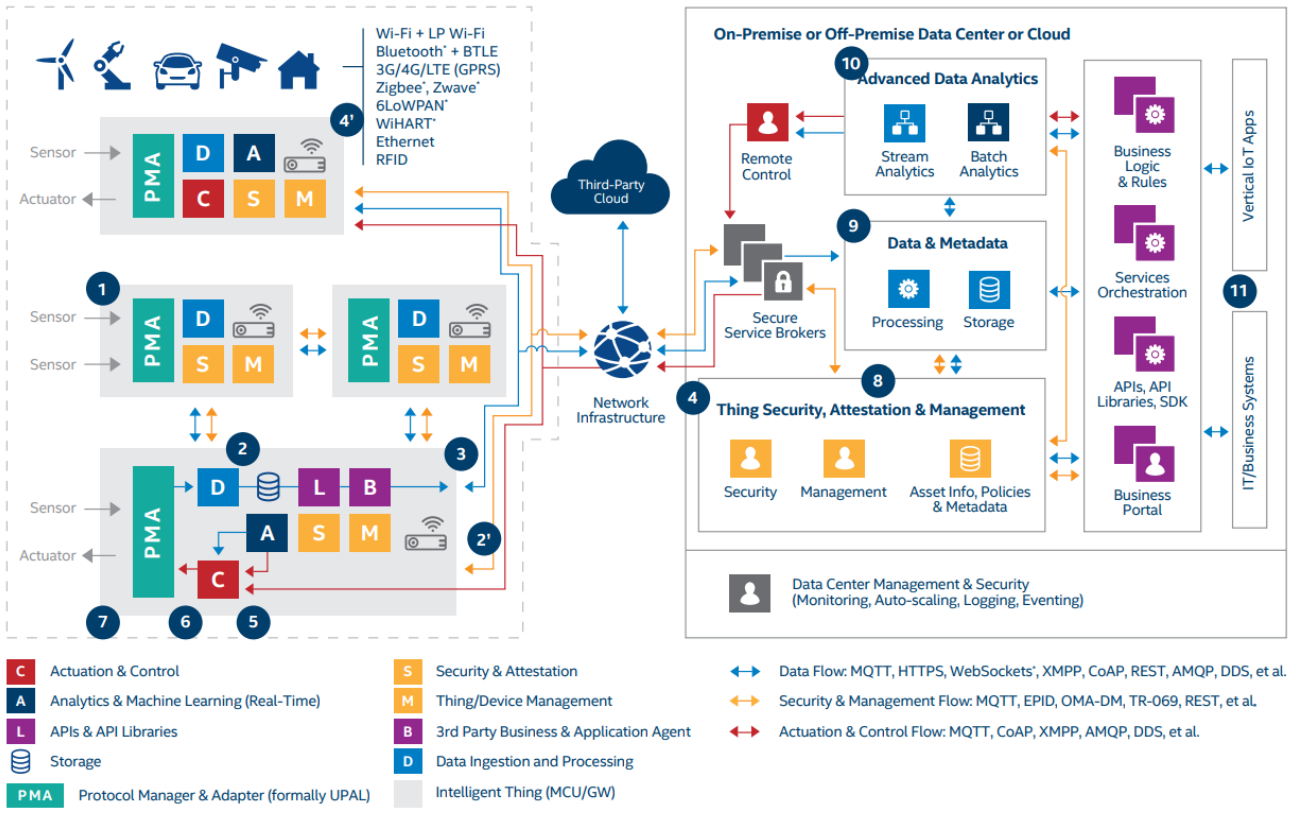

Figura 3 – Diagrama de conexão IoT detalhado (INTEL®, 2015).

2.2 Segurança da Informação

Como foi descrito anteriormente, há uma quantidade enorme de informação que trafega pela rede com a Internet das Coisas. No entanto, há algo essencial para o uso efetivo de IoT que é a segurança.

A segurança é um serviço primordial para IoT. Como diversos dispositivos se conectam a uma rede e caso ocorram falhas na segurança essas poderão comprometer toda a rede. Há questões quanto aos serviços básicos da segurança da informação que são a confidencialidade, a integridade, a autenticidade e o não-repúdio (KIM DAVID; SOLOMON, 2018, p. 293).

A confidencialidade é o serviço que tem como objetivo de garantir o sigilo da mensagem entre os usuários, ou seja, qualquer pessoa que esteja monitorando o canal e intercepte uma mensagem não será capaz de interpretá-la, como mostra a Fig. 4. Em IoT são utilizados algoritmos de criptografia de chave simétrica (AES, GCM, Salsa20) e criptografia assimétrica (RSA, ECC) para prover sigilo, integridade, autenticidade e não-repúdio. Desta forma, somente aqueles que possuem as chaves criptográficas simétricas utilizadas podem decifrar a mensagem e interpretar seu conteúdo corretamente.

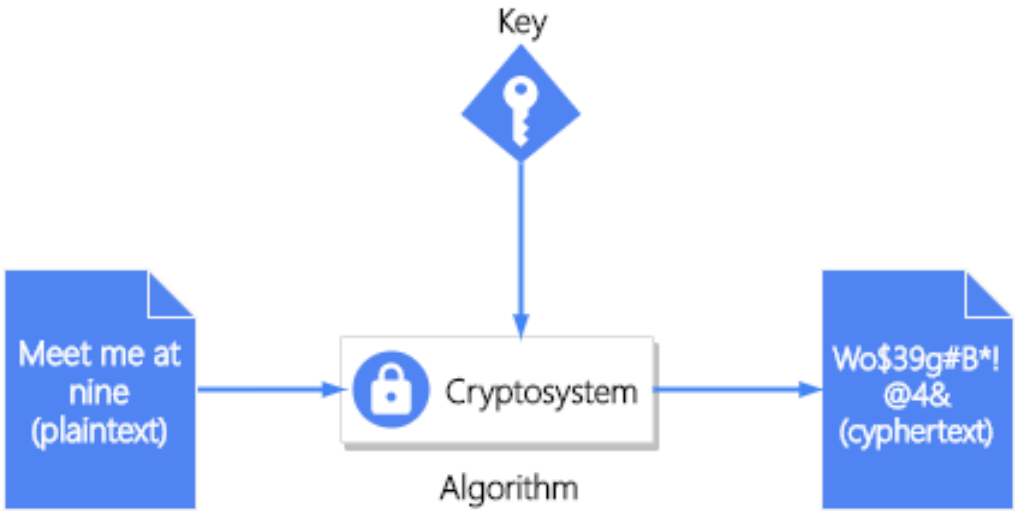

Figura 4 – Funcionamento de um sistema de criptografia.

Os algoritmos de criptografia com chaves simétricas são divididos em *block ciphers* (AES), nos quais um tamanho fixo de bytes são cifrados a cada iteração, e *stream ciphers* (Salsa20), que cifram a mensagem *byte* a *byte* ou bit a bit (KIM DAVID; SOLOMON, 2018, p. 296, 297).

O serviço da integridade visa prover mecanismos que permitem verificar se a mensagem enviada não foi alterada. Ou seja, quando o receptor receber a mensagem, ele será capaz de saber se a mensagem está íntegra ou alterada. Para isso são utilizadas funções *hashs* (SHA-1, SHA-2, SHA-3) que são algoritmos unidirecionais em que a informação

gerada tem sua integridade representada por uma cadeia de bytes chamados de resumo, *digest* ou somente *hash*. Esses algoritmos têm o propósito de evitar a colisão entre *digests* de diferente mensagens. Isso consiste no cenário em que duas ou mais mensagens diferentes não gerem o mesmo *digest*, por sua vez, algoritmos tradicionais, como o MD5 (SELINGER, 2006) e SHA-1 (STEVENS et al., 2017), já foram demonstrados como susceptíveis a esse problema, e por isso têm sido substituídos.

Os serviços de autenticidade e não-repúdio estão bastante próximos. A autenticidade é a garantia de que a informação recebida é autêntica e do transmissor esperado. Caso um transmissor A, desconhecido, envie uma mensagem como sendo o transmissor B, o receptor é capaz de identificar a a falsa identidade do emissor como não sendo B. O não-repúdio, por sua vez, tem como objetivo prover a impossibilidade do emissor de negar a autoria de uma mensagem enviada por ele. Se um transmissor enviou uma mensagem é possível verificar o remetente, logo o transmissor não tem como negar que gerou tal mensagem. Para isso é utilizado o mecanismo de assinatura digital (KIM DAVID; SOLOMON, 2018, p. 294).

A assinatura digital vincula uma mensagem a uma entidade, ou seja, os serviços de integridade, autenticidade e não-repúdio são garantidos. Para o uso da assinatura digital, é necessário um algoritmo de criptografia de chave pública e um algoritmo de *hash*, onde o *hash* da mensagem é processado com a chave privada, como mostra a Fig. 5.

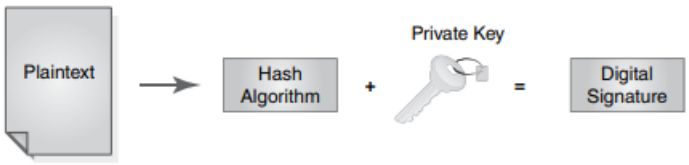

Figura 5 – Funcionamento do algoritmo de assinatura digital (KIM DAVID; SOLOMON, 2018, p. 312).

2.3 Alguns algoritmos viáveis para segurança em IoT

Alguns algoritmos foram previamente mencionados, sendo eles o RSA e o AES. Há algumas questões com relação à aplicação desses algoritmos em IoT. O RSA exige uma chave com tamanho considerável para oferecer nível aceitável de segurança e o AES utiliza operações com matrizes, ou seja, ambos usam recursos que são escassos em dispositivos IoT ou exigiria mais processamento e, consequentemente, impactando negativamente o desempenho da execução de tarefas para as quais um dispositivo tenha sido designado em função de segurança.

A seguir serão apresentados algoritmos viáveis para aplicação em sistemas IoT.

2.3.1 Salsa20

Representa uma família de *stream ciphers* com chaves de 256 bits, suas variantes têm diversas quantidades de rodadas até 20. O Salsa20 foi criado com o propósito de ser mais rápido que o AES e oferecer nível similar de segurança (BERNSTEIN, 2005, p. 1).

Sendo um *stream cipher*, a cifra é feita byte a byte com operação de *or* exclusivo de um byte K com a entrada P resultando na saída do byte cifrado C, como mostra a Fig. 6. O algoritmo é, então, aplicado a uma sequência de bytes, resultando em uma sequência cifrada.

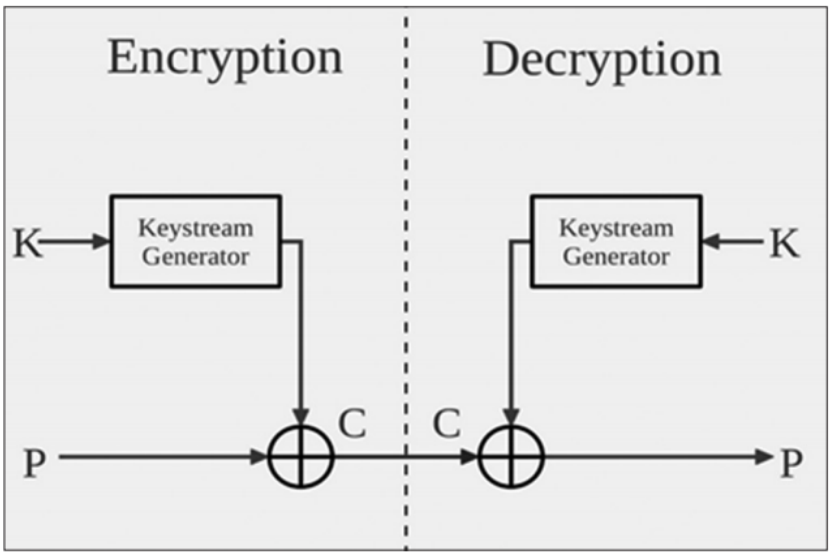

Figura 6 – Diagrama de um *stream cipher* Asaad, Abdulrahman e Hani (2017, p. 41).

O Salsa20 tem uma especificidade nas operações de geração do *Keystream*, para isso há uma entrada de 16 palavras, sendo cada palavra composta por 32 bits, o que corresponde ao comprimento de K com 512 bits.

As entradas para o Salsa20 exigem 256 bits para a chave, 128 bits denominado constante, 64 bits para o *nonce* e 64 bits de contador sequencial. Portanto, os parâmetros efetivos de entrada são chave e *nonce*, onde o *nonce* é um número que não deve ser repetido no par chave-*nonce*.

A Figura 7 representa a estrutura das posições em forma de matriz do Salsa20, sendo o primeiro elemento equivalente à posição 0 e o último à posição 15.

Constante	Chave	Chave	Chave
Chave	Constante	Nonce	Nonce
Posição	Posição	Constante	Chave
Chave	Chave	Chave	Constante

Figura 7 – Estrutura do *Keystream* do Salsa20.

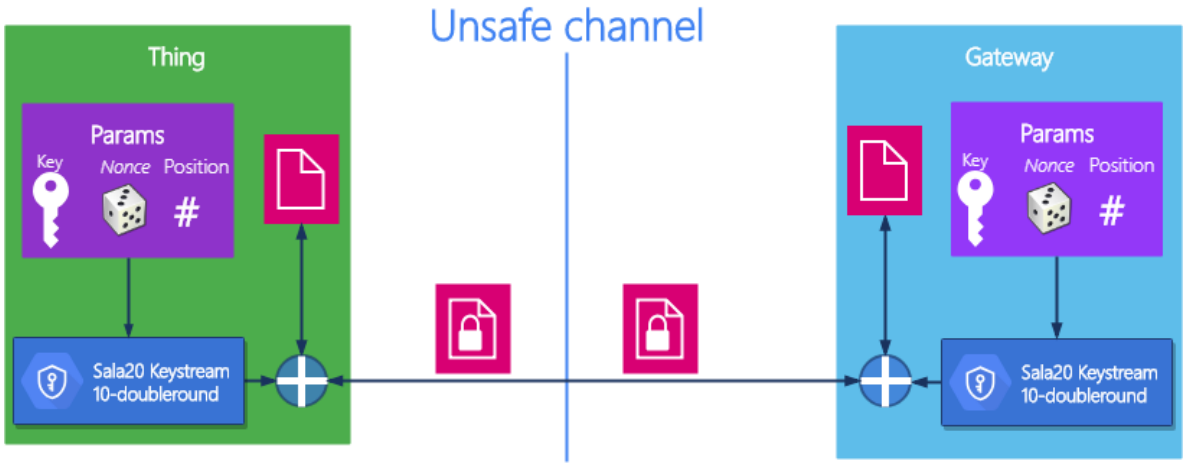

Figura 8 – Uso do Salsa20 entre *thing* e o *gateway*.

Os parâmetros de entrada definidos na Fig. 8 como "Params" são compostos pela chave, *Nonce* e *Position*. Esses três parâmetros devem estar sincronizados, ou seja, para decifrar uma mensagem, que tenha sido cifrada no Salsa20 com a chave compartilhada, é preciso usar o mesmo *nonce* e *position*. Sendo assim para o *keystream* é necessário que todos os parâmetros estejam sincronizados para que a mensagem possa ser decifrada corretamente. O parâmetro de posição, pode ser definido apenas como um incremento a cada cifra realizada, portanto emissor e receptor precisam apenas conhecer o valor inicial da posição.

Uma vez com as entradas recebidas, o Salsa20 realiza 10 rodadas de *doubleround function*, ou seja, dentro de uma rodada é feita duas rodadas com operações definidas pelo algoritmo. (BERNSTEIN, 2005, p. 12)

No Salsa20 são utilizadas operações de rotações e somas binárias que podem ser implementadas somente com portas lógicas, disso parte sua simplicidade computacional. Por exemplo, um Athlon com 900 MHz levou em média 10,04 ciclos por byte. (BERNSTEIN, 2005, p. 2)

2.3.2 SHA2-512

Representa uma função *hash*, denominado *Secure Hash Algorithm*-2 (SHA2) com uma saída de 512 bits. O SHA2 foi anunciado pela *National Institute of Standards and Technology* (NIST) em 2002, como um sucessor do SHA1 em resposta às fragilidades encontradas (SUN et al., 2007, p. 1).

Atualmente o SHA2 possui diversos tamanhos para o *hash*, mas comumente são mais utilizados tamanhos de 256 e 512 que oferecem uma complexidade computacional viável devido ao uso de operações modulares (SUN et al., 2007, p. 1).

Sendo um algoritmo *hash* bastante utilizado, há também muitos estudos relacionados à criptoanálise com complexidade de $2^{255,5}$ (DOBRAUNIG; EICHLSEDER; MENDEL, 2015, p. 1).

2.3.3 Curva elíptica

As curvas elípticas (EC) têm desempenhado um papel fundamental em técnicas avançadas de criptografia de chave pública. A partir do conceito de corpos finitos, pode-se escolher valores que mapeiam uma mensagem na curva elíptica, cuja estrutura está representada pela Eq. 2.1 (LARA; OLIVEIRA, 2010, p. 2).

$$y^2 = x^3 + ax + b. \tag{2.1}$$

A curva elíptica pode então ser utilizada como esquema de chave pública através de uma variante do protocolo Diffie-Hellman, que busca estabelecer uma troca de chave segura entre dois usuários. Para tanto, é necessário definir alguns parâmetros, escolher um número inteiro e aleatório, um gerador do grupo cíclico g e um número primo p para cada usuário. (LARA; OLIVEIRA, 2010, p. 2)

Por exemplo, Alice escolhe um número inteiro aleatório a e Bob escolhe um número inteiro aleatório b. Ambos compartilham o mesmo gerador g. Bob envia P_b para Alice e recebe P_a de Alice. Nesse caso ambos conseguem fazer um acordo da chave secreta K como mostram a Eq. 2.4:

$$P_a = g^a \bmod p \tag{2.2}$$

$$P_b = g^b \bmod p \tag{2.3}$$

$$K = (P_a)^b = (P_b)^a = g^{ab}. \tag{2.4}$$

Sua eficiência para criptografia se deve em usar parâmetros suficientemente grandes, o que resulta em enorme dificuldade computacional para se recuperar a a partir de g e p. O modelo usado para representar a segurança computacional das curvas elípticas é denominado Problema do Logaritmo Discreto em Curvas Elípticas (PLDEC) como mostra Lara e Oliveira (2010).

2.3.3.1 Curve25519

Há diversas curvas elípticas funcionais definidas pela listagem dos parâmetros da Equação 2.1. Uma delas, a Curve25519, consiste em um algoritmo para implementação de curva elíptica para suporte ao protocolo Diffie-Hellman, capaz de ser aproveitada em uma variedade de aplicações e caracterizada por uma boa velocidade de processamento.

A Equação 2.5 mostra os parâmetros de domínio da curva elíptica específica usada no Curve25519 e o número primo p é igual a $2^{255} - 19$: (BERNSTEIN, 2006)

$$y^2 = x^3 + 486662x^2 + x. \tag{2.5}$$

Ao se aplicar a estrutura da Curve25519, cada usuário tem uma chave pública e privada de 256 bits cada uma. Da mesma forma, cada usuário possui uma chave segura compartilhada também de 256 bits.

A Figura 9 mostra o funcionamento do Diffie-Hellman utilizando o Curve25519. Alice e Bob têm suas chaves secretas a e b, respectivamente, ambos então obtêm suas chaves públicas com o Curve25519 e as compartilham entre si. Alice então usa a chave pública de Bob na primitiva do Curve25519 com sua chave secreta e obtém a chave segura compartilhada; analogamente, Bob realiza o mesmo procedimento e ambos alcançam consenso quanto a uma mesma chave segura.

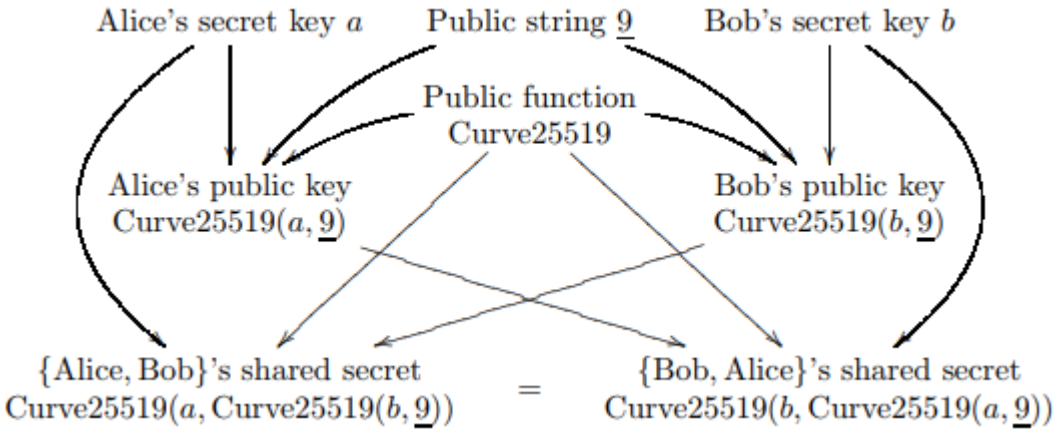

Figura 9 – Acordo de chaves com Curve25519. (BERNSTEIN, 2006)

De acordo com Bernstein (2006), o algoritmo do Curve25519 utiliza instruções comuns de pontos flutuantes para realizar rápidas multiplicações e adições no campo finito da curva elíptica, usando o primo p definido pelo autor.

2.3.3.2 Ed25519

Ed25519 consiste em um algoritmo de assinatura digital por meio do algoritmo *Edwards-curve Digital Signature Algorithm* (EdDSA), ou seja, o Ed25519 é apenas uma variante do EdDSA. (BERNSTEIN et al., 2011)

De acordo com Bernstein et al. (2011), o Ed25519 mantém algumas características do Curve25519 em relação à curva elíptica, principalmente pelas questões do PLDCE, que também se aplica ao Ed25519. O nome do algoritmo também é devido o valor do

número primo $p = 2^{255} - 19$ e uma chave pública de 256 bits. A função *hash* utilizada é o SHA2-512.

Assim como destacado no algoritmo do Curve25519, o Ed25519 também utiliza instruções do CPU para realizar os processos de assinatura e verificação mais rápida que outros EdDSA. (BERNSTEIN et al., 2011)

2.3.3.2.1 Assinatura ECDSA

Para a assinatura digital em curva elíptica, são usados os seguintes parâmetros: o ponto base na curva G, a ordem do número primo n, a mensagem a ser assinada m e a chave privada d. Primeiro é calculado o *digest* $h(m)$ com uma função *hash*, em seguida é escolhido um número inteiro aleatório k maior que 1 e menor que n e então é calculado um ponto na curva como mostra a Eq. 2.6. Após obter o ponto, é calculado o parâmetro r conforme a Eq. 2.7 e, por fim, é calculado o parâmetro s a partir da Eq. 2.8. O par (r,s) são os parâmetros para a assinatura digital a serem enviados. (KHALIQUE; SINGH; SOOD, 2010)

$$(x_1, y_1) = K \cdot G \tag{2.6}$$

$$r = x_1 \bmod n \tag{2.7}$$

$$s = K^{-1}(h(m) + r \cdot d) \bmod n \tag{2.8}$$

A Figura 10 ilustra a assinatura para o transmissor.

2.3.3.2.2 Verificação ECDSA

A validação da assinatura digital precisa dos parâmetros r, s, a ordem do número primo n, o ponto base na curva G e a chave pública do transmissor Q. Primeiro é calculado w a partir dos parâmetros s conforme a Eq. 2.9, depois são calculados u_1 e u_2 como mostram as Eqs. 2.10 e 2.11, respectivamente. Logo então é calculado o ponto na curva a partir dos parâmetros anteriores, de acordo com a Eq. 2.12. Por fim, se o ponto x_2 for igual ao r, recebido pelo transmissor, a assinatura digital é válida para a mensagem m, do contrário é inválida. (KHALIQUE; SINGH; SOOD, 2010)

$$w = s^{-1} \bmod n \tag{2.9}$$

$$u_1 = [h(m) \cdot w] \bmod n \tag{2.10}$$

$$u_2 = (r \cdot w) \bmod n \tag{2.11}$$

$$(x_2, y_2) = u_1 \cdot G + u_2 \cdot Q \tag{2.12}$$

A Figura 10 mostra a validação da assinatura digital para o receptor.

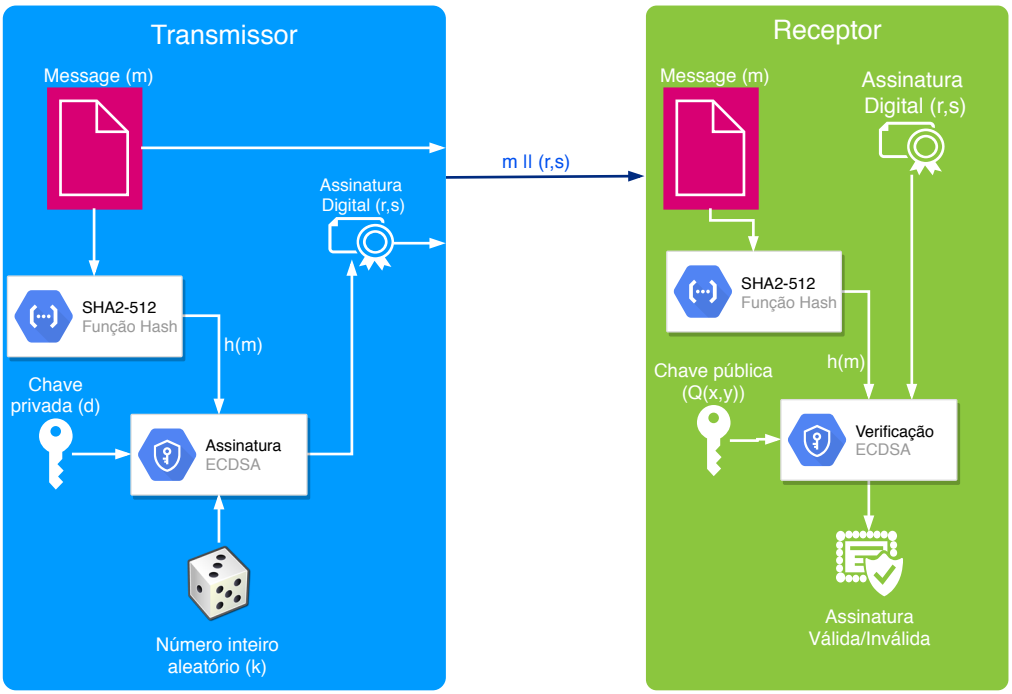

Figura 10 – Operação e validação de uma assinatura digital em curva elíptica.

2.4 Desempenho em dispositivos embarcados

Como já mencionado, há diversos algoritmos que oferecem segurança, entretanto alguns demandam muito processamento e memória, algo valioso para dispositivos IoT. Entre esses algoritmos bastante utilizados estão o RSA, algumas variantes de ECC, AES e o Salsa20. Abaixo, apresentaremos resultados de estudos que se preocuparam com a comparação de desempenho das implementações desses criptossistemas.

2.4.1 ECC vs. RSA

Uma vantagem significativa do ECC em comparação ao RSA, de acordo com Bafandehkar et al. (2013, p. 2), é o tamanho da chave pública. Enquanto o RSA demanda uma chave relativamente grande para um nível elevado de segurança, o ECC demanda muito menor como mostra a Tab. 1 e a Fig. 11.

Claramente, há uma grande diferença no tamanho das chaves, algo fundamental em dispositivos IoT. Desta forma, um volume menor da memória é utilizado para armazenar a chave pública, liberando mais recursos para a tarefa designada do dispositivo.

Essa diferença no tamanho da chave não só é significativa para o dispositivo IoT como também para o *gateway*, considerando que há diversos nós conectados à rede e que

Tabela 1 – Equivalência do tamanho das chaves públicas e chave secreta em bits. (BAFANDEHKAR et al., 2013, p. 2)

RSA	ECC	Tamanho chave secreta
1024	160	80
2048	224	112
3072	256	128
7680	384	192
15360	512	256

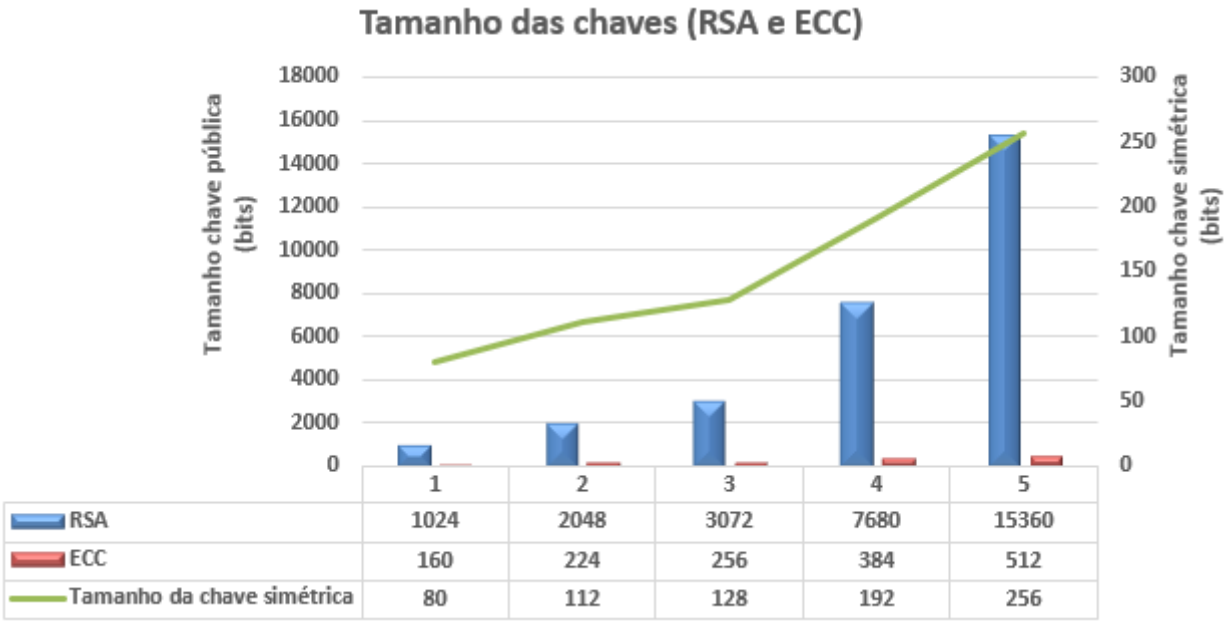

Figura 11 – Comparação entre o tamanho da chave pública entre o RSA e o ECC.

cada um possui sua respectiva chave pública. Isso implica em economia do *gateway* no armazenamento dessas chaves para verificar autenticidade.

Por exemplo, se a rede tem conectada 100 dispositivos IoT para um *gateway*, nesse caso, considerando o RSA para verificar a autenticidade de cada nó e um tamanho de chave pública de 3072 bits, seriam necessários 38400 bytes da memória do *gateway* para armazenar todas as chaves. Com a proteção equivalente em ECC, seriam necessários 3200 bytes.

De acordo com ECRYPT II (2018b), foram realizados testes de diversas implementações em várias plataformas. Para este trabalho, há o interesse em relação aos resultados relativos à plataforma ARM, muito utilizada por dispositivos embarcados como é o caso do BCM2836 utilizado no Raspberry Pi 2. Os algoritmos relevantes são o Curve25519, Ed25519 e o ronald3072, que utiliza RSA com 3072 bits para chave pública.

Para assinatura digital, é possível fazer a comparação do Ed25519 com o ro-

nald3072, onde a Tab. 2 mostra uma clara diferença nos ciclos de *clock* para cada algoritmo.

Tabela 2 – Comparação dos ciclos de *clock* entre ECC e RSA para assinatura digital. (ECRYPT II, 2018b)

Algoritmo	**Gerar chaves**	**Assinar 59 bytes**	**Verificar 59 bytes**
Ed25519	509646	520743	1523344
ronald3072	1304580217	114973425	779958

Com relação ao ECDH com o Curve25519, a Tab. 3 mostra quantos ciclos são gastos para gerar o par de chaves e computar a chave compartilhada.

Tabela 3 – Ciclos de *clock* gastos para o Curve25519 (ECRYPT II, 2018a)

Algoritmo	**Gerar par de chaves**	**Computar chaves compartilhadas**
Curve25519	937922	925896

2.4.2 Salsa20 vs. AES

Apesar de o AES ser bem consolidado para criptografia de chave simétrica, o Salsa20 tem mostrado melhores resultados para dispositivos embarcados. Enquanto o AES utiliza operações separadas por rodadas em matrizes, o Salsa20 utiliza operações mais simples em termos de instruções para o CPU.

Uma variante do Salsa20 é o Salsa20/8, onde são feitas apenas 8 rodadas em vez das 20. (BERNSTEIN, 2005, p. 1)

De acordo com Aumasson, Fischer e Khazaei (2008, p. 2), a complexidade do melhor ataque conhecido para o Salsa20/8 foi de 2^{251} operações, enquanto, conforme Bogdanov, Khovratovich e Rechberger (2012, p. 3), o AES-256 com 9 rodadas apresentou complexidade próxima de 2^{252}.

A Tabela 4 mostra o desempenho de ambos os algoritmos operando com 256 bits de chave simétrica e usando como métrica a quantidade de ciclos por byte utilizada para o processamento de uma mensagem de 64 bytes. Os testes foram realizados no BCM2836, em uma plataforma Raspberry Pi 2.

Tabela 4 – Comparação entre Salsa20/8 e AES-256. (ECRYPT II, 2018c)

Algoritmo	**Ciclos por byte**
AES256ctr	83,92
Salsa20/8	14,66

De acordo com a Tab. 4, há uma clara vantagem do Salsa20/8 em relação ao AES para o Raspberry Pi 2, que é uma plataforma amplamente utilizada em projetos de eletrônica embarcada.

3 Materiais e Métodos

A metodologia utilizada foi a *waterfall* para atender os objetivos específicos dividido em etapas sequenciais. Para alguns dos objetivos específicos serem cumpridos, foi implementado um protocolo simples o suficiente para IoT, que forneça os serviços básicos de segurança da informação a partir dos algoritmos citados. Em seguida foi embarcado esse protocolo em dispositivos típicos de IoT, sendo composto pelo *gateway* e nós da ponta (*things*), para isso são necessários os materiais listados abaixo.

Dentro ainda da metodologia, foram explorados possíveis ataques que o protocolo possa vir a sofrer com intenção de prejudicar a rede, em especial o *gateway*. Para isso será observado o comportamento do sistema sob ataque e então apresentadas contramedidas que protegem o sistemas desses ataques.

3.1 Materiais e recursos necessários

Os materiais utilizados foram dispositivos para eletrônica embarcada. Portanto, são utilizadas placas como NodeMCU (ESP8266, ESP32), Fig. 12, e Raspberry Pi 3, Fig. 13, como *gateway*, além de sensores e atuadores.

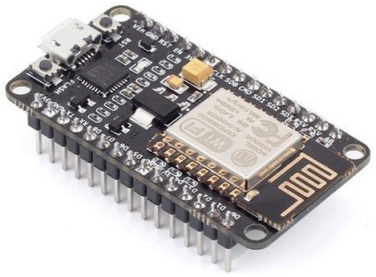

Figura 12 – Placa de desenvolvimento NodeMCU (ESP8266).

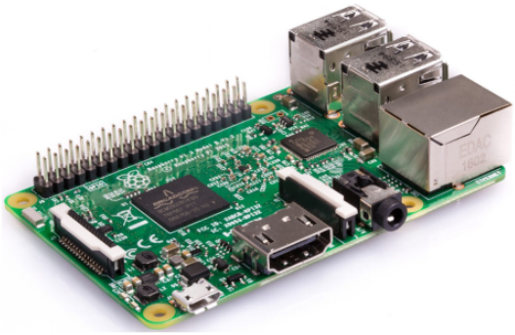

Figura 13 – Placa de desenvolvimento Raspberry Pi 3 Model B.

A implementação do protocolo foi realizada utilizando o sistema Raspbian para o Raspberry Pi 3 e o FreeRTOS para nós da ponta *things*. Foi necessário habilitar o encaminhamento de pacotes e NAT no *gateway*. Por fim, bibliotecas e funções que implementem os algoritmos utilizados pelo protocolo, como Salsa20, Ed25519, Curve25519 e SHA2-512. Para esses algoritmos, foram testadas várias implementações e selecionadas as que consumiram menos memória e tiverem um melhor tempo de resposta.

Para realizar a validação, medir latência e eficiência do protocolo, foi criada uma rede local *wireless* no Raspberry Pi 3 (*gateway*) e os dispositivos da ponta se conectarão à Internet através da rede Wi-Fi. O Raspberry Pi 3 recebeu os dados dos nós da ponta e os envia ao local designado, seja um *cloud*, um banco de dados, entre outros. O *gateway* foi conectado a outro roteador via cabo Ethernet que esse proveu acesso à Internet, como mostra a Fig. 14.

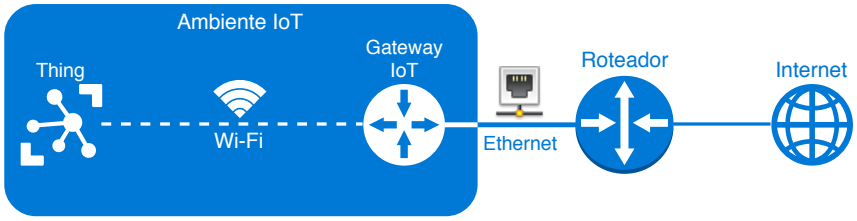

Figura 14 – Rede IoT através do Gateway até a Internet

3.2 Proposta de protocolo

O protocolo tem duas premissas, uma delas é a simplicidade para os nós da ponta com poucas interações para prover os serviços de segurança a outra é que tanto o *gateway* quanto o dispositivo têm conhecimento prévio das chaves públicas um do outro para a assinatura digital. A Fig. 15 mostra o diagrama de sequência para o protocolo para sensores. Abaixo será explicado detalhes do protocolo.

A primeira etapa é a de *Greeting*, onde serão acordados parâmetros necessários para a conexão segura do protocolo.

O dispositivo que se conecta ao *gateway*, denominado como *thing*, envia uma mensagem, assinada digitalmente usando o Ed25519 com a função DSA_t e a chave privada específica para o DSA, para o *gateway* que contém uma *flag* para conectar, junto com a chave pública $PubKey_t$, apenas para cifração, para ser usada no Curve25519.

O *gateway* armazena a chave pública, verifica a assinatura digital do *thing* com a função $ValidarDSA_t$ com a chave pública específica para assinatura previamente compartilhadas. Caso a assinatura seja válida, realiza o acordo de chaves com o Curve25519 usando a chave pública $PubKey_t$ e a chave privada $PrivKey_g$.

Em seguida o *gateway* envia uma semente $Seed$ junto com a sua chave pública para cifração $PubKey_g$, assinados digitalmente.

O *thing* verifica a assinatura digital usando a função $ValidarDSA_g$ com a chave pública também previamente compartilhada. Se a assinatura for válida realiza o Curve25519 com a chave pública recebida do *gateway* $PubKey_g$ e sua chave privada $PrivKey_t$, por fim inicia a posição Pos igual a 0 para ser usada pelo Salsa20 nos passos seguintes.

Nessa etapa, ambos os dispositivos têm uma chave de cifração compartilhada para ser usada no Salsa20, que ficou implícita no diagrama do protocolo junto com o *nonce*. Logo depois, o *thing* envia uma *flag* de *Ready*, no Salsa20, junto com a Pos de parâmetros, assinado digitalmente.

O *gateway*, inicia também o Pos igual a 0 para o Salsa20, a assinatura digital é validada na função $ValidarDSA_t$. Caso seja válida, aplica a função $ReSalsa20$ para decrifrar a mensagem recebida do *thing*.

A partir dos passos descritos acima, encerra-se a etapa de *Greeting*, e se todos os parâmetros foram devidamente trocados entre os dispositivos corretos, as mensagens em diante podem ser enviadas garantindo os serviços básicos de segurança.

As mensagens em diante são trocadas entre os dispositivos usando apenas o Salsa20 e uma função *hash* SHA2-512 da mensagem para garantir a integridade.

Por fim, caso o dispositivo deseje encerrar a conexão, envia uma flag de encerramento *end*, assinado e cifrado. O *gateway* valida a assinatura, caso válido, responde também com a mesma flag indicando que vai encerrar a conexão. Ambos os dispositivos liberam os recursos utilizados na comunicação.

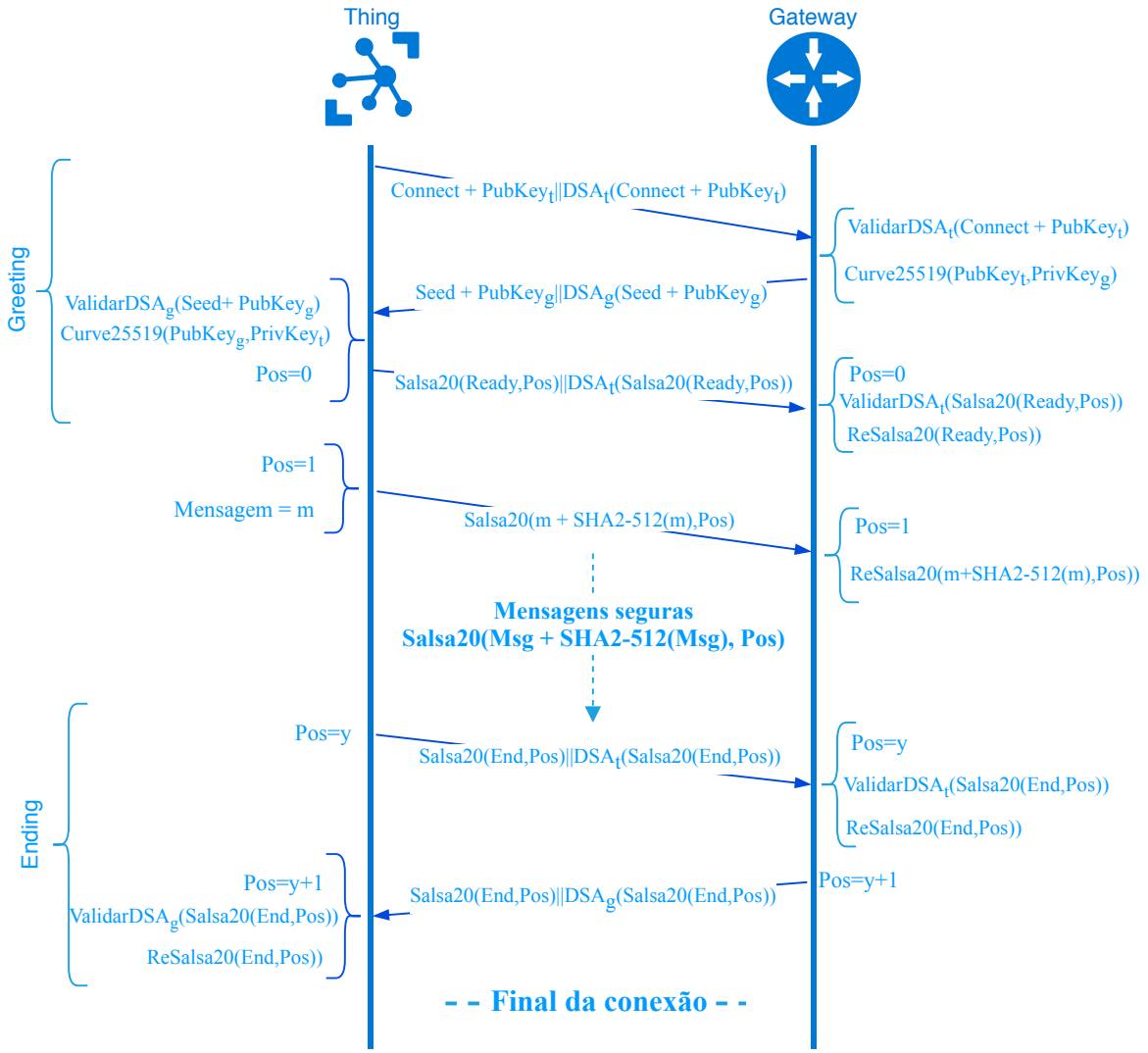

Figura 15 – Diagrama de sequência do protocolo.

4 Resultados e discussões

A implementação do protocolo e toda a análise foi feita no OpenRTOS, um *framework* aberto RTOS baseado no FreeRTOS específico para ESP8266 utilizando protocolo de transporte TCP e, no Gateway, foi utilizado o Qt 5.7, um framework em C++ muito utilizado por diversas empresas que desenvolvem na área de dispositivos embarcados e outras aplicações, outro motivo pela escolha do Qt é sua praticidade para desenvolvimento, aumentando a produtividade.

O protocolo foi implementado com base numa máquina de estados finita, iniciando no Greeting e passando aos passos seguintes como mostra a Fig. 16.

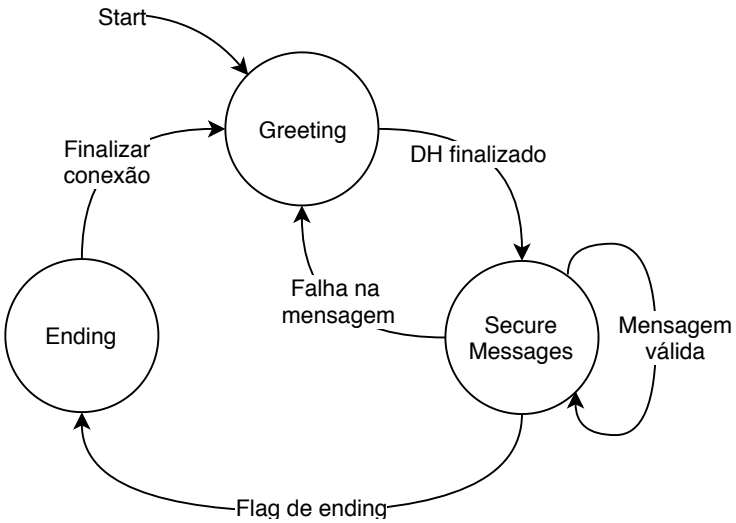

Figura 16 – Máquina de estados do protocolo implementado.

Na *thing* que foi utilizado o OpenRTOS, funciona a partir de *tasks* que permitem a execução de várias tarefas de forma concorrente. A rotina de *tasks* foi implementada seguindo o *User init* depois o *wifi_task* e por fim o *connect_gateway_task* como mostra a Fig. 17.

Figura 17 – Organização das tasks na *thing*.

Dentro do código da *thing* foram criadas 4 funções que forneciam toda a operação do protocolo como mostra a Fig. 18

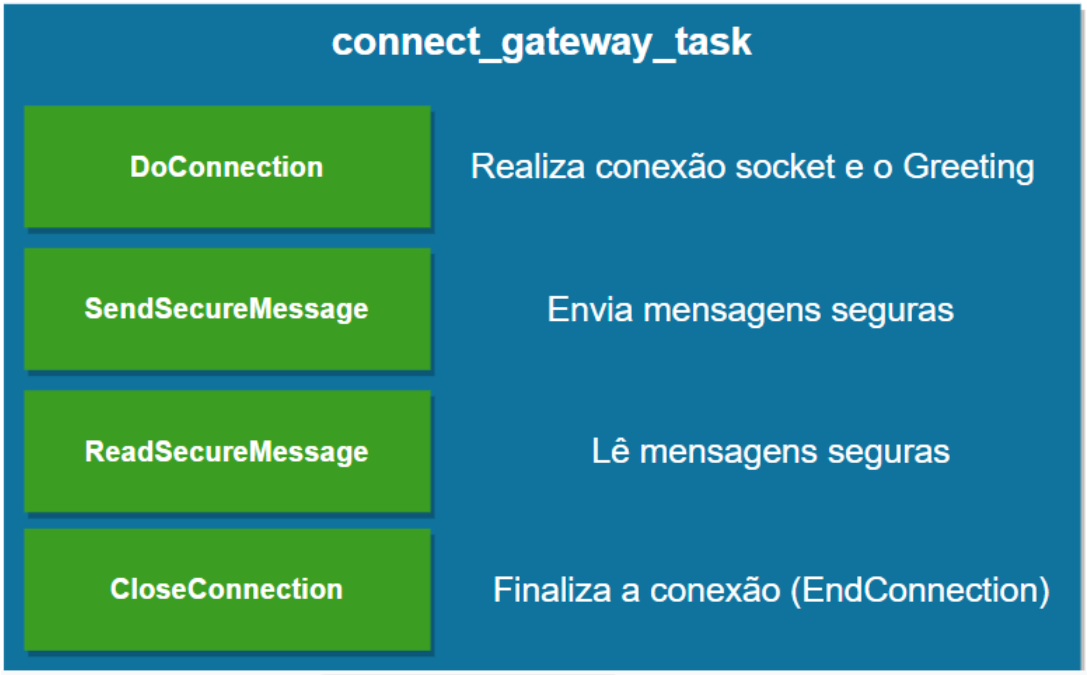

Figura 18 – Funções criadas para manipulação do protocolo.

A implementação do Gateway ficou semelhante ao mostrado para a *thing*, porém foram utilizados conceitos de $SIGNAL$ e $SLOT$ do Qt que são mecanismos de controle de eventos que ocorrem dentro do código quando ocorre um *trigger* que os dispara.

Para métrica de resultados, foram utilizados latência, consumo de memória e vazão em kB/s. Alguns parâmetros foram comparados com o TLS 1.2 usando a biblioteca mbedtls. Para o TLS, foi usado um servidor HTTPS apenas por conveniência em que era desconsiderado o tempo de processamento das instruções HTTP.

Por fim, foram analisados casos de mensagem curta (8 bytes) e longa (100 bytes) para o protocolo implementado. Ambos os valores foram escolhidos com base em tamanhos comuns para mensagens de dispositivos típicos de IoT.

O consumo de memória dinâmica na *thing* ficou em média de 3200 bytes enquanto estava conectado ao *gateway* enquanto o TLS usou 3400 bytes. Entretanto, isso se refere a valores dinâmicos alocados durante a execução. Para medidas de memória estática alocada em tempo de compilação, o TLS usou aproximadamente 29 mil bytes, representando 9 mil bytes a mais que o protocolo, com variação em variáveis locais utilizadas em ambos os códigos. Considerando que a memória do ESP8266 são $64kB$, essa diferença se torna bastante expressiva.

Na Tab. 5, é apresentado a latência para ambos os protocolos, comparando o *Handshake* do TLS e o Greeting do Protocolo desenvolvido. A análise foi feita utilizando a função *sdk_system_get_time* que retorna um inteiro desde que o sistema inicia em microssegundos.

Tabela 5 – Latência em ambos os protocolos.

Acordos	**Latência de acordo** $[ms]$
Greeting	950
Handshake	2498

Essa grande diferença se deve à simplicidade do Greeting, que não utiliza *cipher suite* e outros campos durante o acordo como no Handshake. Na primeira parte onde a *thing* envia a requisição de conexão são utilizados 4 bytes para ID, 32 bytes para o ECDH e 64 bytes da mensagem assinada garantindo ao Gateway que a conexão é de uma *thing* esperada. O mesmo tamanho se repete para resposta do Gateway a *thing*.

A comparação de envio e vazão em ambos os protocolos foi feita na mesma rede com um RTT medido através ICMP de aproximadamente 10 ms entre Gateway e *thing*. As medições foram feitos na placa ESP8266 considerando como objetivo central de análises do trabalho.

Na Tab. 6, é apresentado o tempo de envio para 50 mensagens curtas (8 bytes) e 50 mensagens longas (100 bytes). Essa quantidade de mensagens foi escolhida para se obter uma média de valores mais precisos, simulando o envio de várias informações do Gateway à *thing*.

Para a implementação de envio de mensagens seguras, por conveniência procura-se processar múltiplo de 64 bytes, sendo assim, em uma mensagem curta (8 bytes) são enviados 64 bytes preenchendo 0 nas posições restantes da mensagem. Para mensagens longas, são enviados 128 bytes, sendo 28 bytes preenchidos com 0 para a cifra com o Salsa20.

Tabela 6 – Latência em ambos os protocolos.

Protocolos	**Envio de 8 bytes** $[ms]$	**Envio de 100 bytes** $[ms]$
Protocolo IoT	63	91
TLS 1.2	58	115

Para o Protocolo IoT, obteve-se uma vazão de $49,6$ kB/s e $68,68$ kB/s para mensagens curtas e longas, respectivamente, considerando que a implementação envia múltiplos de 64 bytes. Em outro teste, cujo objetivo era medir a vazão do algoritmo Salsa20/8, obteve-se $892,86$ kB/s o que representa $89,60$ ciclos por byte (cpb) para 80 MHz de clock.

Na Tab. 4, é apresentado o tempo de recebimento para 20 mensagens curtas (8 bytes) e 20 mensagens longas (100 bytes). Essa quantidade de mensagens foi escolhida para não ocorrer um ataque de negação de serviço decorrente do envio de várias mensagens

em um pequeno intervalo de tempo.

Tabela 7 – Latência no Protocolo IoT.

Protocolos	**Recebimento de 8 bytes** $[ms]$	**Recebimento de 100 bytes** $[ms]$
Protocolo IoT	45	59

Os valores obtidos resultam em vazões de $27,78$ kB/s e $42,37$ kB/s para mensagens curtas e longas, respectivamente. Para o TLS, não foi possível forçar o servidor de teste a enviar somente 8 ou 100 bytes devido o tamanho que o HTTPS envia de payload. Sendo assim uma alternativa seria uma implementação do OpenSSL para testes nesse caso, mas poderia invalidar medidas realizadas anteriormente.

Para resultados em segurança do protocolo, um teste realizado foi usar chaves incorretas nas conexões. Obteve-se o resultado esperado: o não estabelecimento de conexão. Isso previne tentativas de ataques aos dispositivos. Outro tipo de ataque conhecido como *replay attack* foi prevenido, pois o protocolo utiliza-se de *nonces*.

Durante quedas de conexões, o protocolo se comportou tentando reestabelecer conexão, no caso da *thing*, ou liberando memória no caso do Gateway para conexão com a *thing*. Em casos de DDoS na *thing*, foi adotado a medida de contingência padrão do OpenRTOS que é reiniciar o dispositivo, considerando que um ataque desse tipo sobrecarrega o processamento da placa que pode influenciar na precisão das medidas de dispositivos IoT. Nesse caso, é reiniciando o dispositivo, as medidas são descartadas e são iniciados novamente os sensores.

5 Conclusão

O protocolo desenvolvido atingiu o objetivo de ser simples o bastante para dispositivos com baixo poder computacional, apresentando resultados melhores e em alguns casos próximos do TLS. Embora não tenha sido possível perceber diferenças de desempenho no cenário de mensagens curtas, para mensagens mais longas a diferença se torna mais significativa.

Quanto ao uso de memória, o protocolo apresentou melhores resultados que o TLS, mas devem ser feitas algumas considerações acerca da implementação do TLS. Seu uso de memória estática foi maior, um resultado procovado, provavelmente, pela forma que o TLS foi implementado no mbedTLS e pelo código de teste. Entretanto a menor quantidade de memória utilizada pelo protocolo IoT viabiliza o uso do microcontrolador para outras tarefas que poderiam ser mais difíceis utilizando o TLS considerando a pequena quantidade de memória que a *thing* possui.

Alguns pontos podem ser posteriormente investigados para obter melhor performance no protocolo. Por exemplo, o TLS usava SHA256 enquanto o protocolo desenvolvido utiliza SHA512. A escolha de uma hash maior foi considerada para uma longevidade do protocolo até que se torne obsoleto e necessite de uma nova versão.

A implementação do protocolo pode ser melhorada com alguns ajustes, por exemplo, no uso do Salsa20 que, para simplificação, trabalha com envio de mensagens múltiplos de 64 bytes, preenchendo com 0 para chegar aos tamanhos múltiplos. Porém, como apresentado nos resultados de vazão para o Salsa20, o tempo de processamento gasto no Salsa20 foi muito menor que o do SHA512. Desta forma essa simplificação traz uma vazão próxima do tamanho exato para a mensagem.

O OpenRTOS apresentou boas ferramentas e comportamento para desenvolvimento, sendo seu grande diferencial o uso de Tasks que permitem o processamento concorrente de tarefas. Além de possuir diversas bibliotecas que auxiliam o desenvolvimento e ser open source.

O Qt é um framework com pouca representatividade no meio acadêmico, mas com bastante no mercado, algo que produtos mais complexos e bem trabalhados usam pela facilidade de sua implementação e desempenho equivalente ao nativo, principalmente por utilizar C++.

O presente trabalho buscou o desenvolvimento de um protocolo que forneça segurança à rede e aos dispositivos sem influenciar significativamente na função que o dispositivo foi designado.

Referências

ASAAD, R. R.; ABDULRAHMAN, S. M.; HANI, A. A. Partial Image Encryption using RC4 Stream Cipher Approach and Embedded in an Image. 2017. Disponível em: <https://www.researchgate.net/publication/318517979_Partial_Image_Encryption_using_RC4_Stream_Cipher_Approach_and_Embedded_in_an_Image>. Citado 2 vezes nas páginas 11 e 25.

AUMASSON, J.-P.; FISCHER, S.; KHAZAEI, S. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. 2008. Disponível em: <https://eprint.iacr.org/2007/472.pdf>. Citado na página 32.

BAFANDEHKAR, M. et al. Comparison of ECC and RSA Algorithm in Resource Constrained Devices. 2013. Disponível em: <https://ieeexplore.ieee.org/document/6717816/>. Citado 3 vezes nas páginas 13, 30 e 31.

BERNSTEIN, D. J. The Salsa20 family of stream ciphers. 2005. Disponível em: <https://cr.yp.to/snuffle/salsafamily-20071225.pdf>. Citado 3 vezes nas páginas 25, 26 e 32.

BERNSTEIN, D. J. Curve25519: new Diffie-Hellman speed records. 2006. Disponível em: <https://cr.yp.to/ecdh/curve25519-20060209.pdf>. Citado 2 vezes nas páginas 11 e 28.

BERNSTEIN, D. J. et al. High-speed high-security signatures. 2011. Disponível em: <https://ed25519.cr.yp.to/ed25519-20110926.pdf>. Citado 2 vezes nas páginas 28 e 29.

BOGDANOV, A.; KHOVRATOVICH, D.; RECHBERGER, C. Biclique Cryptanalysis of the Full AES. 2012. Disponível em: <https://www.webcitation.org/68GTcKdoD?url=http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf>. Citado na página 32.

Cisco Systems, Inc. *Networking Fundamentals*. 2006. Disponível em: <https://www.cisco.com/c/dam/global/fi_fi/assets/docs/SMB_University_120307_Networking_Fundamentals.pdf>. Citado na página 21.

Cisco® Visual Networking Index™. *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021*. [S.l.], 2017. Disponível em: <https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf>. Citado na página 19.

DOBRAUNIG, C.; EICHLSEDER, M.; MENDEL, F. Analysis of SHA-512/224 and SHA-512/256. 2015. Disponível em: <https://eprint.iacr.org/2016/374.pdf>. Citado na página 27.

ECRYPT II. *Measurements of public-key Diffie–Hellman secret-sharing systems, indexed by machine*. [S.l.], 2018. Disponível em: <http://bench.cr.yp.to/results-dh.html>. Citado 2 vezes nas páginas 13 e 32.

ECRYPT II. *Measurements of public-key signature systems, indexed by machine*. [S.l.], 2018. Disponível em: <http://bench.cr.yp.to/results-sign.html>. Citado 3 vezes nas páginas 13, 31 e 32.

ECRYPT II. *Measurements of stream ciphers, indexed by machine*. [S.l.], 2018. Disponível em: <http://bench.cr.yp.to/results-stream.html>. Citado 2 vezes nas páginas 13 e 32.

GERBER, A. Top 10 IoT security challenges. Novembro 2017. Disponível em: <https://developer.ibm.com/dwblog/2017/iot-security-challenges>. Citado na página 20.

Integral PLM Experts. *Impacto de IoT na fabricação*. 2016. Disponível em: <http://integralplm.com.br/iot-internet-das-coisas/>. Citado 2 vezes nas páginas 11 e 21.

INTEL®. *The Intel® IoT Platform*. [S.l.], 2015. Disponível em: <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/iot-platform-reference-architecture-paper.pdf>. Citado 2 vezes nas páginas 11 e 22.

KHALIQUE, A.; SINGH, K.; SOOD, S. Implementation of Elliptic Curve Digital Signature Algorithm. 2010. Disponível em: <https://www.ijcaonline.org/volume2/number2/pxc387876.pdf>. Citado na página 29.

KIM DAVID; SOLOMON, M. G. *Fundamentals of information systems security*. 3. ed. [S.l.]: Jones & Bartlett Learning, 2018. (Jones & Bartlett Learning information systems security & assurance series). Citado 3 vezes nas páginas 11, 23 e 24.

LARA, P. C. da S.; OLIVEIRA, F. B. de. Curvas Elípticas: Aplicação em Criptografia Assimétrica. 2010. Disponível em: <https://www.lncc.br/~borges/doc/Curvas%20Elipticas%20-%20Aplicacao%20em%20Criptografia%20Assimetrica.pdf>. Citado na página 27.

ROSE, K.; ELDRIDGE, S.; CHAPIN, L. *THE INTERNET OF THINGS: AN OVERVIEW*. [S.l.], 2015. Disponível em: <https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf>. Citado na página 19.

SELINGER, P. *MD5 Collision Demo*. [S.l.], 2006. Disponível em: <https://www.mscs.dal.ca/~selinger/md5collision/>. Citado na página 24.

STEVENS, M. et al. The first collision for full SHA-1. 2017. Disponível em: <https://shattered.io/>. Citado na página 24.

SUN, W. et al. Design and Optimized Implementation of the SHA-2(256, 384, 512) Hash Algorithms. 2007. Disponível em: <https://ieeexplore.ieee.org/abstract/document/4415766>. Citado na página 26.