UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Gabriel Bristot Loli

Certificação Digital em Dispositivos da Internet das Coisas

Florianópolis
2024

Gabriel Bristot Loli

Certificação Digital em Dispositivos da Internet das Coisas

Trabalho de Conclusão de Curso do Curso de Graduação em Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Sistemas de Informação.
Orientadora: Prof. Thaís Bardini Idalino, Dra.
Coorientador: Gabriel Estevam de Oliveira, Me.

Florianópolis
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Loli, Gabriel Bristot
Certificação Digital em Dispositivos da Internet das Coisas / Gabriel Bristot Loli ; orientadora, Thaís Bardini Idalino, coorientador, Gabriel Estevam de Oliveira, 2024.
80 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Sistemas de Informação, Florianópolis, 2024.

Inclui referências.

1. Sistemas de Informação. 2. Internet das Coisas. 3. Certificação Digital. 4. Assinatura Digital. 5. Autenticidade. I. Idalino, Thaís Bardini. II. Oliveira, Gabriel Estevam de. III. Universidade Federal de Santa Catarina. Graduação em Sistemas de Informação. IV. Título.

Gabriel Bristot Loli

Certificação Digital em Dispositivos da Internet das Coisas

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Sistemas de Informação" e aprovado em sua forma final pelo Curso de Graduação em Sistemas de Informação.

Florianópolis, 29 de Novembro de 2024.

Prof. Álvaro Junio Pereira Franco, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Thaís Bardini Idalino, Dra.
Orientadora

Prof. Martin Augusto Gagliotti Vigil, Dr.
Avaliador
UFSC

Prof. Jean Everson Martina, Dr.
Avaliador
UFSC

AGRADECIMENTOS

Este trabalho não seria possível sem todo o apoio e o amor que recebi de meus pais, Cida e Valmor. A eles, um muito obrigado não bastaria para representar o quanto eu sou grato. Agradeço também aos meus irmãos, Bruno e Samuel, por sempre acreditarem em mim e serem meus exemplos de profissionais da área. A todas as amizades que fiz na universidade, em especial ao Higor, Luiza, Duda e Victor: muito obrigado, a minha formação não seria a mesma sem vocês. Por fim, agradeço à minha namorada, Isabela, por sempre estar ao meu lado, e por todo o apoio e motivação que recebi durante o desenvolvimento deste trabalho. A todos vocês, muito obrigado.

RESUMO

A evolução tecnológica nos últimos anos tem permitido o surgimento de diversos paradigmas tecnológicos disruptivos. Um deles é a Internet das Coisas, ou *Internet of Things* (IoT), um paradigma onde diversos dispositivos, "coisas" do dia-a-dia, como relógios de pulso e geladeiras por exemplo, possuem um computador embarcado que os permitem se comunicar com outros dispositivos, formando redes de compartilhamento de dados de grandes dimensões. Embora isso possa trazer muitos benefícios, estender a rede com esses dispositivos abre muitas possibilidades de ataques, visto que é mais difícil garantir a segurança numa rede tão espalhada e cujos dispositivos possuem pouco poder computacional para se protegerem adequadamente. Por isso, faz-se necessário a criação de soluções de segurança que garantam que os dados transmitidos por dispositivos IoT originem de um dispositivo legítimo, e que eles sejam íntegros e autênticos. Esses atributos podem ser conquistados com a utilização de algoritmos de certificação e assinatura digital. Entretanto, como dito antes, esses aparelhos possuem restrições mais altas de recursos, e a execução de algoritmos criptográficos complexos dentro deles é um desafio. Além do mais, notou-se que em trabalhos correlatos muito foco é dado na garantia da privacidade e confidencialidade dos dados comunicados pelo dispositivo IoT, e num contexto onde os dados são públicos, esses atributos não são interessantes. Portanto, este trabalho objetiva propor uma solução de certificação digital em dispositivos IoT que garanta a autenticidade e integridade dos dados transmitidos, bem como a legitimidade do dispositivo remetente, para permitir que dados públicos provenientes de aparelhos IoT possam ser transmitidos com segurança.

Palavras-chave: Internet das Coisas. Certificação digital. Assinatura digital. Autenticidade. Integridade.

ABSTRACT

The technological evolution in recent years has enabled the emergence of various disruptive technological paradigms. One of them is the Internet of Things (IoT), a paradigm where multiple devices — everyday "things" such as wristwatches and refrigerators, for example — have embedded computers that allow them to communicate with other devices, forming extensive data-sharing networks. Although this can bring many benefits, extending the network with these devices opens up many possibilities for attacks, as it is more challenging to ensure security in such a widespread network where the devices have limited computational power to adequately protect themselves. Therefore, it is necessary to create security solutions that ensure the data transmitted by IoT devices originates from a legitimate device and that the data is integral and authentic. These attributes can be achieved using digital certification and digital signature algorithms. However, as mentioned earlier, these devices have higher resource constraints, and executing complex cryptographic algorithms within them is a challenge. Furthermore, it has been noted in related works that a lot of focus is given to ensuring the privacy and confidentiality of the data communicated by IoT devices, and in a context where the data is public, these attributes are not of interest. Therefore, this work aims to propose a digital certification solution for IoT devices that ensures the authenticity and integrity of the transmitted data, as well as the legitimacy of the sending device, to allow public data from IoT devices to be transmitted securely.

Keywords: Internet of Things. Digital Certification. Digital Signature. Authenticity. Integrity

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O surgimento e o aperfeiçoamento de novos paradigmas tecnológicos nas últimas décadas afetou drasticamente a vida das pessoas. Dentre essas paradigmas, a Internet das Coisas, ou *Internet of Things* (IoT), é um que tem mudado substancialmente a forma como interagimos com o mundo. A Internet das Coisas pode ser definida como uma rede mundial de dispositivos, ou "coisas", interconectados e unicamente endereçáveis, capazes de trocar dados entre si (LI; XU; ZHAO, 2015; ATZORI; IERA; MORABITO, 2010). Os dados transmitidos são normalmente capturados dos arredores dos dispositivos IoT por meio de seus sensores, o que possibilita que essa tecnologia seja aplicada em diversos contextos, como em aplicações industriais, na área da saúde, e até mesmo na infraestrutura de cidades (LI; XU; ZHAO, 2015).

Essa interconexão entre bilhões de dispositivos traz inúmeras aplicações benéficas, como seu uso no meio médico, onde sensores vestíveis ou implantados monitoram as condições fisiológicas dos pacientes, como exemplificado por Liu *et al.* (2022). Entretanto, essa mesma interconexão acaba também aumentando a área de superfície para ataques, gerando diversas problemáticas de segurança, entre elas algumas relacionadas principalmente à integridade dos dados e à autenticação (ATZORI; IERA; MORABITO, 2010). Por isso, é importante conseguir meios de garantir a integridade e a autenticidade dos dados transmitidos, bem como de comprovar a identidade do dispositivo IoT remetente, e isso pode ser conquistado por meio de assinaturas e certificados digitais (BELLARE; ROGAWAY, 2001; KATZ, 2010; AHMED; BARUKAB, 2022).

Contudo, dispositivos IoT possuem normalmente baixo poder computacional e restrições de recursos como memória e energia (DHANDA; SINGH; JINDAL, 2020), o que torna a execução de algoritmos criptográficos dentro deles um desafio, ainda mais algoritmos complexos que envolvem criptografia assimétrica, que é o caso dos usados para gerar assinaturas digitais (YAVUZ; OZMEN, 2019; MAMUN; RANA, 2019). Felizmente, encontrou-se na literatura casos em que algoritmos criptográficos foram utilizados em IoT com sucesso (FAUZAN; SUKARNO; WARDANA, 2020; PORAMBAGE *et al.*, 2014).

Porém, ficou evidente que as soluções propostas focam principalmente na garantia da privacidade e confidencialidade dos dados transmitidos, o que faz sentido para as aplicações abordadas, mas que não se aplica num contexto em que os dados são públicos. Também, não foram encontrados trabalhos cujo o único objetivo fosse a garantia de autenticidade e integridade dos dados provindos de dispositivos IoTs. No contexto de dados públicos, a única preocupação é garantir que os dados sejam íntegros, autênticos, e que o dispositivo IoT remetente seja legítimo. Como exemplo, pode-se citar as novas bombas de combustíveis regulamentadas pelo INMETRO (Instituto Nacional de Metrologia, Qualidade e Tecnologia), em que os dados do último abastecimento dos consumidores são públicos, e é fundamental garantir que eles provêm de uma bomba de combustível legítima.

A verificação de assinaturas digitais, nesse caso, permite identificar fraudes em bombas de combustíveis (INMETRO, 2023). Motivados por essa aplicação de sucesso, almeja-se entender a possibilidade de integrar certificação digital em outros dispositivos IoT.

O objetivo dessa integração é ter um IoT padrão certificado que poderia, potencialmente, ser um semáforo de trânsito, uma balança, um medidor de energia. Num contexto destes, precisamos de uma proposta que utilize algoritmos de assinatura digital bem estabelecidos, que sejam ou que tenham mais chances de serem padronizados e aceitos a nível nacional. A utilização de novos esquemas de assinatura digital é interessante no contexto científico, incentivando melhor desempenho, mas que no contexto deste trabalho, para que a assinatura tenha validade jurídica no país, é necessário seguir as recomendações impostas. Por conta disso, este trabalho busca propor uma solução de certificação digital em IoT, utilizando algoritmos de assinaturas digitais aceitos pelo ITI (Instituto Nacional de Tecnologia da Informação) (ICP-BRASIL, 2022), para garantir a autenticidade e integridade na emissão e transmissão dos dados capturados pelos sensores medidores do dispositivo.

1.1 OBJETIVOS

1.1.1 Pergunta de pesquisa

Neste trabalho, almejamos responder à seguinte pergunta de pesquisa:

- Como é possível atribuir autenticidade e integridade dos dados emitidos por dispositivos IoT?

1.1.2 Objetivo Geral

Propor uma solução de certificação digital em dispositivos IoT de medição, que garanta a integridade e autenticidade dos dados transmitidos.

1.1.3 Objetivos Específicos

- Selecionar algoritmos de assinatura digital que consigam ser executados dentro de dispositivos com limitações de recursos, como processamento e memória.
- Implementar uma prova de conceito de certificação digital em dispositivos IoT, sendo que o dispositivo deve ser capaz de assinar dados digitalmente.
- Realizar experimentos com os algoritmos escolhidos em um dispositivo IoT para medir o tempo de execução, consumo de memória e ciclos de clock.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão introduzidos os conceitos teóricos básicos para o entendimento da proposta do trabalho, como Internet das Coisas e conceitos relacionados à criptografia e certificação digital.

2.1 INTERNET DAS COISAS

A Internet das Coisas (IoT) é um paradigma da computação cuja definição varia dependendo da perspectiva adotada (ATZORI; IERA; MORABITO, 2010; ČOLAKOVIĆ; HADŽIALIĆ, 2018). Em uma síntese das várias definições possíveis, pode-se descrever IoT como sendo uma rede de objetos físicos interconectados, equipados com sensores, software e outras tecnologias, que são capazes de capturar e processar dados do mundo real para armazenamento ou compartilhamento (AL-FUQAHA *et al.*, 2015; WANG *et al.*, 2022; STALLINGS, 2020). Esses objetos são "coisas" do cotidiano, como geladeiras, lâmpadas e automóveis por exemplo, e podem pertencer a diversos contextos de aplicação, tais como automação residencial, saúde, cidades inteligentes, etc. Devido à sua natureza embarcada, os dispositivos IoT normalmente possuem pouco poder computacional, o que deve ser levado em consideração ao desenvolver-se soluções com estes dispositivos (DHANDA; SINGH; JINDAL, 2020; RAO; PREMA, 2019a; AL-FUQAHA *et al.*, 2015; ČOLAKOVIĆ; HADŽIALIĆ, 2018).

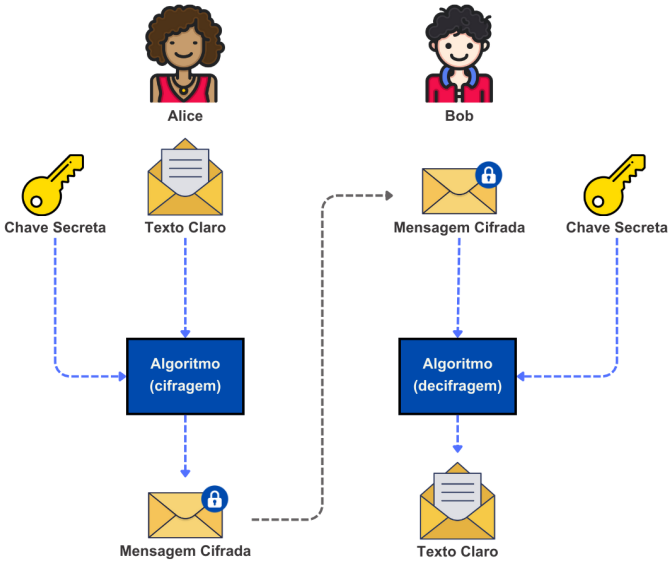
Por se tratar muitas vezes de redes de dispositivos ubíquos, é natural estarem expostas a diversos ataques, e ainda mais se os dispositivos estiverem conectados à Internet (DHANDA; SINGH; JINDAL, 2020; BHARDWAJ; KUMAR; BANSAL, 2017). Isso é preocupante, pois alguns dados manipulados pelos sensores IoT podem ser sensíveis, como no caso de implantes médicos por exemplo (WANG *et al.*, 2022); e outros dados, embora não sejam sensíveis, devem ser íntegros e autênticos (ATZORI; IERA; MORABITO, 2010). Assim, torna-se de suma importância garantir a segurança das informações geradas por estes dispositivos, e para isso faz-se necessário o uso de criptografia, cujos principais conceitos serão apresentados a seguir.

2.2 CIFRAGEM E DECIFRAGEM

Cifragem e decifragem são a base para qualquer sistema criptográfico. Dizem respeito, respectivamente, à transformação de dados, como uma mensagem em texto claro, para um formato ininteligível; e da transformação desta mensagem agora incompreensível de volta para sua forma original, de maneira que possa ser compreendida novamente (STINSON; PATERSON, 2019). Para exemplificar o processo, tomemos como exemplo a Figura 1. Nesta figura, temos duas pessoas, Bob e Alice, que desejam comunicar-se de maneira segura, a fim de garantir a confidencialidade da mensagem. Caso Alice escrevesse

uma mensagem para Bob sem cifrá-la, mantendo-a em texto claro, qualquer um que interceptasse a mensagem seria capaz de lê-la. Para evitar que isso aconteça, Alice utiliza uma chave secreta, conhecida somente por ela e por Bob, e um algoritmo criptográfico para "embaralhar" sua mensagem, criando uma nova mensagem que é ininteligível. Este processo é chamado de cifragem. A mensagem cifrada é então enviada para Bob, o qual utiliza a mesma chave secreta e algoritmo criptográfico usados por Alice para transformar a mensagem de volta em texto claro, tornando-a novamente compreensível e possibilitando Bob de entendê-la. Este processo é chamado de decifragem.

Figura 1 – Cifragem e decifragem.



Fonte: Autoria própria.

2.3 CRIPTOGRAFIA SIMÉTRICA E ASSIMÉTRICA

Criptografia simétrica e criptografia assimétrica referem-se a dois tipos diferentes de criptografia. A principal diferença entre elas tem relação com a "simetria" utilizada em seus nomes: na criptografia simétrica, a mesma chave é usada tanto para cifrar quanto para decifrar uma mensagem; enquanto que na criptografia assimétrica, duas chaves diferentes são utilizadas, uma para cifrar a mensagem, e outra para decifrá-la (STALLINGS, 2020).

2.3.1 Criptografia Simétrica

Entrando em mais detalhes, o exemplo dado anteriormente para demonstrar os conceitos de cifragem e decifragem corresponde à criptografia simétrica. Nela, tanto o

destinatário quanto o remetente de uma mensagem têm conhecimento da chave secreta. Consideremos um cenário onde Alice quer se comunicar com Bob de forma segura. Para cifrar uma mensagem, Alice fornece uma chave secreta S_k e a mensagem em texto puro M como entrada para um algoritmo de cifragem simétrica $E(.)$, gerando uma mensagem cifrada Y:

$$Y = E(S_k, M)$$

Um processo similar é usado para decifrar a mensagem: após receber Y, Bob fornece S_k e Y como entrada para um algoritmo de decifragem simétrica $D(.)$, resultando na mensagem original M.

$$M = D(S_k, Y)$$

Este processo é exemplificado na Figura 1. A segurança da criptografia simétrica está na suposição de que é computacionalmente inviável, para um adversário que possui conhecimento de uma mensagem cifrada e do algoritmo utilizado na cifragem, mas sem acesso à chave S_k, obter a mensagem original (STALLINGS, 2020). Além disso, é importante destacar também que nesta forma de criptografia, o algoritmo de decifragem da mensagem é o mesmo algoritmo usado para cifragem, porém revertido. Como exemplos de algoritmos, podemos citar o DES (*Data Encryption Standard*) e o AES (*Advanced Encryption Standard*) (STINSON; PATERSON, 2019; STALLINGS, 2020).

O modo de uso descrito anteriormente para criptografia simétrica garante a confidencialidade da mensagem, pois somente as partes comunicantes têm conhecimento da chave secreta e, logo, quem não conhece a chave secreta, em princípio não consegue decifrar a mensagem cifrada. Entretanto, algoritmos de criptografia simétrica que garantem confidencialidade, não garantem autenticidade; isto é, não garantem que a mensagem é de origem confiável (STALLINGS, 2020). No entanto, esta propriedade pode ser obtida ao se utilizar MACs (*Message Authentication Code*), que são algoritmos de criptografia simétrica que garantem autenticidade, mas não confidencialidade. Essa autenticidade, no entanto, é fraca, pois como tanto Alice como Bob conhecem a chave secreta, qualquer um deles pode ter gerado o MAC. Neste algoritmo, a remetente Alice fornece uma mensagem M e uma chave secreta S_k para uma função de MAC $F(.)$, que gera o MAC C_A:

$$C_A = F(S_k, M)$$

Então, C_A é anexado à M e enviado para o destinatário Bob, que, para verificar se a mensagem é autêntica, realiza a mesma operação em M para o cálculo de C_A por Alice, e obtém o MAC C_B. Por fim, Bob compara os MACs C_A e C_B e, se ambos forem iguais e se somente Alice e Bob conhecem S_k, a mensagem é autêntica e realmente teve sua origem em Alice. Um aspecto importante é que algoritmos de MAC, ao contrário de algoritmos de cifragem e decifragem, não são reversíveis.

Embora a criptografia simétrica é eficiente na tarefa que se propõe, por não necessitar de tamanhos de chaves muito grandes e por possuir algoritmos relativamente rápidos (STINSON; PATERSON, 2019), um desafio que ela enfrenta é: como fazer Bob e Alice entrar em acordo sobre qual chave secreta S_k usar, sendo que eles podem estar se comunicando apenas por canais inseguros, como a Internet? Para resolver este problema, foi criada a criptografia assimétrica.

2.3.2 Criptografia Assimétrica

Na criptografia assimétrica, também chamada de criptografia de chave-pública, há a utilização de um par de chaves: uma chave privada, mantida em segredo e conhecida somente pelo gerador do par; e uma chave pública, que como o próprio nome sugere, é mantida pública, e distribuída para os participantes da comunicação (STALLINGS, 2020; STINSON; PATERSON, 2019). Neste tipo de criptografia, uma das chaves é usada para cifrar a mensagem, enquanto a outra é usada para decifrá-la. Como exemplo, suponhamos que Alice deseja enviar uma mensagem M para Bob, mantendo a confidencialidade da mensagem, como mostra a Figura 2. Para isso, primeiro Alice cifra M utilizando a chave pública de Bob Pu_B e um algoritmo de cifragem assimétrica $E(.)$, obtendo a mensagem cifrada Y:

$$Y = E(Pu_B, M)$$

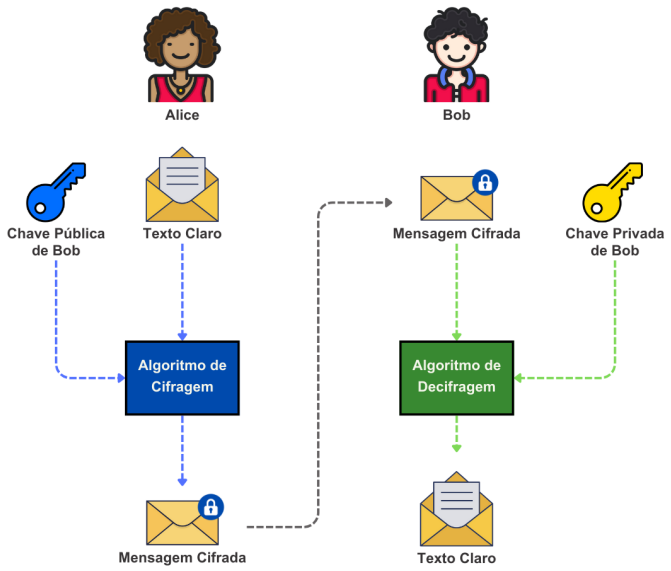
Então, Alice envia Y para Bob, que utiliza sua chave privada Pr_B e um algoritmo de decifragem assimétrica $D(.)$ para decifrar a mensagem, obtendo a mensagem original M:

$$M = D(Pr_B, Y)$$

Neste caso, a chave pública foi usada para cifrar a mensagem, o que garante que somente o possuidor da chave privada correspondente seja capaz de decifrá-la, conferindo confidencialidade à mensagem. No entanto, se Alice cifra M com sua chave privada Pr_A, como demonstra a Figura 3, Bob e todos que possuírem a chave pública de Alice, Pu_A, poderão obter M, o que não garante confidencialidade, mas provê a autenticidade e integridade da mensagem, pois se Pr_A é conhecido somente por Alice, então a mensagem só pode ter se originado de Alice e somente ela pode modificá-la (STALLINGS, 2020).

A segurança da criptografia assimétrica encontra-se em duas suposições: a primeira, de que é computacionalmente inviável para um adversário, com conhecimento da chave pública, determinar a chave privada correspondente; e a segunda, de que é computacionalmente inviável para um adversário, com conhecimento tanto da chave pública como dos algoritmos utilizados e de uma mensagem cifrada, obter a mensagem original (STAL-

Figura 2 – Uso de criptografia assimétrica para obter confidencialidade.



Fonte: Autoria própria.

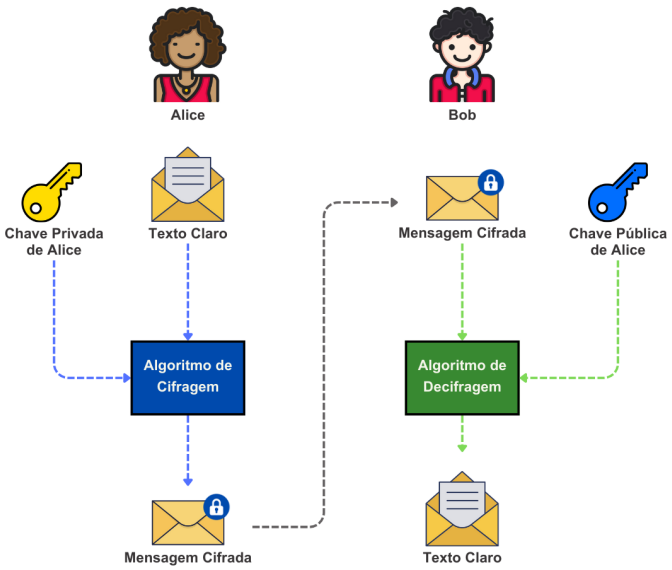
LINGS, 2020). É importante notar também que na criptografia assimétrica os algoritmos de cifragem e decifragem são distintos, diferentemente da criptografia simétrica.

2.4 HASH CRIPTOGRÁFICO

Uma função de hash é uma função $H(.)$ que aceita um bloco de dados de tamanho variável M e retorna um resultado de tamanho fixo $h = H(M)$. Hashes criptográficos são utilizados para verificar a integridade dos dados, pois qualquer mudança em M, por menor que seja, resulta em um valor de hash diferente (*avalanche effect*) (STALLINGS, 2020). Para uma função de hash ser considerada segura, isto é, para ser considerada uma função de hash criptográfica, ela deve atender a três propriedades. A primeira delas, é ser *resistente à pré-imagem*; isto é, deve ser praticamente impossível derivar a mensagem original a partir de um hash. Esta propriedade também é conhecida como propriedade do caminho único. A segunda propriedade, é ser *resistente à segunda pré-imagem*; isto é, dado uma mensagem e seu hash, deve ser computacionalmente inviável achar uma mensagem alternativa com o mesmo valor de hash. Por fim, a terceira e última propriedade é ser *resistente a colisões*; ou seja, deve ser computacionalmente inviável achar duas mensagens diferentes que gerem o mesmo hash (STALLINGS, 2020; STINSON; PATERSON, 2019).

Devido a essas características que as funções de hash podem exibir, elas são muito usadas em criptografia, como por exemplo nas funções de MAC, explicadas anteriormente,

Figura 3 – Uso de criptografia assimétrica para obter autenticidade.



Fonte: Autoria própria.

e em assinaturas digitais, explicadas a seguir.

2.5 ASSINATURA DIGITAL

Assinaturas digitais, assim como assinaturas escritas à mão, asseguram a autenticidade e a origem do que foi assinado (STINSON; PATERSON, 2019). Logo, assinaturas digitais são realizadas por meio de criptografia de chave-pública. O exemplo dado anteriormente, da Figura 3, ilustra o caso mais básico de assinatura digital, onde a mensagem cifrada inteira é utilizada como assinatura. Entretanto, essa não é uma forma muito eficiente de assinatura, pois se a mensagem for grande, o tempo para cifrá-la e decifrá-la também será maior. Para resolver este problema, deve-se assinar um hash da mensagem, a qual será transmitida em texto claro porém com a assinatura em anexo (STALLINGS, 2020). Além de resolver a questão do tempo, realizar o hash da mensagem também resolve outros problemas, especialmente os de alguns ataques para falsificação de mensagens.

Portanto, um algoritmo genérico de assinatura digital pode ser descrito da seguinte maneira, conforme ilustra a Figura 4: Alice deseja enviar uma mensagem para Bob, e quer que Bob tenha certeza de que a mensagem veio dela e que não sofreu alterações durante a transmissão. Para isso, Alice utiliza uma função de hash criptográfico segura, como SHA-512, para gerar o hash H_M da mensagem M. Para assinar a mensagem, ela fornece sua chave privada Pr_A e H_M como entradas para um algoritmo de geração de assinatura

$G(.)$, obtendo a assinatura S:

$$S = G(Pr_A, H_M)$$

Então, Alice envia M com S em anexo. Após receber a mensagem M', para verificar a assinatura, Bob calcula o hash de M', obtendo $H_{M'}$, e o fornece, juntamente com a chave pública de Alice Pu_A e S, como entrada para um algoritmo de verificação de assinatura digital $V(.)$, que retornará um resultado de *true* ou *false* dependendo da validade da assinatura. Em outras palavras:

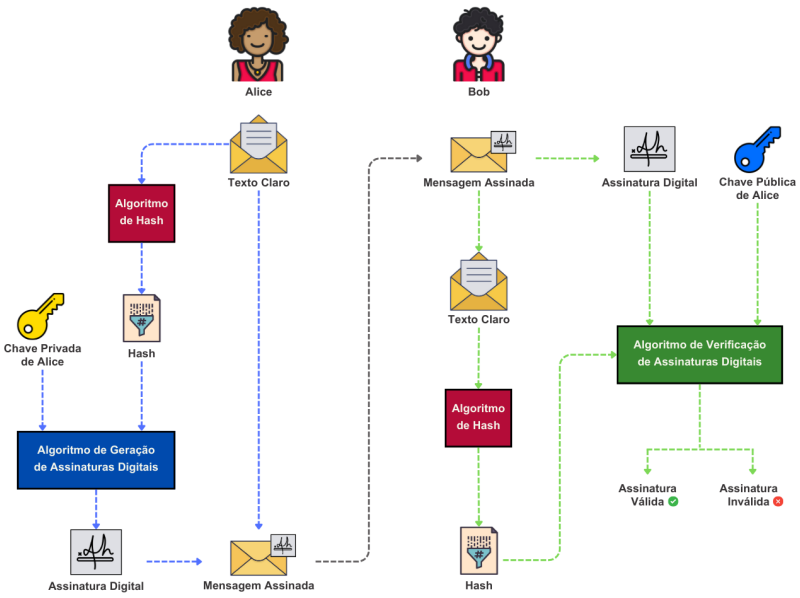
$$resultado = V(Pu_A, S, H_{M'})$$

Pode-se perceber que o processo descrito é similar ao realizado com um MAC. É importante notar, entretanto, que pela chave secreta de um MAC ser compartilhada entre as partes, ele não oferece a propriedade de não-repúdio; isto é, qualquer uma das partes envolvidas na comunicação pode alegar não ter enviado uma mensagem, pois a outra parte pode tê-la forjado. Isto não acontece com assinaturas digitais, pois como só o remetente da mensagem conhece sua chave privada, todos em posse da chave pública conseguem confirmar, pela verificação da assinatura digital, que a mensagem só pode ter se originado dele (STALLINGS, 2020). Além disso, caso o resultado seja *true*, também teremos a garantia de que a mensagem não foi alterada entre o momento de assinatura e o momento de verificação. Isso se dá pelo fato de que o hash da mensagem original H_M é comparado com o hash da mensagem recebida $H_{M'}$ dentro do algoritmo de verificação. Portanto, ao utilizar uma função de hash criptográfica segura, teremos também garantia de integridade da mensagem.

2.5.1 Algoritmos de Assinatura Digital

Há diversos algoritmos que realizam a geração e verificação de assinaturas digitais. Dentre eles, os mais populares são RSA, ElGamal, Schnorr, DSA (*Digital Signature Algorithm*) e ECDSA (*Elliptic Curve Digital Signature Algorithm*) (STALLINGS, 2020). Diferentemente dos demais, o RSA pode ser usado tanto como um esquema de assinatura quanto como um criptossistema de chave-pública, enquanto os outros algoritmos foram projetados especificamente para atuarem com assinaturas digitais. A principal diferença entre os algoritmos, além da operação, cuja explicação está fora do escopo deste trabalho, está no problema matemático usado como base para prover segurança. A segurança do RSA baseia-se na dificuldade prática da fatoração do produto de dois números primos grandes, enquanto que a segurança do ElGamal baseia-se na dificuldade de computar logaritmos discretos. Por sua vez, o algoritmo Schnorr é uma variante do ElGamal, em que o tamanho da assinatura é bastante reduzido, e o DSA é baseado tanto no ElGamal como no Schnorr; logo, esses algoritmos também têm o problema do logaritmo discreto

Figura 4 – Exemplo do uso de assinatura digital.



Fonte: Autoria própria.

como base matemática para segurança. Já o ECDSA é uma modificação do DSA para o contexto de curvas elípticas, e oferece segurança equivalente aos outros esquemas com tamanhos de chaves menores. Vale destacar também que o RSA, o DSA e o ECDSA são algoritmos padronizados pela NIST (NIST, 2023).

2.5.2 Padrões de assinatura

Existem quatro principais padrões de assinaturas: CAdES, XAdES, PAdES e JAdES. Antes de explicar a diferença entre eles, primeiro é preciso entender o conceito de AdES (*Advanced Electronic Signature*), ou assinatura eletrônica avançada. Uma assinatura eletrônica avançada é uma assinatura que atende aos seguintes critérios: (a) estar associada de modo único ao signatário; (b) permitir identificar o signatário; (c) ser criada utilizando dados para a criação de uma assinatura eletrônica que o signatário pode, com um elevado nível de confiança, utilizar sob o seu controle exclusivo; e (d) estar ligada aos dados por ela assinados de tal modo que seja detectável qualquer alteração posterior dos dados (UNIÃO EUROPEIA, 2014).

Os padrões supracitados nada mais são do que conjuntos de extensões para padrões específicos de arquivos para torná-los adequados para assinaturas eletrônicas avançadas, que permanecem válidas por longos períodos de tempo. Mais detalhadamente, o CAdES (*CMS Advanced Electronic Signature*) estende o padrão CMS (*Cryptographic Message Syntax*); o XAdES (*XMLDSig Advanced Electronic Signature*) estende o padrão XMLDSig

(*XML Digital Signature*); o PAdES (*PDF Advanced Electronic Signature*) estende o padrão PDF (*Portable Document Format*); e, por fim, o JAdES (*JSON Advanced Electronic Signature*) estende o padrão JWS (*JSON Web Signature*) (ICPBRASIL, 2015; BRZICA; HERCEG; STANČIĆ, 2013).

2.6 CERTIFICAÇÃO DIGITAL

Numa ocasião de uma rede com muitos membros, onde cada um deles possui um par de chaves e se comunicam entre si, faz-se necessário de um meio de autenticar as chaves públicas dos participantes. Uma maneira de fazer isso é criando uma Infraestrutura de Chaves Pública (ICP ou PKI), onde há uma ou mais Autoridades Certificadoras (AC) que irão assinar as chaves públicas de todas as pessoas da rede (STINSON; PATERSON, 2019). Suponhamos que Alice e Bob, membros dessa rede, estejam se comunicando, e Bob pede para que Alice prove que sua chave pública Pu_A é autêntica. Para isso, Alice deve ter um certificado válido emitido por uma AC de confiança que ateste que sua chave pública de fato pertence à Alice. Caso ela não tenha um certificado, ela poderá obter um enviando para a AC sua chave pública e uma informação identificadora ID_A (STALLINGS, 2020). Então, a AC irá fornecer como entrada para um algoritmo de geração de certificado $E(.)$ a assinatura S_A que ela gerou sobre Pu_A, ID_A, um *timestamp* T, e sua chave privada, Pr_{AC}, obtendo o certificado C_A:

$$C_A = E(Pr_{AC}, S_A)$$

Após a obtenção de seu certificado, Alice envia-o para Bob, que o fornece juntamente com a chave pública da AC Pu_{AC} para um algoritmo verificador de certificados $D(.)$, que retornará Pu_A, ID_A e T:

$$(T, ID_A, Pu_A) = D(Pu_{AC}, S_A)$$

Assim, Bob tem certeza que: o certificado veio da AC, pois ele é legível somente usando Pu_{AC}; que o certificado é atual, devido a T; e que o certificado pertence à Alice, devido a presença da informação identificadora que Alice forneceu (STALLINGS, 2020). Por fim, ele obtém também a chave pública de Alice, que ele pode comparar com a que ele possui para verificar se são iguais ou não.

2.6.1 Caminho de certificação

É válido ressaltar, entretanto, que no exemplo dado a AC de Alice pode não ser a mesma AC de Bob. Nesse caso, em que os dois estão associados a ACs diferentes, é importante que Bob tenha acesso à chave pública da AC de Alice de maneira de segura; caso contrário, ele não poderá verificar a assinatura do certificado de Alice (STALLINGS, 2020). Para garantir as chaves públicas sejam distribuídas com segurança entre ACs e

seus usuários, as ACs de Bob e Alice devem conseguir acesso, de forma segura, às chaves públicas uma da outra. Às vezes, as ACs de Bob e Alice podem estar subordinadas à ACs diferentes, que estão subordinadas à outras ACs, e assim por diante, formando uma cadeia, ou caminho, de certificação.

Para exemplificar como uma chave pública seria obtida nesta situação, vamos supor que Alice obteve seu certificado da AC X_1, e Bob obteve o seu da AC X_2. Também vamos supor que X_1 e X_2 obtiveram acesso às chaves públicas uma da outra de maneira segura, também por meio de certificados. Assim, Bob obtém a chave pública de Alice da seguinte forma: primeiro, ele obtém o certificado de X_1 assinado por X_2. Como Bob conhece com segurança a chave pública de X_2, ele consegue obter a chave pública de X_1 a partir de seu certificado, e consegue verificá-la devido à assinatura de X_2 presente no certificado. Depois, Bob acessa o certificado de Alice assinado por X_1. Já que Bob agora possui uma cópia confiável da chave pública de X_1, ele consegue verificar a assinatura do certificado de Alice e obter sua chave pública.

Vale destacar, também, que não é qualquer entidade que pode ser tornar uma AC; aqui no Brasil, por exemplo, nós confiamos nas ACs da ICPBrasil e da plataforma GOV.BR (Infraestrutura de Chaves Públicas Brasileira) (SÁ; SOUZA, 2024; UFSC, 2024). Logo, cadeias de certificação confiáveis devem ser compostas por ACs dessas entidades.

2.6.2 Componentes de um certificado digital

Para que o processo de verificação completa de uma assinatura por meio de uma cadeia de certificação ocorra, como a descrita anteriormente, é importante que todos os certificados estejam num mesmo padrão, contendo todas as informações necessárias. A União Internacional de Comunicações (UIT) estabeleceu o padrão X.509, que define o formato de certificados digitais (STALLINGS, 2020; ITU-T, 2019). Os componentes que formam um certificado digital, segundo esse formato, estão descritos a seguir:

- **Versão:** identifica a versão do formato do certificado digital;
- **Número de série:** um valor inteiro exclusivo na AC emissora associado ao certificado;
- **Identificador do algoritmo de assinatura:** algoritmo utilizado pela AC para assinar o certificado;
- **Nome do emissor:** o nome da AC que criou e assinou o certificado;
- **Período de validade:** intervalo de tempo durante o qual o certificado é válido;
- **Nome do sujeito:** o nome do usuário a quem o certificado pertence;
- **Informação de chave pública do sujeito:** a chave pública do sujeito, e um identificador do algoritmo para o qual essa chave deve ser usada;

- **Identificador exclusivo do emissor:** usado para identificar unicamente um emissor no caso de reuso de nomes;
- **Identificador exclusivo do sujeito:** usado para identificar unicamente o sujeito no caso de reuso de nomes;
- **Extensões:** um conjunto de uma ou mais extensões;
- **Assinatura:** abrange todos os outros campos do certificado; um dos componentes deste campo é a assinatura digital aplicada aos outros campos do certificado. Também inclui o identificador do algoritmo de assinatura.

Além desses componentes fixos, há diversas extensões que podem ser adicionadas aos certificados, por meio do campo "Extensões", a depender da necessidade. Essas extensões podem ser divididas em três categorias: (1) informações de chave e política, que trazem informações adicionais sobre as chaves do emissor e do sujeito, e indicadores da política do certificado; (2) atributos do certificado do sujeito e emissor, que trazem informações adicionais sobre o sujeito que reforçam sua identidade; e, finalmente, (3) restrições do caminho de certificação, que permitem que especificações de restrição sejam incluídas em certificados emitidos por ACs para outras ACs (STALLINGS, 2020; ITU-T, 2019).

2.6.3 Revogação de certificados

Um dos componentes presente no certificado é o período de validade, que determina um intervalo de tempo durante o qual o certificado é válido, como dito anteriormente. Entretanto, há três situações em que pode ser desejável que a validade de um certificado seja anulada antes de sua expiração: (1) quando se supõe que a chave privada do usuário foi comprometida; (2) quando o usuário não é mais certificado pela AC; e (3), quando se supõe que o certificado da AC foi comprometido (STALLINGS, 2020; ITU-T, 2019). Cada AC deve manter uma lista pública de todos os certificados revogados mas não expirados emitidos por ela, chamada de Lista de Revogação de Certificados (LCR). Cada LCR é assinada por sua AC, e deve conter: nome do emissor; data de criação da lista; data em que a próxima LCR está programada para ser emitida; e, por fim, uma entrada para cada certificado revogado. Cada entrada consiste no número de série do certificado e da data em que ele foi revogado.

3 REVISÃO SISTEMÁTICA DA LITERATURA

3.1 METODOLOGIA

Com o intuito de entender o estado atual da literatura relacionada ao uso de certificação digital em dispositivos IoT, realizou-se uma revisão sistemática da literatura sobre o tema.

A busca foi executada em quatro bases de artigos acadêmicos: IEEEXplore, ACM Digital Library, Springer Link e Google Scholar. A *string* de busca utilizada no campo de busca de cada um destes sites foi:

(“iot” OR “internet of things” OR “embedded systems” OR “arduino” OR “esp”) AND (“integrity” OR “authentic”) AND (“digital signature” OR “digital certificate” OR “message authentication code” OR “asymmetric” OR “public-key” OR “cryptography” OR “certification”).*

Os resultados das buscas, em todos os sites, foram filtrados por relevância. O processo de seleção dos trabalhos relacionados para revisão encontra-se resumido na Tabela 1. A Busca Inicial representa a quantidade inicial total de publicações resultantes da pesquisa pela *string* de busca supracitada. Na Seleção 1, selecionou-se os trabalhos acadêmicos possivelmente relacionados com o tema pelo título, em cada site de pesquisa, até que a busca não retornasse nenhum resultado interessante por algumas páginas; na Seleção 2, filtrou-se os artigos mais relevantes pelo *abstract*; por fim, na Seleção 3, os trabalhos resultantes foram lidos: alguns parcialmente, até notar-se pouca afinidade com o objetivo deste trabalho; e outros, que demonstravam maior afinidade, foram lidos por completo. Destes que foram lidos na íntegra, escolheu-se os mais relevantes.

Tabela 1 – Resultados da revisão sistemática da literatura

Site de pesquisa	**Busca Inicial**	**Seleção 1**	**Seleção 2**	**Seleção 3**
IEEXplore	6 671	175	25	2
ACM DL	3 153	29	3	1
Springer Link	2 251	49	5	0
Google Scholar	111 000	47	10	4
Total	**123 075**	**300**	**48**	**7**

3.2 TRABALHOS RELACIONADOS

3.2.1 Overhead Analysis of the Use of Digital Signature in MQTT Protocol for Constrained Device in the Internet of Things System

Fauzan, Sukarno e Wardana (2020) realizam uma análise do *overhead* gerado por assinaturas digitais no protocolo MQTT (*Message Queue Telemetry Transport*) em dispo-

sitivos IoT com restrições de recursos. Analisar o *overhead* significa, para este contexto, identificar e medir os recursos computacionais adicionais que o programa consome além do necessário. Neste caso, o *overhead* analisado foi o tempo de decifragem, a performance de verificação de assinatura, o tempo de envio da mensagem, o consumo de memória RAM e o consumo de memória Flash. Eles subdividem os dispositivos IoT utilizados em três classes, baseado na quantidade de memória RAM e *flash* de cada um. Notavelmente, eles utilizam um dispositivo ESP32, considerado da classe de maior memória, e um Arduino Uno com módulo WiFi, considerado da classe de menor memória. Os autores demonstram preocupação com confidencialidade, de forma que no esquema de comunicação proposto é implementada criptografia de ponta-a-ponta com o algoritmo AES (*Advanced Encryption Standard*), onde as mensagens são criptografadas no publicador e descriptografadas no consumidor. O hash da mensagem é assinado no ESP32, que atua como publicador de eventos, por meio do algoritmo EdDSA (*Edwards-curve Digital Signature Algorithm*) com a curva Ed25519. A verificação de assinatura, por outro lado, é realizada em um dispositivo de qualquer uma das classes, que atua como consumidor de eventos. Dentre os resultados relevantes para este artigo, resultantes da análise de *overhead*, está que dispositivos da classe de maior memória, como o ESP32, possuem performance de verificação de assinatura de menos de 2000 milissegundos, um resultado 14 vezes menor que dispositivos da classe de menor memória.

3.2.2 Light-weight hashing method for user authentication in Internet-of-Things

Rao e Prema (2019b) propõe um esquema de autenticação entre dispositivos IoT, utilizando uma versão modificada do método de hash de baixo custo computacional BLAKE2b, a qual chamam de cBLAKE2b, e o algoritmo de assinatura digital ECGDSA (*Elliptic Curve German Digital Signature Algorithm*). Os objetivos almejados pelo trabalho são reduzir o tempo de hashing da mensagem, projetar um esquema de assinatura digital leve, e reduzir o tempo de verificação da assinatura. Para realizar experimentos na solução, eles utilizam dois dispositivos Raspberry Pi 3, um com sensores, atuando como cliente, e outro atuando como servidor. A solução proposta consiste da etapa de geração de chaves, onde cada nó participante gera seu par de chaves; a etapa de registro, onde os nós cliente compartilham suas chaves-pública com o servidor para serem registrados nele; e a etapa de autenticação, onde o nó cliente assina a mensagem e compartilha ela com sua assinatura para o servidor, o qual verifica a assinatura, aceitando-a ou não. Em relação aos testes de performance, comparou-se o tempo das operações do esquema proposto usando cBLAKE2b com o esquema usando BLAKE2b, e o modelo proposto mostrou-se mais rápido em todas as operações, sendo elas: hashing dos dados, geração das chaves, geração de assinatura e verificação da assinatura.

3.2.3 ECDSA on Things: IoT Integrity Protection in Practise

Bauer *et al.* (2016) documentam as experiências e lições aprendidas durante o desenvolvimento e teste de uma aplicação de assinatura digital do framework RERUM, o qual eles desenvolveram anteriormente e tem como objetivo prover proteção de integridade de ponta-a-ponta para dados de dispositivos IoT. Segundo eles, a integridade dos dados poderia ser alcançada simplesmente utilizando MACs simétricos, porém, somente com assinaturas digitais eles conseguem garantir a autenticidade dos dados e de sua origem. Neste trabalho, eles têm como foco analisar o impacto prático que as assinaturas têm nos dispositivos IoT, em termos de overhead de tempo, duração de bateria e de seu uso em uma conexão de rede instável. Para o dispositivo dos experimentos, inicialmente eles iriam usar a Z1 Platform, mas depois trocaram para a Zolertia RE-Mote, pois o Z1 era muito limitado em termos de memória RAM (8KB) e flash (60KB) para conseguir fornecer uma segurança adequada, isto é, com curvas de tamanho maior ou igual a 192 bits. O dispositivo possui uma interface REST para acesso de seus recursos, e retorna seus dados em JSON, no formato padrão ou no formato JSS (*JSON Sensor Signatures*), proposto por um dos autores, que contém metadados adicionais sobre a assinatura integrada e o algoritmo utilizado. Em relação a implementação da assinatura, eles utilizam SHA-256 para obter o hash da mensagem e o algoritmo ECDSA da biblioteca MicroECC com a curva secp192r1 para gerar a assinatura. Após a assinatura ser gerada, ela e os dados do dispositivos são codificados no formato JSS e exposto pela interface REST CoAP. Quanto aos experimentos, os autores compararam o tempo de codificação, de hash, e de geração e verificação de assinaturas de uma implementação normal, com implementações com otimizações de assembly e com aceleração de hardware, esta última se mostrando superior às outras na maioria dos casos. Em relação aos experimentos com a duração da bateria dos dispositivos, não houve diferença notável entre um dispositivo que assinava e emitia mensagens, e um dispositivo que só emitia mensagens. Por fim, os autores mostram que a utilização do protocolo CoAP e UDP fornecem uma comunicação confiável e robusta em uma rede instável.

3.2.4 An Efficient Privacy Preserving Message Authentication Scheme for Internet-of-Things

Wei, Phuong e Yang (2020) revisam o esquema de autenticação SAMA (*Source Anonymous Message Authentication*) proposto na literatura, e propõem uma nova versão desse esquema corrigindo uma falha de segurança presente nele e implementando melhorias de versatilidade e eficiência. O trabalho foca na integridade, autenticidade e também na privacidade de origem na transmissão de dados entre dispositivos IoT. O SAMA é um esquema de assinaturas em anel derivado do esquema modificado de assinaturas ElGamal, e oferece autenticação de mensagem *hop-by-hop*, em que cada nó na rota consegue verificar a

integridade e a autenticidade da mensagem recebida, mas não consegue determinar de qual nó a mensagem originou. Embora privacidade de origem não seja algo que seja interessante para este trabalho, é importante notar que cada nó IoT da rede pode gerar e verificar assinaturas digitais. No novo esquema SAMA proposto, a melhoria na versatilidade se dá pela possibilidade dos nós no anel poderem utilizar algoritmos de assinatura diferentes um dos outros. No caso, os autores assumem que os nós utilizarão RSA ou assinaturas baseadas em logaritmo discreto (DL), como DSA, Schnorr ou ElGamal, por serem os algoritmos mais utilizados na atualidade. Em relação a melhoria de eficiência, eles aplicam o paradigma offline/online, em que o dispositivo pode realizar operações custosas de chave pública offline (quando não está sendo utilizado), e só realiza computações online quando a mensagem está pronta para ser enviada. O experimento relevante para este trabalho foi realizado em um Raspberry Pi 3, onde para um anel de 100 nós RSA e 100 nós DL, o tempo de computação tanto para a geração como para a verificação de assinaturas foi por volta de 1 segundo.

3.2.5 Analysis of ECDSA's Computational Impact on IoT Network Performance

Clark e Ali (2023) desenvolveram um sistema de autenticação simples para IoT utilizando assinaturas digitais, com o objetivo de minimizar o impacto do *overhead* computacional gerado pela verificação de assinaturas digitais na rede. O algoritmo de assinatura digital utilizado é o ECDSA com a curva secp160r1, e os dispositivos utilizados para experimentação são todos ESP32. Para a comunicação entre os dispositivos, eles utilizam o protocolo de comunicação sem fio ESP-NOW. O sistema utilizado para os experimentos é composto de um dispositivo receptor e dois ou mais emissores, onde os emissores enviam pacotes de dados assinados e o receptor, que contém as chaves-públicas de todos os dispositivos participantes, verifica a assinatura dos pacotes de dados recebidos. É válido ressaltar que embora privacidade não seja o foco do trabalho, no protocolo ESP-NOW, os dados enviados são protegidos com criptografia simétrica. No experimento realizado, cada emissor envia 10.000 pacotes de dados para o receptor, com intervalos variando de 5ms a 125ms entre eles. A performance do sistema é avaliada em termos de tempo de envio de pacotes e de perda de pacotes, com estas métricas sendo aplicadas também, para comparação, em um cenário onde não há assinatura dos pacotes de dados. Os resultados dos experimentos demonstram que o overhead de tempo para o envio de pacotes gerados pelas assinaturas chega a ser até cerca de 300ms, e que embora a porcentagem de pacotes perdidos é menor na maiorias do casos para pacotes não assinados, a diferença não é tão significativa, visto que o que mais influencia nesse fator é a duração do intervalo entre envios.

3.2.6 JSON Sensor Signatures (JSS): End-to-End Integrity Protection from Constrained Device to IoT Application

Pöhls (2015) busca proteger a integridade das informações na cadeia de processamento de dados IoT de ponta-a-ponta. Para isso, eles propõem um formato de JSON chamado JSS (JSON Sensor Signature). Baseado nos formatos JWS (JSON Web Signatures), COSE (CORB Object Signing and Encryption) e JCT (JSON Clear Text Signatures), o JSS mantém a simplicidade do JSON, enquanto integra uma assinatura digital e seus meta-dados como elementos JSON dentro do próprio objeto JSON assinado. Para garantir que o JSON gere sempre a mesma assinatura mesmo com seus elementos trocados de ordem por intermediários, por exemplo, o JSS prevê também a realização da canonicalização do objeto JSON antes de ser assinado e antes de ser verificado. Em relação aos experimentos, o algoritmo de assinatura digital escolhido foi o ECDSA com a curva secp160r1, implementado por meio da biblioteca ECClight para o Contiki OS, e o dispositivo IoT escolhido foi o Zolertia Z1. No experimento, um dispositivo Z1 assina os dados, e envia o JSS via UDP para outro dispositivo Z1 que realiza a verificação da assinatura. O tempo de geração do JSS completo, numa média de 200 execuções, foi de aproximadamente 2123ms, e a geração somente da assinatura foi de aproximadamente 1016ms.

3.2.7 Towards quantifying the cost of a secure IoT: Overhead and energy consumption of ECC signatures on an ARM-based device

Mössinger *et al.* (2016) avaliam o overhead para assinar e verificar uma mensagem com assinaturas digitais baseadas em curvas elípticas, em termos de tempo de execução, tamanho de código, consumo de energia e comunicação, em dispositivos ARM com restrições de recursos. Para os experimentos, utilizou-se o dispositivo Zolertia Re-Mote, que possui um processador ARM, com o sistema operacional Contiki. Foram analisadas 3 bibliotecas diferentes que implementavam assinaturas digitais: TweetNaCl, Piñol e MicroECC. As bibliotecas foram portadas para rodar no Contiki, e seus códigos foram ajustados quando necessário. A biblioteca TweetNaCl implementava a curva Ed25519, enquanto que a Piñol implementava a curva secp256r1, e a MicroECC implementava as curvas secp160r1, secp192r1, secp224r1, e secp256r1, e possibilitava otimizações de assembly. As curvas implementadas pela MicroECC demonstraram maior velocidade e menor consumo de energia do que as outras bibliotecas, e os autores escolheram a curva secp192r1 para ser utilizada no restante dos experimentos. O protocolo usado para o envio das mensagens foi o CoAP, e o formato utilizado foi o JSS. Por fim, em relação aos resultados, a geração de assinatura gerou um overhead de aproximadamente 200ms, e o tempo total de transmissão da mensagem foi de 361ms, com um tamanho de código de cerca de 86kB.

3.2.8 Outros trabalhos

Durante a revisão da literatura, encontrou-se muitos trabalhos que embora possuíssem objetivos ou certas características em comum com o presente trabalho, eles se divergiam significativamente em relação ao método proposto para alcançar tais objetivos. Por exemplo, há muitos trabalhos que buscam obter autenticação mútua entre um dispositivo IoT e outro dispositivo ou entidade, não objetivando autenticação de cada mensagem transmitida, mas do estabelecimento de uma sessão autêntica e confiável com outro dispositivo em que as mensagens possam ser transmitidas de forma segura (HOMADI; DAWOOD, 2023; LI, B. *et al.*, 2022; SUN *et al.*, 2021; SACHAN; KUMAR; ADWITEEYA, 2019; PANDA; CHATTOPADHYAY, 2020; ZHAO *et al.*, 2011). Oposto a isso, há também propostas de MACs para garantir a integridade e autenticidade das mensagens transmitidas nos trabalhos de Jian Li *et al.* (2020), He Li *et al.* (2021), Younis, Farrag e Althouse (2011), Yan *et al.* (2022) e Ullah, Meratnia e Havinga (2020), mas que não oferecem o não-repúdio de assinaturas digitais.

Embora o trabalho atual foque no uso mais simples de assinaturas digitais, percebeu-se que há muitos artigos em que tipos diferentes de assinatura e certificação digital são propostos para IoT. Nos trabalhos de Malik, Dutta, Granjal *et al.* (2023), Sciancalepore *et al.* (2016), Porambage *et al.* (2014) e Ha, Nguyen e Zao (2016), os autores buscam autenticação em IoT por meio de certificados implícitos usando ECQV (*Elliptic Curve Qu-Vanstone*), onde a chave pública é reconstruída a partir do certificado, ocasionando certificados menores e operações mais rápidas. Também há trabalhos que fazem uso de assinaturas agregadas, como em Hou *et al.* (2021), Verma, Singh *et al.* (2019), Zhu *et al.* (2021), Verma, Kumar *et al.* (2021) e Kaâniche, Jung e Gehani (2018), em que várias assinaturas são agregadas em uma só, reduzindo o overhead de comunicação e tornando a verificação mais eficiente, pois somente a assinatura agregada necessita ser transmitida e verificada. Por fim, também há uma proposta de *redactable signature* feita por Liu *et al.* (2022).

O único trabalho relacionado que chegou a ter mais intersecção com o presente trabalho, foi o de Suárez-Albela, Fraga-Lamas e Fernández-Caramés (2018), em que os autores também realizam testes com os algoritmos de chave pública ECDSA e RSA no ESP32, também utilizando a aceleração de hardware do dispositivo. Entretanto, o objetivo desses autores era saber o impacto na performance e no consumo de energia ao se utilizar TLS com esses algoritmos, o que implica que eles também realizam troca de chaves e cifragem da mensagem além da assinatura, não focando somente na garantia de autenticidade e integridade. Outra diferença substancial do trabalho destes autores, é que eles não realizam a medição do tempo de cada operação relacionada ao processo de assinar e verificar uma mensagem, eles medem a taxa de transferência de dados entre dois dispositivos, em requisições por segundo. Além do mais, eles utilizam somente a biblioteca MbedTLS em seus experimentos, o que implica em duas coisas: (1) eles tem acesso somente

aos algoritmos disponíveis dessa biblioteca, o que não inclui algoritmos EdDSA; e (2), os resultados obtidos por eles ficam restritos ao desempenho desta única biblioteca. Por fim, eles não conseguiram utilizar a aceleração de hardware com o algoritmo RSA com chaves de 4096 bits.

4 DESENVOLVIMENTO

Neste trabalho, foi desenvolvida uma prova de conceito de certificação digital no dispositivo IoT ESP-WROOM-32, ou ESP32, em que utilizou-se algoritmos de assinaturas digitais aceitos pelo ITI (ICP-BRASIL, 2022). Esta prova de conceito se materializou na forma de uma biblioteca chamada CryptoAPI, de forma que ela possa servir como uma interface para as operações criptográficas implementadas por outras bibliotecas.

Realizou-se no dispositivo: a geração e o armazenamento de um par de chaves criptográficas; a geração e o armazenamento de uma assinatura digital; e, por fim, a verificação da assinatura. Embora não faça muito sentido o próprio dispositivo realizar a verificação de uma assinatura que ele acabou de gerar, o propósito aqui é demonstrar como seria o tempo de execução e o consumo de memória desta operação caso ela fosse realizada em outro dispositivo IoT, que iria receber a assinatura, a mensagem e a chave pública do dispositivo que as gerou.

A realização com sucesso de todas estas etapas demonstrará que qualquer operação criptográfica comum relacionada à assinatura digital pode ser realizada no ESP32. A diferença desta solução para as existentes na literatura, é o foco na garantia de integridade e autenticidade dos dados transmitidos pelo dispositivo, sem a necessidade de obter confidencialidade ou de autenticação mútua. Conseguir assegurar essas garantias é essencial para que o dispositivo possa ser aplicado em contextos onde os dados são públicos, como em semáforos de trânsito ou em balanças por exemplo, de modo a evitar que esses dados sejam adulterados por atores maliciosos. Ainda que ocorram adulterações, elas devem ser facilmente identificadas.

Depois, foram realizados experimentos com a biblioteca criada para medir o desempenho em termos de velocidade e consumo de memória dos algoritmos criptográficos utilizados. Os resultados foram então comparados e discutidos. Finalmente, este trabalho termina com as conclusões sobre os experimentos e com dois artefatos gerados, sendo um deles o próprio trabalho de conclusão de curso, e o outro o projeto da CryptoAPI.

4.1 ESCOLHA DAS BIBLIOTECAS

Para a escolha das bibliotecas criptográficas a serem utilizadas, levou-se em consideração três principais fatores: (1) se a biblioteca é confiável; (2) se ela implementa algum algoritmo de assinatura digital; e (3) se é possível utilizá-la no ESP32 sem precisar portá-la. Considerou-se que bibliotecas confiáveis devem possuir um número próximo ou superior à 1000 estrelas em sua página do GitHub (se houver) ou ter sido utilizada por algum dos artigos revisados. Após diversas pesquisas, as bibliotecas encontradas e consideradas para a análise foram: mbedtls, wolfssl, micro-ecc, arduinolibs, psa-crypto-arduino, TweetNaCl, arduino-crypto e tinyECC. Essas bibliotecas foram avaliadas conforme os critérios de seleção previamente elencados, e as que cumpriram todos os critérios foram escolhidas,

conforme mostra a Tabela 3.

Tabela 2 – Análise de bibliotecas criptográficas

Nome	**Estrelas**	**Assinatura digital**	**Porte nativo**	**Escolhida?**
arduinolibs	436	Sim	Sim	Não
arduino-crypto	140	Não	Sim	Não
mbedtls	**5 253**	**Sim**	**Sim**	**Sim**
micro-ecc	**1 249**	**Sim**	**Sim**	**Sim**
psa-crypto-arduino	3	Sim	Sim	Não
tinyECC	22	Sim	Sim	Não
TweetNaCl	N/A	Sim	Não	Não
wolfssl	**2 314**	**Sim**	**Sim**	**Sim**

Como pode ser observado, as únicas bibliotecas que cumprem os três requisitos elencados são a mbedtls, a wolfssl e a micro-ecc. Portanto, estas foram as três bibliotecas escolhidas para serem utilizadas na CryptoAPI. Estas bibliotecas estão descritas a seguir.

- **mbedtls:** a MbedTLS é uma biblioteca open-source que implementa primitivas criptográficas, manipulação de certificados X.509 e os protocolos SSL/TLS e DTLS. Os frameworks para desenvolvimento em ESP32 já incluem essa biblioteca por padrão. (MBEDTLS, 2024; ESPRESSIF, 2024c).
- **wolfssl:** a wolfSSL é uma biblioteca open-source de SSL/TLS leve escrita em ANSI C, voltada para ambientes embarcados, RTOS e com recursos limitados. Uma versão do módulo criptográfico dessa biblioteca possui a validação FIPS 140-3 da NIST (WOLFSSL, 2024b,a).
- **micro-ecc:** a micro-ecc é uma biblioteca open-source que possui implementações pequenas e rápidas de ECDH e ECDSA, para serem utilizadas em processadores 8-bit, 32-bit e 64-bit (MACKAY, 2024).

4.2 ESCOLHA DOS ALGORITMOS

Nesta seção serão apresentados os algoritmos de assinatura digital e de hash criptográfico utilizados, bem como o porquê da escolha de cada algoritmo.

4.2.1 Algoritmos de assinatura digital

Os algoritmos de assinatura digital a serem utilizados foram escolhidos com base na versão 5.0 do documento "Padrões e algoritmos criptográficos da ICP-Brasil" (ICP-BRASIL, 2022), em que são listados os algoritmos de geração de chaves assimétricas aceitos pelo ITI. A Figura 5 mostra quais são esses algoritmos. Embora o ideal seria conseguir testar todos eles, ficamos limitados pelas implementações disponíveis nas bibliotecas criptográficas escolhidas. Portanto, os algoritmos de assinatura digital que irão integrar a CryptoAPI são:

- ECDSA com as curvas brainpool:
  - brainpool256r1, com chaves de tamanho 256 bits;
  - brainpool521r1, com chaves de tamanho 512 bits.
- ECDSA com as curvas NIST:
  - secp256r1 (ou P-256), com chaves de tamanho 256 bits;
  - secp521r1 (ou P-521), com chaves de tamanho 521 bits.
- RSA com chaves de tamanho 2048 e 4096 bits;
- Ed25519 com chaves de tamanho 256 bits;
- Ed448 com chaves de tamanho 448 bits.

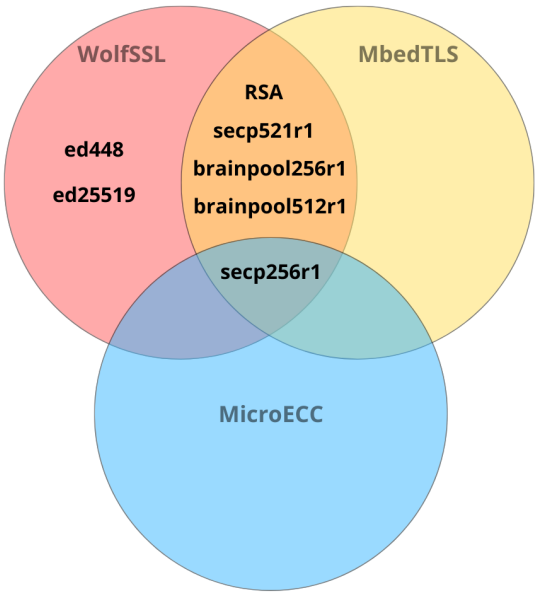
Figura 5 – Algoritmos de assinatura digital aceitos pelo ITI.

Geração de Chaves Assimétricas de Usuário Final	
Normativo ICP-Brasil	DOC-ICP-04 - item 6.1.5.2
Algoritmo	RSA ou ECC-Brainpool (conforme RFC 5639) ou Curve25519 (Conforme RFC 7748) ou Ed25519 (PureEdDSA e HashEdDSA, conforme RFC 8032) ou Ed448-Goldilocks (PureEdDSA e HashEdDSA, conforme RFC 8032) ou E-521 (Conforme parâmetros da curva estabelecidos neste DOC-ICP-01.01, PureEdDSA e HashEdDSA, conforme RFC 8032).
Tamanho de chave A1, A2, A3, A CF-e-SAT, S1, S2, S3, T3, OM-BR	RSA 2048 ou brainpoolP256r1 ou Curve25519 (256 bits) ou Ed25519 (256 bits) ou Ed448 (448 bits) ou E-521 (521 bits)
Tamanho da chave A4, S4, T4	RSA 2048 ou RSA 4096 ou brainpoolP512r1 ou Curve25519 (256 bits) ou Ed25519 (256 bits) ou Ed448 (448 bits) ou E-521 (521 bits)

Fonte: Padrões e Algoritmos Criptográficos da ICP-BRASIL (ICP-BRASIL, 2022)

É importante notar que embora o ITI não aceite as curvas da NIST, como elas estão disponíveis em todas as bibliotecas criptográficas escolhidas, elas ainda serão integradas à CryptoAPI e farão parte dos testes, pois é de grande valor conseguir comparar seu desempenho com a dos algoritmos aceitos. A Figura 6 demonstra de forma esquematizada todos os algoritmos que serão utilizados, bem como quais bibliotecas os implementam.

Figura 6 – Algoritmos de assinatura digital que serão utilizados.



Fonte: Autoria própria.

4.2.2 Algoritmos de hash

Os algoritmos de hash utilizados para calcular o resumo das mensagens também serão escolhidos com base nos algoritmos aceitos pelo ITI. Como demonstrado pela Figura 7, os algoritmos aceitos são: SHA-1; SHA-256; SHA-512; e SHAKE-512. Entretanto, como o SHA-1 chegou ao fim de sua vida útil segundo a NIST (NIST, 2022), ele não será utilizado, sendo substituído pelo mais moderno e mais seguro SHA-3.

A Figura 8 evidencia quais bibliotecas implementam os algoritmos de hashes que serão utilizados. Como se pode notar, a biblioteca micro-ecc não provê nenhum algoritmo de hash e, portanto, para gerar o resumo das mensagens nos testes com micro-ecc, serão utilizados os algoritmos de hash da biblioteca MbedTLS.

4.3 IMPLEMENTAÇÃO

Nesta seção são apresentadas todas as informações pertinentes à implementação da CryptoAPI. Todo o código desenvolvido do projeto está disponível no repositório do autor no GitHub (LOLI, 2024), com o nome de esp32-crypto-api.

4.3.1 Ambiente de desenvolvimento

Para a implementação da CryptoAPI, inicialmente tentou-se utilizar o ambiente de desenvolvimento Arduino IDE, que possui suporte oficial para desenvolvimento em ESP32.

Figura 7 – Algoritmos de hash criptográfico aceitos pelo ITI.

Assinaturas Digitais ICP-Brasil	
Normativo ICP-Brasil	DOC-ICP-15, item 6.1
Função resumo	SHA - 1 SHA - 256 SHA – 512 SHAKE - 256
Suíte de Assinatura	sha256WithRSAEncryption sha256WithECDSAEncryption sha512WithRSAEncryption sha512WithECDSAEncryption

Fonte: Padrões e Algoritmos Criptográficos da ICP-BRASIL (ICP-BRASIL, 2022)

Contudo, embora simples de ser utilizada, a IDE possui algumas limitações relacionadas ao nível de controle sobre a execução do código na placa, o que impossibilitava a utilização da aceleração de hardware para operações criptográficas do ESP32 com os algoritmos da biblioteca wolfSSL. Devido a este impeditivo, migrou-se o código para o framework ESP-IDF (Espressif IoT Development Framework). O ESP-IDF é um framework próprio da Espressif (fabricante dos dispositivos ESP) e o recomendado para o desenvolvimento em ESP32 (ESPRESSIF, 2023a). Ele permite alto nível de controle sobre a escrita e execução dos programas nas placas ESP. Além disso, o ESP-IDF possui uma extensão oficial para o editor de texto VSCode, que permite que o framework seja utilizado inteiramente dentro do editor, e foi este o caminho escolhido para o projeto.

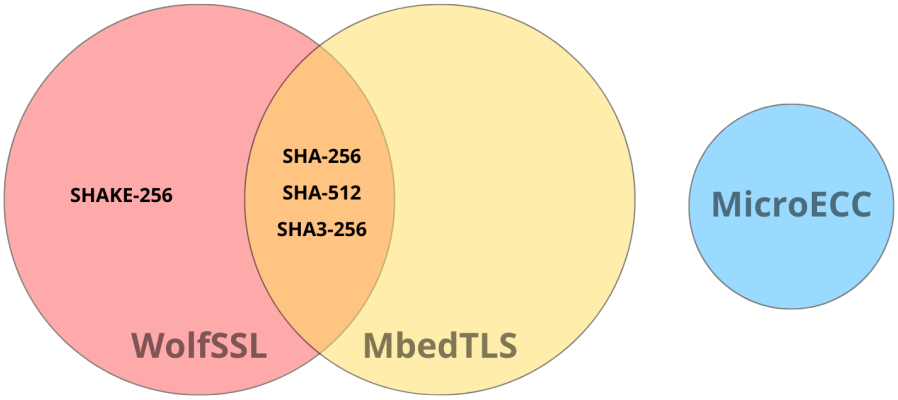
4.3.2 Configurações do projeto

Nesta seção são detalhadas as configurações que foram necessárias realizar no projeto.

4.3.2.1 Configuração para utilizar C++

Objetivou-se utilizar C++ ao invés de C devido à sua natureza orientada à objetos, que facilita a codificação e a organização de um projeto complexo. Para que C++ pudesse ser utilizado no projeto, foi necessário adicionar o componente *esp-idf-cxx* da Espressif,

Figura 8 – Algoritmos de hash criptográfico que serão utilizados.



Fonte: Autoria própria.

que está disponível no *ESP-IDF Component Registry* (ESPRESSIF, 2023b).

4.3.2.2 Configuração para poder utilizar o sistema de arquivos do ESP32

Para que fosse possível salvar arquivos na memoria Flash do dispositivo, foi necessário adicionar o component *littlefs*, que está disponível no *ESP-IDF Component Registry* (JOLTWALLET, 2024). Após isso, precisou-se configurar as partições, e para isso criou-se um arquivo chamado *partitions.csv* na raiz do projeto, contendo as configurações necessárias.

4.3.2.3 Configuração para a biblioteca MbedTLS

A biblioteca mbedtls já vem incluída como um componente padrão do framework ESP-IDF, e portanto não é necessário realizar nenhuma configuração para começar a utilizá-la.

4.3.2.4 Configuração para a biblioteca micro-ecc

A biblioteca micro-ecc precisou ser incluída como um componente no projeto manualmente, mas não foi necessário nenhuma configuração adicional.

4.3.2.5 Configuração para a biblioteca wolfSSL

A biblioteca wolfSSL precisou ser incluída como um componente no projeto manualmente, e foi necessário realizar algumas configurações nela, para que todos os algoritmos de criptografia necessários estivessem habilitados. Essas configurações são feitas adicionando ou removendo a definição de macros específicas da biblioteca wolfSSL no arquivo *user_settings.h*. É importante notar, ainda, que este arquivo já vem com uma configuração inicial; isto é, já possui uma série de macros definidas. A seguir são listadas as macros que precisaram ser definidas, juntamente de uma breve descrição de seu propósito:

- **WOLFSSL_SHAKE256** - para habilitar o algoritmo de hash criptográfico SHAKE-256;
- **WOLFSSL_CUSTOM_CURVES** - para habilitar curvas não padronizadas pela NIST, como as curvas Brainpool;
- **WOLFSSL_KEY_GEN** - para habilitar a geração de chaves com RSA;
- **ECC256** - para habilitar a curva secp256r1;
- **HAVE_ECC_BRAINPOOL** - para habilitar as curvas Brainpool;
- **HAVE_ED448** - para habilitar o algoritmo de assinatura digital Ed448.

Além disso, foi necessário alterar o valor da macro FP_MAX_BITS de 4096 para 8192 no arquivo *tfm.h*, para possibilitar a geração de chaves RSA de 4096 bits de tamanho. Isso é necessário pois a FP_MAX_BITS necessita ter o dobro do tamanho da maior chave utilizada, a qual, no caso deste trabalho, é de 4096 bits. Por fim, a seguir são listadas algumas macros relevantes que já vieram definidas por padrão, juntamente com uma breve descrição de seu propósito:

- **USE_FAST_MATH** - faz com que a biblioteca FastMath seja utilizada ao invés da biblioteca de matemática padrão. A FastMath utiliza Assembly se possível, aumentando consideravelmente a velocidade das operações de chaves assimétricas (WOLFSSL, 2023). Além disso, para suas operações, toda a memória é alocada na stack;
- **WOLFSSL_ESPIDF** - para compilar a biblioteca para ESP-IDF;
- **WOLFSSL_ESP32** - para habilitar configurações específicas da placa, como a aceleração de hardware;
- **RSA_LOW_MEM** - utiliza metade da memória necessária para o RSA, mas o deixa duas vezes mais lento;

- **WOLFSSL_SMALL_STACK** - aumenta o uso da memória *heap*, mas pode resultar em um desempenho mais lento;
- **WOLFSSL_SHA512** - habilita o algoritmo de hash criptográfico SHA-512;
- **WOLFSSL_SHA3** - habilita o algoritmo de hash criptográfico SHA3;
- **HAVE_ED25519** - habilita o algoritmo de assinatura digital Ed25519;
- **HAVE_ECC** - habilita o uso de algoritmos de criptografia de curvas elípticas.

Nota-se que há duas definições que já vem ativas por padrão que podem afetar consideravelmente o desempenho dos algoritmos: RSA_LOW_MEM e WOLFSSL_SMALL_STACK. A implicação delas no desempenho será tratada com a devida atenção no Capítulo 5.

4.3.3 Estrutura do projeto

O projeto CryptoAPI é constituído primariamente por seis classes, sendo elas: *CryptoAPI*, *CryptoApiCommons*, *MbedtlsModule*, *MicroeccModule*, *WolfsslModule*, e *ICryptoModule*. A ideia por trás dessa estrutura foi abstrair os detalhes das implementações dos algoritmos de criptografia por trás de "módulos" criptográficos. Esses módulos nada mais são do que classes, e levam cada um o nome da biblioteca que ele abstrai. Neste caso, o módulo *MbedtlsModule* abstrai as operações da biblioteca MbedTLS; o módulo *MicroeccModule* abstrai as operações da biblioteca micro-ecc; e, por fim, o módulo *WolfsslModule* abstrai as operações da biblioteca WolfSSL.

Para tornar mais fácil a adição ou remoção de módulos, resolveu-se padronizar os métodos essenciais para operações criptográficas, como geração de chaves, geração de assinaturas e verificação de assinaturas, por exemplo, e assim criou-se uma interface chamada de *ICryptoModule*, para reunir esses métodos essenciais. Assim, cada módulo criptográfico implementa esta interface. Isso facilita a posterior adição de outros módulos, pois fornece um *template* dos métodos que ele deve possuir para ser compatível com a CryptoAPI.

Ainda, para tornar mais fácil a troca de módulos ao testar o projeto, abstraiu-se os módulos em uma classe, chamada de *CryptoAPI*. A função desta classe nada mais é do que rotear a chamada de um método seu para o módulo criptográfico que está sendo utilizado no momento. Por isso, esta classe deve possuir os mesmos métodos essenciais dos outros módulos, e assim ela acaba implementando a interface *ICryptoModule*, o que torna a própria CryptoAPI um módulo criptográfico.

Por fim, a classe *CryptoApiCommons* nada mais é do que uma classe de utilidade, para armazenar dados e funcionalidades em comum, utilizadas por mais de um módulo.

Para exemplificar melhor como a CryptoAPI é estruturada, a Figura 9 apresenta o diagrama de classes resumido do projeto, sem os parâmetros dos métodos. O diagrama de classes completo está em anexo a este trabalho.

A seguir serão explicados os métodos essenciais supracitados. Como os principais métodos dos módulos são os mesmos, diferenciando-se pela implementação específica de cada módulo, serão utilizados somente os métodos da interface *ICryptoModule* para explicá-los.

- **Método *init*:** Utilizado para inicializar classes e outros objetos necessários para as funções de cada módulo. Ao chamá-lo, devem ser passados como argumentos: o algoritmo de assinatura digital a ser utilizado; o algoritmo de hash criptográfico a ser utilizado; e o tamanho de hash para o algoritmo SHAKE-256, caso esse tenha sido o algoritmo de hash escolhido. A classe *CryptoAPI* possui uma sobrecarga deste método, onde é passado adicionalmente qual biblioteca criptográfica deve ser utilizada, servindo como ponto de entrada para as outras funcionalidades da biblioteca e devendo ser sempre o primeiro método dela a ser invocado.

- **Método *get_signature_size*:** Utilizado para recuperar o tamanho da assinatura após esta ter sido gerada.

- **Método *gen_rsa_keys*:** Utilizado para gerar um par de chaves RSA. Recebe como argumentos o tamanho da chave e o valor do expoente. Foi necessário uma função separada justamente por causa desses parâmetros. Na biblioteca micro-ecc, como ela não possui o algoritmo RSA, a implementação deste método é vazia.

- **Método *gen_keys*:** Utilizado para gerar um par de chaves utilizando o algoritmo que foi escolhido no método *init*. Diferentemente do RSA, como os outros algoritmos têm um tamanho fixo para suas chaves, os módulos se encarregam internamente de conseguir esses valores para serem usados nos métodos de geração de chave de cada biblioteca.

- **Método *sign*:** Utilizado para assinar uma mensagem. Recebe como argumentos a mensagem a ser assinada, o tamanho desta mensagem, um buffer para guardar a assinatura após ser gerada, e o tamanho desta assinatura. É importante salientar que a mensagem passada como argumento não deve ser um hash, pois cada módulo realiza o hash da mensagem internamente, com o algoritmo de hash escolhido no método *init*. Logo, é interessante ressaltar também que os métodos de gerar assinaturas das bibliotecas utilizadas não realizam o hash internamente, sendo necessário realizar o hash antes, ação que é executada neste método.

- **Método *verify*:** Utilizado para verificar se uma assinatura digital é válida ou não. Recebe como argumento a mensagem a ser verificada, o tamanho desta mensagem,

a assinatura a ser verificada, e o tamanho desta assinatura. Uma mensagem será impressa no monitor serial com o resultado da validade da assinatura. Assim como no método *sign*, o hash também é realizado internamente.

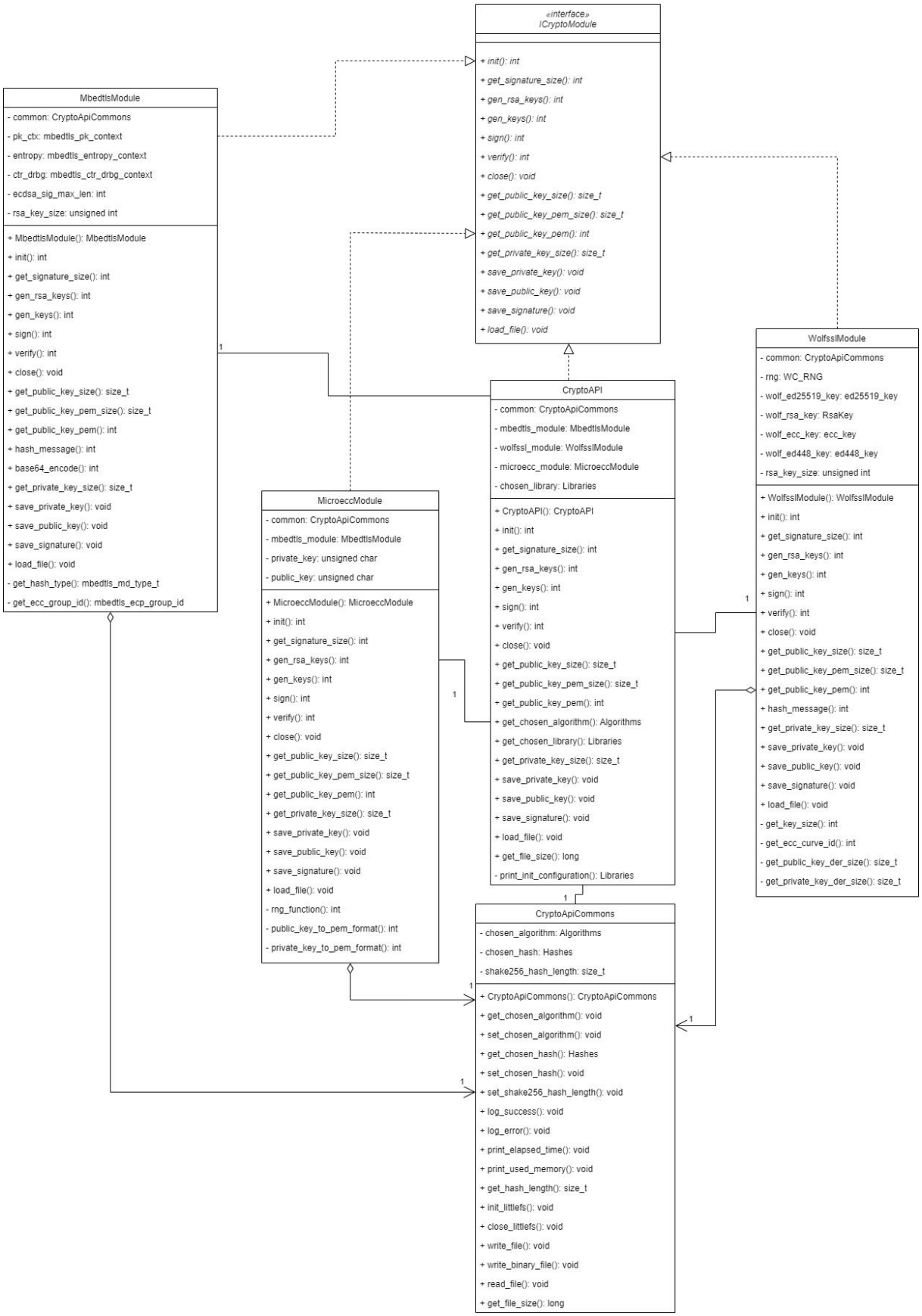
- **Método *close*:** Utilizado para liberar os recursos que foram alocados durante a utilização do módulo.
- **Método *get_public_key_size*:** Utilizado para recuperar o tamanho da chave pública gerada.
- **Método *get_public_key_pem_size*:** Utilizado para recuperar o tamanho da chave pública gerada no formato PEM.
- **Método *get_public_key_pem*:** Utilizado para recuperar a chave pública gerada no formato PEM. Recebe como argumento um buffer para guardar a chave pública após ela ser convertida para o formato PEM.
- **Método *get_private_key_size*:** Utilizado para recuperar o tamanho da chave privada gerada.
- **Método *save_private_key*:** Utilizado para salvar a chave privada, no formato PEM. Recebe como argumento o nome do arquivo em que a chave privada será salva (deve terminar com .pem no final), o buffer contendo a chave privada, e o tamanho da chave privada.
- **Método *save_public_key*:** Utilizado para salvar a chave pública, no formato PEM. Recebe como argumento o nome do arquivo em que a chave pública será salva (deve terminar com .pem no final), o buffer contendo a chave pública, e o tamanho da chave pública.
- **Método *save_signature*:** Utilizado para salvar a assinatura, no formato binário. Recebe como argumento o nome do arquivo em que a assinatura será salva (deve terminar com .bin no final), o buffer contendo a assinatura, e o tamanho da assinatura.
- **Método *load_file*:** Utilizado para recuperar da memória Flash um arquivo. Recebe como argumento o nome do arquivo que terá seu conteúdo lido, um buffer para guardar os dados lidos do arquivo, e o tamanho do buffer.

4.3.4 Memória ocupada pelo projeto

Ao compilar o projeto, o ESP-IDF exibe algumas informações no terminal integrado do VSCode relacionadas à memória ocupada pelos binários do projeto, como mostra a Figura 10. Pela imagem, pode-se observar que dentre os arquivos binários, todo o código

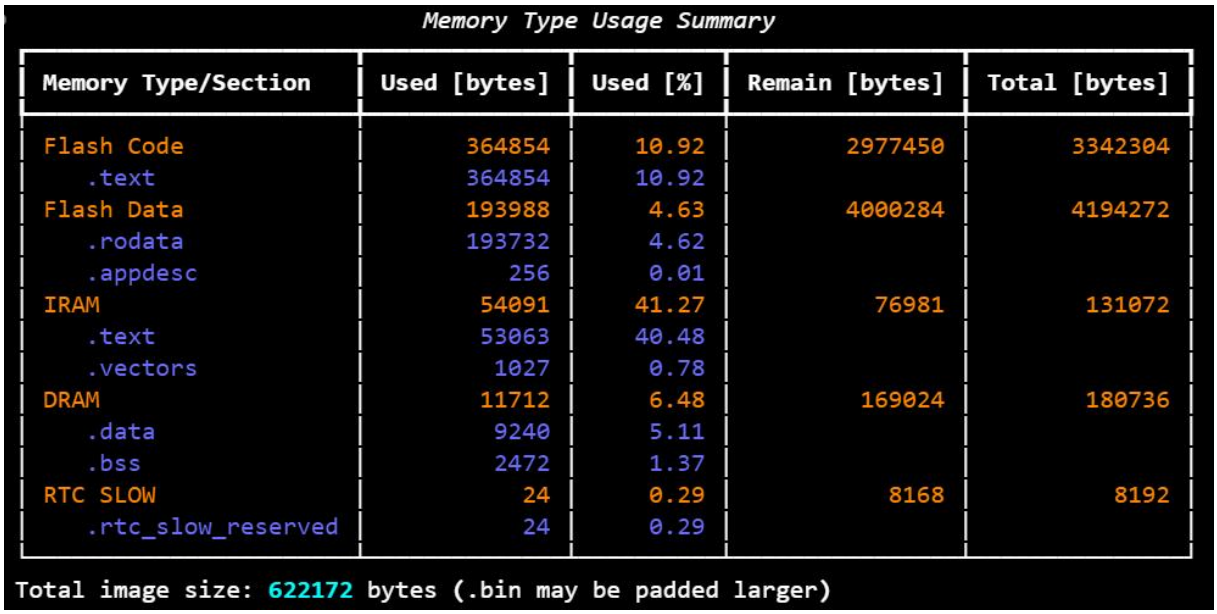
ocupa apenas 10% da memória flash reservada para ele, correspondendo a aproximadamente 357kB. Também é possível ver que há 4MB de memória flash livres para uso, o que significa que é possível realizar o armazenamento de muitas chaves criptográficas e assinaturas digitais.

Figura 9 – Diagrama de classes da CryptoAPI.



Fonte: Autoria própria.

Figura 10 – Resumo do uso de memória pelo projeto CryptoAPI.



Memory Type Usage Summary

Memory Type/Section	Used [bytes]	Used [%]	Remain [bytes]	Total [bytes]
Flash Code	364854	10.92	2977450	3342304
.text	364854	10.92		
Flash Data	193988	4.63	4000284	4194272
.rodata	193732	4.62		
.appdesc	256	0.01		
IRAM	54091	41.27	76981	131072
.text	53063	40.48		
.vectors	1027	0.78		
DRAM	11712	6.48	169024	180736
.data	9240	5.11		
.bss	2472	1.37		
RTC SLOW	24	0.29	8168	8192
.rtc_slow_reserved	24	0.29		

Total image size: 622172 bytes (.bin may be padded larger)

Fonte: Autoria própria.

5 TESTES E RESULTADOS

Nesta seção serão descritas as condições dos testes realizados com a CryptoAPI e como eles ocorreram. Após isso, serão apresentados e discutidos os resultados desses testes.

5.1 TESTES

5.1.1 Dispositivo utilizado

O dispositivo utilizado para os testes foi o ESP32, modelo ESP-WROOM-32, o mesmo dispositivo usado para o desenvolvimento do código. Ele utiliza o processador *Xtensa dual-core 32-bit LX6*, que apresenta um clock máximo de 240MHz. O aparelho possui 4MB de memória flash, 448KB de memória ROM e 512KB de memória RAM (ESPRESSIF, 2024b). Embora não esteja sendo utilizado neste projeto, ele também permite comunicação Bluetooth e WiFi.

Além disso, o ESP32 apresenta aceleração de hardware para operações de certos algoritmos criptográficos. Esses algoritmos são o AES (AES-128, AES-192 e AES-256), SHA (SHA-1, SHA256, SHA384 e SHA512) e RSA (ESPRESSIF, 2024a). Neste projeto, a aceleração de hardware está sendo utilizada para os algoritmos SHA256, SHA512 e RSA.

5.1.2 Metodologia de teste

Para testar os algoritmos de assinatura digital escolhidos, cada etapa do processo de geração de uma assinatura digital por um algoritmo foi medida em termos de tempo de execução (em milissegundos), consumo de memória (em bytes) e ciclos de clock (em ciclos). Essas etapas são: (1) geração do par de chaves; (2) geração da assinatura; e (3) verificação da assinatura. Como durante a implementação da CryptoAPI foi possível separar a geração de hash das operações de gerar e verificar assinaturas, o tempo de execução e consumo de memória dos algoritmos de hash foram medidos separadamente.

O cálculo do tempo de execução foi realizado com uma função própria do ESP32, chamada *esp_timer_get_time*. Para saber o tempo decorrido, subtraiu-se o tempo registrado após o término de certa operação com o tempo registrado no início da operação. Para calcular a memória consumida, utilizou-se três funções: (1) *heap_caps_monitor_local_minimum_free_size_start*, para começar a monitorar o valor do mínimo de bytes livres na memória heap a partir do momento em que esta função é chamada, ao invés de desde o início do programa; (2) *heap_caps_monitor_local_minimum_free_size_stop*, para parar o monitoramento; e (3) *esp_get_minimum_free_heap_size*, que retorna o menor valor de bytes livres na memória heap desde o início do programa. Essas funções permitiram calcular o pico de memória consumida por cada operação isoladamente, subtraindo o menor valor registrado após a

operação do menor valor registrado antes dela. Por fim, para calcular a quantidade de ciclos de clock gerada por cada operação, utilizou-se a função *esp_cpu_get_cycle_count*, que retorna o número de ciclos realizados pelo núcleo da CPU em uso desde o início do programa. Para determinar os ciclos utilizados por uma operação, subtraíram-se os valores registrados antes e após sua execução. A métrica de ciclos de clock foi considerada particularmente relevante, pois fornece uma medida independente de fatores externos como a precisão do temporizador ou oscilações na magnitude dos tempos medidos. Dessa forma, os ciclos de clock permitem uma avaliação mais direta do desempenho do hardware durante as operações realizadas.

Para a maioria dos algoritmos, foram realizadas 10 execuções, e calculou-se a média do tempo de execução, consumo de memória e ciclos de clock de cada etapa. Para o algoritmo RSA com chaves de tamanho 4096 bits das bibliotecas MbedTLS e wolfSSL, no entanto, como o tempo de execução possuía alto desvio padrão, realizou-se 20 execuções. O valor do expoente público do RSA utilizado foi de 65537, uma escolha padrão amplamente adotada em implementações de RSA por seu equilíbrio entre eficiência computacional e segurança, conforme descrito na RFC 8017 (MORIARTY *et al.*, 2016). A função de hash utilizada nas execuções dos algoritmos de assinatura digital foi a SHA512, exceto para o algoritmo Ed448, em que deve ser utilizado o algoritmo de hash SHAKE-256, segundo especifica a RFC 8032 (JOSEFSSON; LIUSVAARA, 2017), documento que define os algoritmos EdDSA. Em todos os testes, a mensagem utilizada para gerar o hash, que foi posteriormente assinado e utilizado na verificação, tem tamanho de 575 bytes e é a seguinte:

"Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum."

5.1.3 Considerações sobre o armazenamento de chaves e assinaturas

Implementou-se funcionalidade para salvar e carregar da memória chaves e assinaturas para todas as bibliotecas. Entretanto, para o RSA com chaves de 4096 bits da biblioteca wolfSSL, não foi possível realizar todas as operações. Embora o dispositivo consiga salvar as chaves públicas e privadas, após isso, ao tentar realizar a assinatura da mensagem, acontece um erro e o dispositivo reinicia. Não foi possível identificar com exatidão a origem do erro, mas ele está relacionado com a memória, possivelmente com a corrupção da memória *heap*. Com exceção deste algoritmo, foi possível realizar a operação

de salvar e carregar chaves e assinaturas com todos os outros algoritmos e bibliotecas, inclusive com o RSA com chaves de 4096 bits de tamanho da biblioteca MbedTLS.

5.1.4 Definições do wolfSSL

Conforme foi visto anteriormente, há duas macros no arquivo de configurações *user_settings.h* do wolfSSL que podem impactar negativamente na performance dos algoritmos: RSA_LOW_MEM e WOLFSSL_SMALL_STACK. A primeira impacta somente o RSA, enquanto a segunda impacta todos os algoritmos da biblioteca.

Para decidir se essas macros iriam permanecer durante os testes, realizou-se testes com o algoritmo RSA com chaves de 4096 bits nos mesmos moldes em que os testes finais foram realizados: 20 execuções, medindo o tempo de execução e o consumo de memória para as operações de gerar chaves, assinar uma mensagem e verificar a assinatura.

A Tabela 3 compara os resultados da média do tempo de execução das execuções para 4 situações: (1) ambas as macros definidas (padrão); (2) somente a macro RSA_LOW_MEM, ou A, definida; (3) somente a macro WOLFSSL_SMALL_STACK, ou B, definida; e, por fim, (4) nenhuma das macros definidas. Pelos resultados, observa-se que desabilitar a macro A impacta negativamente no tempo de assinatura, aumentando-o em aproximadamente 1 segundo. Além disso, embora habilitar ambas as macros resulte no melhor tempo de geração de chaves, esse resultado não é muito confiável, pois para essa operação os testes demonstraram um alto desvio padrão, que será abordado mais à frente. Por fim, os tempos para verificar a assinatura continuam praticamente os mesmos.

Tabela 3 – Média do tempo de execução para as operações de RSA com chaves de 4096 bits da biblioteca wolfSSL, para testar o impacto das macros RSA_LOW_MEM (A) e WOLFSSL_SMALL_STACK (B)

Algoritmo	**Gerar chaves**	**Assinar mensagem**	**Verificar assinatura**
RSA 4096 (macros A e B)	37.643,9	2.070,2	888,9
RSA 4096 (somente macro A)	46.261,6	2.070,8	889,0
RSA 4096 (somente macro B)	45.699,0	2.964,8	889,2
RSA 4096 (nenhuma das duas)	43.646,3	2.964,7	889,8

Fonte: Autoria própria.

Já a Tabela 4 compara os resultados da média de consumo de memória, para as mesmas situações que as descritas anteriormente. Nota-se que o consumo de memória permaneceu praticamente igual para a geração de chaves e verificação de assinaturas, o que é curioso, pois segundo a documentação, a macro RSA_LOW_MEM utiliza menos memória, e a macro WOLFSSL_SMALL_STACK utiliza mais memória *heap* do que stack, e memória *heap* é justamente o que está sendo medido. Para a operação de assinar uma mensagem, o consumo de memória com somente a macro B habilitada faz sentido, pois ela utiliza mais memória heap, o que explica seu maior valor se comparado a não

habilitar nenhuma das macros. Contudo, o valor obtido ao habilitar somente a macro B não faz sentido, pois supostamente era para menos memória ser utilizada, e ela foi a que apresentou o maior consumo de memória. Habilitar as duas macros ao mesmo parece ter feito isso acontecer, no entanto.

O critério utilizado para escolher qual combinação de macros utilizar foi o menor tempo total resultante da soma de tempos das três operações, pois como o objetivo deste trabalho é explorar certificação digital em todas as suas etapas, a configuração que permitisse a finalização mais rápidas das operações foi considerada a melhor. Neste caso, a configuração que permite isso é ter as duas macros habilitadas, e esta foi a configuração utilizada nos testes. Esta escolha também apresenta o segundo menor consumo de memória entre as opções, o que é bom. Entretanto, é importante ressaltar que dependendo da aplicação pode-se identificar a necessidade de priorizar certas funções, como verificação de assinatura sobre a geração de assinaturas, e nesse caso é interessante reavaliar essa decisão.

Tabela 4 – Média do consumo de memória para as operações de RSA com chaves de 4096 bits da biblioteca wolfSSL, para testar o impacto das macros RSA_LOW_MEM (A) e WOLFSSL_SMALL_STACK (B)

Algoritmo	**Gerar chaves**	**Assinar mensagem**	**Verificar assinatura**
RSA 4096 (macros A e B)	21.166,2	11.738,4	8044,0
RSA 4096 (somente macro A)	21.166,2	13.069,4	8044,0
RSA 4096 (somente macro B)	21.166,2	12.004,6	8044,0
RSA 4096 (nenhuma das duas)	21.166,2	11.472,0	8044,0

Fonte: Autoria própria.

5.2 ANÁLISE DOS RESULTADOS

Nesta seção serão apresentados e analisados os resultados dos testes realizados.

5.2.1 Geração de chaves

Em relação ao tempo de execução, é possível notar pela Tabela 5 que a biblioteca MbedTLS possui tempos muito menores para ECDSA com as curvas NIST, em relação às outras bibliotecas. Em contrapartida, a wolfSSL apresenta tempo menor com todos os outros algoritmos, e a diferença entre os tempos das curvas brainpool com as curvas NIST é bem menor, se comparado com a diferença da MbedTLS. O algoritmo RSA em ambas as implementações, mesmo com aceleração de hardware, continua sendo bastante custoso, principalmente ao se usar chaves com 4096 bits de tamanho, o que já era de se esperar. Como a Tabela 6 mostra, no entanto, os valores para o RSA 4096 possuem um alto desvio padrão, de aproximadamente 20 segundos para as duas bibliotecas, e portanto a média final não ficou muito confiável. Os outros algoritmos, no entanto, tiveram um desvio padrão

baixíssimo, e até mesmo o RSA 2048 possui um desvio padrão suficientemente baixo para a média ser confiável. Por fim, o algoritmo mais rápido mostrou-se ser o ed25519, com apenas 27 milissegundos necessários para gerar seu par de chaves.

Tabela 5 – Média do tempo de geração de chaves, em milissegundos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	94,6	530,6	168,1
secp521r1	173,8	1.954,1	-
brainpool256r1	1.231,7	736,1	-
brainpool512r1	3.980,5	2.026,1	-
RSA 2048	6.200,4	5.463,6	-
RSA 4096	43.646,3	37.643,9	-
ed448	-	106,5	-
ed25519	-	26,9	-

Fonte: Autoria própria.

Tabela 6 – Desvio padrão do tempo de geração de chaves, em milissegundos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	0,5	0,5	0,3
secp521r1	1,9	0,3	-
brainpool256r1	2,5	0,3	-
brainpool512r1	3,9	0,3	-
RSA 2048	3.343,8	3.569,3	-
RSA 4096	26.257,8	21.771,8	-
ed448	-	0,5	-
ed25519	-	0,3	-

Fonte: Autoria própria.

Agora, em relação ao consumo de memória, na Tabela 7 fica claro que a wolfSSL consome muito mais memória que as outras bibliotecas, exceto para os algoritmos EdDSA, que apresentam um consumo extremamente baixo de memória. Entretanto, é importante levar em consideração que as chaves, tanto na MbedTLS como na wolfSSL, são objetos complexos, o que explicaria um maior consumo para essas bibliotecas. Ainda assim, a diferença de consumo entre a wolfSSL e a mbedTLS é exorbitante. A biblioteca micro-ecc, por outro lado, requer que o usuário crie arrays para guardar os bytes de cada chave, o que explicaria seu menor tamanho. Uma possível explicação para o fato dos algoritmos da wolfSSL geralmente serem mais rápidos para gerar suas chaves seria o maior consumo de memória, mas isso acaba não fazendo muito sentido quando a MbedTLS consegue consumir menos memória e ser mais rápida com a curva secp256r1, e o algoritmo ed25519 consegue ser o mais rápido de todos e o segundo com menor consumo de memória.

A Tabela 8 demonstra que os ciclos de clock gerados pela operação de gerar as chaves está de acordo com o desempenho observado na medição do tempo. As curvas NIST da mbedTLS são bem menos custosos que suas correspondentes da wolfSSL, e contrário se

Tabela 7 – Média do consumo de memória para a geração de chaves, em bytes

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	715,6	28.784,0	104,0
secp521r1	3.470,0	28.784,0	-
brainpool256r1	946,4	29.852,0	-
brainpool512r1	1.787,2	29.852,0	-
RSA 2048	3.466,0	21.002,6	-
RSA 4096	6.786,2	21.166,2	-
ed448	-	252,8	-
ed25519	-	135,2	-

Fonte: Autoria própria.

aplica para os algoritmos brainpool: para a curva brainpool512r1, a wolfSSL utiliza metade dos ciclos de clock que a mesma curva da mbedTLS. Os algoritmos RSA continuam sendo os mais custosos, com a implementação da wolfSSL sendo mais performática que a da mbedTLS. Por fim, o algoritmo ed25519 continua sendo o melhor, apresentando o menor valor, de apenas 4 milhões de ciclos de clock. Em relação ao desvio padrão, a Tabela 9 indica que, com exceção do RSA, a maioria dos algoritmos possuíram resultados confiáveis para suas médias. Como o RSA apresentou alto desvio padrão no tempo de execução, já era de se esperar que o mesmo ocorresse com seus ciclos de clock.

Tabela 8 – Média do número de ciclos de clock da geração de chaves, em ciclos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	15.141.233,2	84.908.554,2	26.896.810,5
secp521r1	27.777.557,0	312.663.820,2	-
brainpool256r1	197.061.815,4	117.792.179,2	-
brainpool512r1	636.903.947,4	324.184.226,8	-
RSA 2048	992.075.602,0	874.210.748,5	-
RSA 4096	1.829.431.266,0	1.728.064.595	-
ed448	-	17.041.338,8	-
ed25519	-	4.301.285,8	-

Fonte: Autoria própria.

Tabela 9 – Desvio padrão do número de ciclos de clock para a geração de chaves, em ciclos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	55.990,58	30.034,1	21.277,0
secp521r1	311.029,1	23.717,2	-
brainpool256r1	389.449,1	30.690,5	-
brainpool512r1	644.660,3	34.622,3	-
RSA 2048	535.011.259,9	571.080.376,9	-
RSA 4096	1.292.367.6	996.467.828,6	-
ed448	-	17.292,7	-
ed25519	-	11.181,2	-

Fonte: Autoria própria.

5.2.2 Geração da assinatura

Agora, em relação ao tempo para assinar uma mensagem, a Tabela 10 nos mostra relações que são similares às feitas anteriormente na geração do par de chaves: a MbedTLS é superior com ECDSA utilizando as curvas NIST e inferior ao utilizar as curvas brainpool, e a wolfSSL é superior em todos os outros algoritmos. O algoritmo ed25519 continua sendo o mais rápido de todos. Entretanto, há uma nova relação que surgiu: para a MbedTLS, o RSA, com ambos os tamanhos de chaves, é mais rápido ao assinar do que o ECDSA com as curvas brainpool. A aceleração de hardware influencia neste resultado, pois desativando-a, o RSA 2048 leva por volta de 1.700,0 milissegundos para assinar uma mensagem, um aumento de aproximadamente 2 segundos. Para o RSA 4096 não é diferente, sem aceleração de hardware, ele leva por volta de 9 segundos para assinar a mensagem. A Tabela 11 demonstra que todos os resultados possuem bom nível de confiabilidade, pois todos possuem baixo desvio padrão.

Tabela 10 – Média do tempo para assinar a mensagem, em milissegundos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	108,7	536,5	175,7
secp521r1	207,9	1.967,1	-
brainpool256r1	1.246,0	741,4	-
brainpool512r1	4.010,4	2.035,3	-
RSA 2048	629,6	433,7	-
RSA 4096	2.707,9	2.070,25	-
ed448	-	113,4	-
ed25519	-	30,0	-

Fonte: Autoria própria.

Tabela 11 – Desvio padrão do tempo para assinar a mensagem, em milissegundos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	0,6	0,5	0,4
secp521r1	2,1	0,5	-
brainpool256r1	2,1	1,9	-
brainpool512r1	5,3	0,4	-
RSA 2048	7,7	0,6	-
RSA 4096	31,1	1,2	-
ed448	-	0,5	-
ed25519	-	0	-

Fonte: Autoria própria.

A Tabela 12 detalha a memória necessária para se realizar a assinatura de uma mensagem. Similarmente à etapa de geração de chaves, a biblioteca wolfSSL é a que mais consome memória no geral. Entretanto, agora é possível observar que, para essa biblioteca, o algoritmo RSA consome bem menos memória que os demais, exceto é claro, para os

algoritmo EdDSA, em específico o ed25519, em que não foi registrado nenhum consumo de memória. Isso pode ter acontecido por causa da forma como realizamos a medição, em que só contabilizamos memória heap, ou devido à própria implementação do algoritmo, que realmente não necessitar de memória adicional para assinar uma mensagem. O mesmo ocorreu com o algoritmo ECDSA de curva secp256r1 da biblioteca micro-ecc. Outro detalhe que é interessante observar é que, embora as curvas NIST possuem desempenhos de tempo muito melhores que as curvas brainpool, na biblioteca MbedTLS, seu consumo de memória é aproximadamente o mesmo, com as curvas brainpool consumindo um pouco menos de memória que as curvas NIST.

Tabela 12 – Média do consumo de memória para assinar uma mensagem, em bytes

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	1.722,8	38.420,0	0
secp521r1	2.893,6	38.420,0	-
brainpool256r1	1.701,2	39.488,0	-
brainpool512r1	2.820,8	39.488,0	-
RSA 2048	6.882,0	13.055,8	-
RSA 4096	13.533,6	11.738,4	-
ed448	-	252,0	-
ed25519	-	0	-

Fonte: Autoria própria.

Em relação ao ciclos de clocks contabilizados, a Tabela 13 demonstra um padrão similar ao observado anteriormente: as curvas NIST da MbedTLS requerem bem menos ciclos que as mesmas curvas da biblioteca wolfSSL e que as curvas brainpool; e o algoritmo ed25519 continua sendo o mais performático de todos. Entretanto, agora é possível observar que o RSA requer bem menos ciclos do que o necessário para a geração de chaves, pois agora ele está até mesmo mais performático que o algoritmo ECDSA com a curva brainpool512r1 da biblioteca MbedTLS. Inclusive, o algoritmo RSA com chaves de 2048 bits da wolfSSL é muito mais performática que o mesmo algoritmo da MbedTLS, e fica bem próximo da performance do algoritmo ECDSA com a curva secp256r1 da MbedTLS também. Além disso, conforme os dados da Tabela 14, os resultados das médias são bastante confiáveis, devido ao baixo desvio padrão apresentado por todos os algoritmos.

5.2.3 Verificação da assinatura

Considerando agora o tempo levado para verificar uma assinatura, segundo a Tabela 15, em relação ao algoritmo ECDSA com a curva secp256r1, a biblioteca micro-ecc agora se mostra a mais rápida, e com a curva secp521r1 a MbedTLS continua mais rápida que a wolfSSL. Com as curvas brainpool, as observações feitas até então se repetem: a wolfSSL é mais rápida do que a MbedTLS com estas curvas, por uma boa margem. A curva brainpool512r1 da MbedTLS foi a mais demorada para verificar uma assinatura,

Tabela 13 – Média do número de ciclos de clock para assinar uma mensagem, em ciclos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	17.408.477,0	85.822.371,2	28.106.546,3
secp521r1	33.258.746,8	314.734.243,6	-
brainpool256r1	199.378.077,6	118.732.881,2	-
brainpool512r1	641.610.814,2	325.638.116,4	-
RSA 2048	100.740.178,5	19.004.084,3	-
RSA 4096	433.269.037,4	331.247.838,6	-
ed448	-	18.146.413,0	-
ed25519	-	4.764.296,0	-

Fonte: Autoria própria.

Tabela 14 – Desvio padrão do número de ciclos de clock para assinar uma mensagem, em ciclos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	68.797,3	14.244,1	10.319,4
secp521r1	327.831,2	13.489,9	-
brainpool256r1	326.871,1	14.236,5	-
brainpool512r1	849.062,5	36.633,6	-
RSA 2048	1.232.329,8	73.445,9	-
RSA 4096	4.970.841,9	198.247,9	-
ed448	-	2.925,5	-
ed25519	-	2.843,7	-

Fonte: Autoria própria.

levando quase 19 segundos para concluir esta operação. Em relação aos algoritmos EdDSA, eles continuam rápidos, com o ed25519 sendo até mais rápido que o RSA da wolfSSL. Entretanto, a verificação de assinatura dos algoritmos RSA da MbedTLS possui os menores tempos de todos, não levando nem 5 milissegundos para concluir. Embora isso também tenha a ver com a aceleração de hardware, pois ao desativá-la o tempo levado para verificar uma assinatura com o RSA 2048 aumenta, este aumento não é muito grande, ficando por volta de 90 milissegundos. Ou seja, o RSA 2048 da MbedTLS, sem aceleração de hardware, é em média 30 milissegundos mais rápido que o mesmo algoritmo da wolfSSL, com a aceleração de hardware ativa, o que implica que as diferenças de implementação entre as bibliotecas também influenciam bastante no resultado. Não foi possível averiguar qual seria o tempo de verificação de assinatura do algoritmo RSA 2048 da biblioteca wolfSSL com a aceleração de hardware desativada, pois a execução do teste ficou presa na etapa de geração de chaves por 30 minutos, até que se decidiu encerrar a execução. Como a Tabela 16 mostra, as médias de tempo obtidas são confiáveis, pois todos os algoritmos demonstraram baixo desvio padrão.

A Tabela 17 detalha o consumo de memória para verificar uma assinatura. Um padrão parecido aos anteriores se mantém, com a wolfSSL sendo a biblioteca que mais consome memória no geral. Entretanto, agora é possível ver que o algoritmo ed25519

Tabela 15 – Média do tempo para verificar uma assinatura, em milissegundos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	376,8	1.021,2	200,2
secp521r1	755,4	3.759,6	-
brainpool256r1	5.001,5	1.424,3	-
brainpool512r1	18.430,3	3.909,1	-
RSA 2048	1,9	118,7	-
RSA 4096	4,0	888,9	-
ed448	-	286,4	-
ed25519	-	59,7	-

Fonte: Autoria própria.

Tabela 16 – Desvio padrão do tempo para para verificar uma assinatura, em milissegundos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	1,1	0,4	2,8
secp521r1	2,9	0,5	-
brainpool256r1	6,6	0,4	-
brainpool512r1	12,5	0,3	-
RSA 2048	0,2	0,4	-
RSA 4096	0,3	0,3	-
ed448	-	0,6	-
ed25519	-	1,7	-

Fonte: Autoria própria.

também consome memória, e que o único que aparentemente não consome memória adicional para realizar a verificação da assinatura é o ECDSA com a curva secp256r1 da biblioteca micro-ecc. Pode-se observar agora, também, que para a biblioteca MbedTLS, o algoritmo RSA é o que menos consome memória. Isto pode estar relacionado com o tempo de verificação do algoritmo, pois ele é extremamente rápido.

Tabela 17 – Média do consumo de memória para verificar uma assinatura, em bytes

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	3.508,8	36.324,0	0
secp521r1	6.116,4	36.324,0	-
brainpool256r1	3.319,2	37.392,0	-
brainpool512r1	6.095,2	37.392,0	-
RSA 2048	1.302,6	7.788,0	-
RSA 4096	2.581,6	8.044,0	-
ed448	-	4.692,0	-
ed25519	-	2.364,0	-

Fonte: Autoria própria.

A Tabela 18 detalha o número de ciclos de clock necessários para verificar uma mensagem. A performance dos algoritmos aqui anda lado a lado com o seu tempo de execução: o ECDSA com a curva brainpool512r1 da MbedTLS foi o menos performático,

com aproximadamente 3 bilhões de ciclos; e o algoritmo RSA com chaves de 2048 bits da mesma biblioteca foi o mais performático de todos, com aproximadamente 300 mil ciclos. Com exceção dos algoritmos RSA da MbedTLS, o ed25519 da wolfSSL se mostrou o mais rápido dentre os outros. Por fim, a Tabela 19 demonstra que as médias são confiáveis, pois todos os algoritmos apresentaram baixo desvio padrão para o número de ciclos de clock também.

Tabela 18 – Média do número de ciclos de clock para verificar uma mensagem, em ciclos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	60.286.655,8	163.391.980,0	32.028.084,2
secp521r1	120.867.253,4	120.867.253,4	-
brainpool256r1	800.253.074,4	227.880.363,6	-
brainpool512r1	2.948.825.804,0	625.456.416,2	-
RSA 2048	291.297	19.004.084,3	-
RSA 4096	644.955,9	142.257.788,1	-
ed448	-	45.786.648,2	-
ed25519	-	9.565.873,3	-

Fonte: Autoria própria.

Tabela 19 – Desvio padrão do número de ciclos de clock para verificar uma mensagem, em ciclos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	152.851,6	17.526,1	448.454,4
secp521r1	452.514,2	34.075,3	-
brainpool256r1	1.092.000,4	15.337,1	-
brainpool512r1	2.040.589,9	29.635,6	-
RSA 2048	23.728,9	17.453,3	-
RSA 4096	23.996,7	35.384,3	-
ed448	-	71.755,9	-
ed25519	-	260.795,7	-

Fonte: Autoria própria.

5.2.4 Geração do hash

Já em relação ao tempo levado para se gerar um hash, pode-se observar pela Tabela 20 que ele é desprezível, não passando de 5 milissegundos em todos os casos. Fora isto, a única outra conclusão relevante, é que o SHA3-256 é mais lento que o SHA-256 e o SHA-512, e que o SHAKE-256 com hash de tamanho de 512 bits é mais lento que o SHA3-256.

Por fim, a Tabela 21 nos mostra a memória utilizada pelos algoritmos de hash. É interessante notar os valores decimais, indicando que houve variação na memória utilizada para guardar o hash de uma mensagem, o que é estranho se considerarmos que cada hash deveria ter um tamanho fixo. De fato, os tamanhos são fixos, e em 9 das 10 execuções

Tabela 20 – Média do tempo para gerar o hash da mensagem, em milissegundos

Algoritmo	**MbedTLS**	**wolfSSL**
SHA-256	0,5	0,4
SHA-512	0,6	1,4
SHA3-256	2,2	1,6
SHAKE-256 (tamanho de 256 bytes)	-	2,1
SHAKE-256 (tamanho de 512 bytes)	-	2,3

Fonte: Autoria própria.

realizadas os valores foram iguais, e o esperado para cada algoritmo, como o SHA-256 com hash de 36 bytes de tamanho, por exemplo. Entretanto, para a biblioteca MbedTLS, todas as primeiras execuções dos algoritmos ocuparam mais memória que as outras, incluindo as execuções para a geração de chaves. Não foi possível identificar exatamente o porquê disso ocorrer, mas uma explicação possível para este fenômeno é que a primeira operação da MbedTLS tenha guardado algum tipo de cache, economizando memória nas operações subsequentes. Já em relação a wolfSSL, houve uma única execução, para os algoritmos SHA-256 e SHA3-256, em que foram registrados 44 bytes ao invés de 36 bytes. Neste caso, as execuções não foram as primeiras a ocorrer, então o motivo pode ser diferente do MbedTLS. Finalmente, o algoritmo SHAKE-256 foi os que mais ocupou memória, o que era de se esperar, pois o tamanho do hash gerado por ele é definido pelo usuário. No caso, para um tamanho de hash de 256 bytes, foram ocupados 260 bytes; e para um tamanho de 512 bytes, foram ocupados 516 bytes. Os quatro bytes extras provavelmente devem ser de algum overhead interno da biblioteca.

Tabela 21 – Média do consumo de memória para gerar o hash da mensagem, em bytes

Algoritmo	**MbedTLS**	**wolfSSL**
SHA-256	44,7	36,8
SHA-512	71,6	68
SHA3-256	36	36,8
SHAKE-256 (tamanho de 256 bytes)	-	260
SHAKE-256 (tamanho de 512 bytes)	-	516

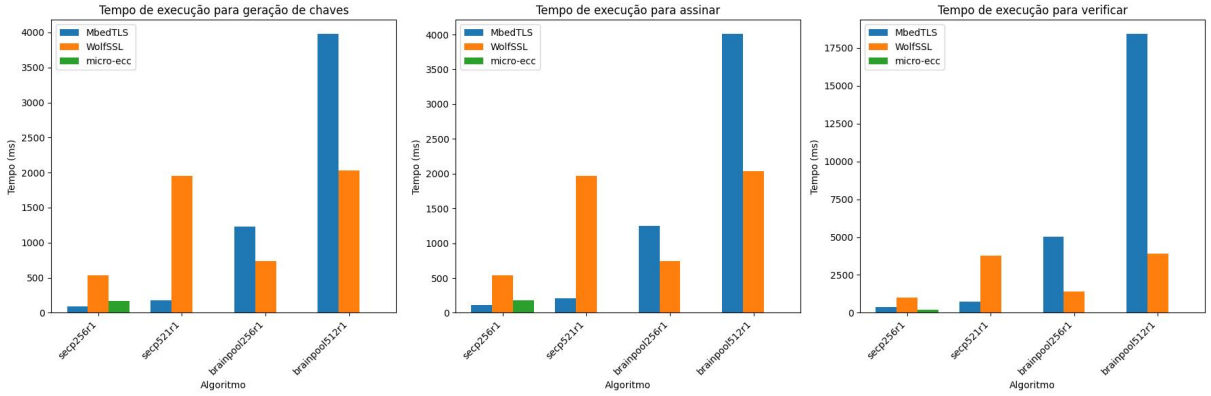
Fonte: Autoria própria.

5.2.5 Comparativos entre algoritmos de assinatura digital

Para facilitar a visualização das diferenças de tempo entre os algoritmos, criou-se gráficos comparando: (1) algoritmos de assinatura digital da mesma família; e (2) todos os algoritmos de assinatura digital. A Figura 11 demonstra o tempo de execução para as operações de gerar chaves, assinar a mensagem e verificar a assinatura, entre os algoritmos da ECDSA. A Figura 12 ilustra o mesmo, mas para os algoritmos RSA, enquanto que a Figura 13 está relacionada aos algoritmos EdDSA. Por fim, a Figura 14 compara todos

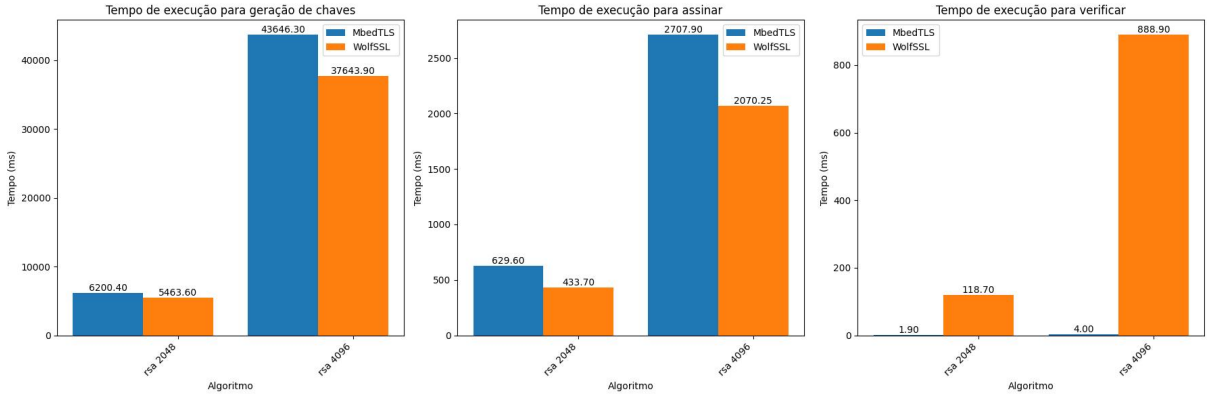
os algoritmos, utilizando uma escala logarítmica no eixo Y, para impedir que tempos de execução muito grandes como os do RSA impeçam a visualização de tempos pequenos.

Figura 11 – Comparativo de tempo entre algoritmos ECDSA.



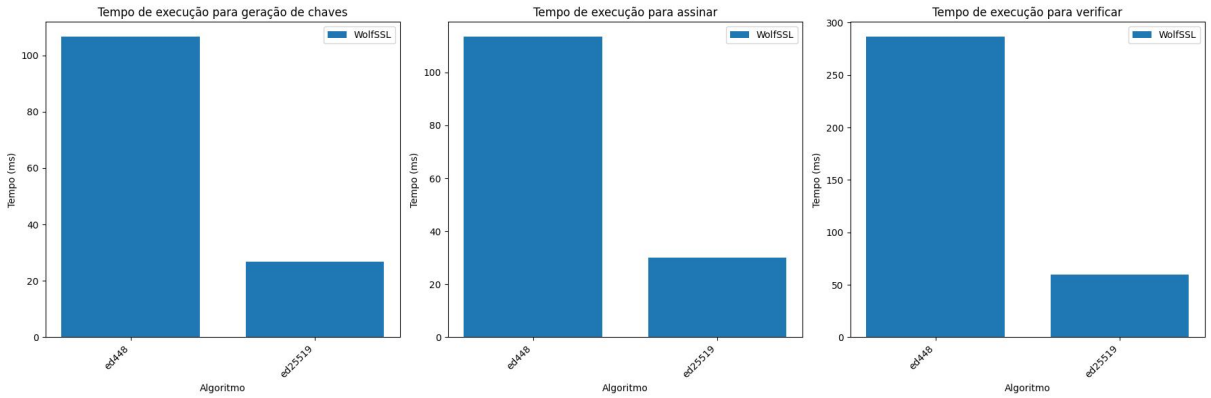
Fonte: Autoria própria.

Figura 12 – Comparativo de tempo entre algoritmos RSA.



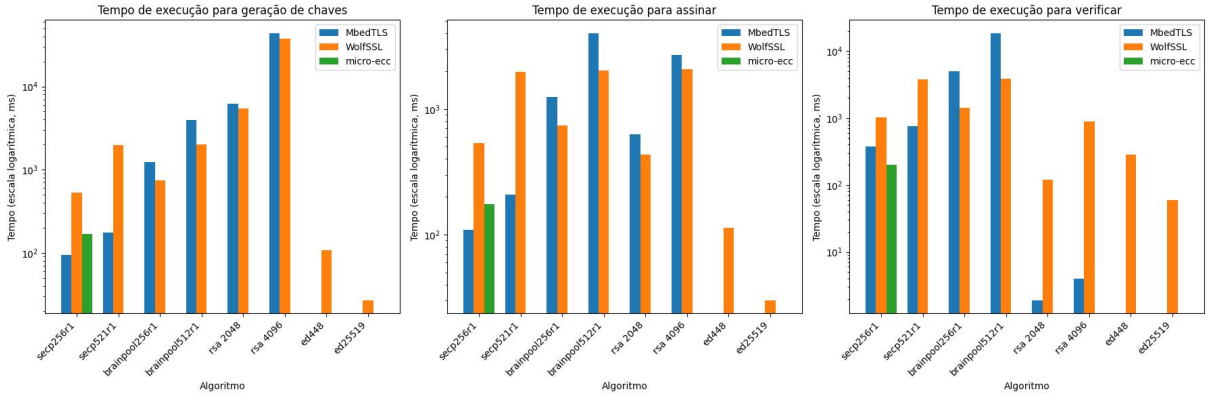
Fonte: Autoria própria.

Figura 13 – Comparativo de tempo entre algoritmos EdDSA.



Fonte: Autoria própria.

Figura 14 – Comparativo de tempo entre todos os algoritmos.



Fonte: Autoria própria.

6 CONCLUSÃO

Este trabalho teve por objetivo analisar a viabilidade da realização de operações relacionadas à certificação digital em um dispositivo IoT com restrição de recursos, mais especificamente o ESP32, visando a garantia da autenticidade e integridade dos dados em sua transmissão.

Inicialmente foi realizada a fundamentação teórica, onde estudou-se sobre os conceitos fundamentais relacionados principalmente à criptografia assimétrica e à Internet das Coisas, o que permitiu melhor entendimento e a posterior implementação de uma solução. Depois, executou-se uma revisão sistemática da literatura sobre o tema, onde diversos trabalhos relacionados foram lidos, para entender as diferenças e as semelhanças entre esses trabalhos e o presente trabalho. Embora encontrou-se trabalhos mais alinhados com o trabalho proposto, os quais foram resumidos, nenhum deles demonstrou preocupação exclusiva com a garantia de autenticidade e integridade dos dados do aparelho IoT ao emiti-los para um terceiro, com muito foco sendo dado principalmente para a garantia da confidencialidade e privacidade dos dados. Isso demonstrou que o tema deste trabalho ainda havia sido pouco explorado na literatura.

Após isso, foram analisadas diversas bibliotecas criptográficas, e três das mais bem avaliadas disponíveis para o dispositivo foram escolhidas para integrar a API criptográfica desenvolvida deste trabalho, a CryptoAPI. Analisou-se cada algoritmo em conformidade com o ITI dessas bibliotecas em quesito de tempo de execução, consumo de memória e ciclos de clock. Os resultados demonstraram que é sim viável realizar essas operações criptográficas no dispositivo, principalmente com o algoritmo de assinatura digital Ed25519 da biblioteca WolfSSL, pois além dele ser um dos algoritmos mais rápidos para todas as operações essenciais relacionadas à assinatura digital, levando menos de 105 milissegundos em média em cada uma, ele também mostrou-se econômico em termos de memória. Além do mais, diferentemente do algoritmo ECDSA com a curva secp256r1, que também mostrou bons resultados, o Ed25519 é um algoritmo aceito pelo ITI.

Com isso, espera-se que esses resultados possam ser úteis ao desenvolvimento de alguma solução de certificação digital para dispositivos IoT em contextos públicos, onde poderá ser possível verificar a integridade e autenticidade de diversos dados gerados pelas pessoas em seu dia-a-dia.

6.1 TRABALHOS FUTUROS

Para trabalhos futuros, há diversos caminhos que podem ser seguidos. Um deles é adaptar a CryptoAPI para suportar algoritmos de assinatura digital pós-quânticos, para que aplicações que a utilizem estejam preparadas a resistir à possíveis ataques por computadores quânticos no futuro. Outra possibilidade, é a de adaptar a CryptoAPI para uma aplicação específica, onde as particularidades dela são levadas em consideração no

uso de suas operações. Neste caso, um exemplo seria o foco maior em realizar a assinatura de mensagens, com uma aplicação que não precisasse realizar a verificação de assinaturas, e que precisasse gerar o par de chaves no máximo uma única vez. Isto implicaria que modificar o programa para otimizar o tempo de geração de assinaturas seria o mais importante. Além dessa possibilidade, também poderia ser trabalhado o caminho onde certificados digitais gerados por ACs são salvos no dispositivo, e ele utilizaria as chaves públicas que estão dentro dos certificados. Por fim, uma última ideia de trabalho futuro é gerar as assinaturas em um padrão avançado, como o CMS, em que se esperam assinaturas maiores e com um maior tempo de geração.

REFERÊNCIAS

AHMED, Adel A; BARUKAB, Omar M. Unforgeable Digital Signature Integrated into Lightweight Encryption Based on Effective ECDH for Cybersecurity Mechanism in Internet of Things. **Processes**, Multidisciplinary Digital Publishing Institute, v. 10, n. 12, p. 2631, 2022. Disponível em: https://www.mdpi.com/2227-9717/10/12/2631?type=check_updateversion%3D3. Acesso em: 28 out. 2023.

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. The internet of things: A survey. **Computer networks**, Elsevier, v. 54, n. 15, p. 2787–2805, 2010. Disponível em: https://www.sciencedirect.com/science/article/pii/S1389128610001568. Acesso em: 28 out. 2023.

BAUER, Johannes *et al.* ECDSA on things: IoT integrity protection in practise. *In:* SPRINGER. INFORMATION and Communications Security: 18th International Conference, ICICS 2016, Singapore, Singapore, November 29–December 2, 2016, Proceedings 18. [*S.l.*]: Springer, Cham, 2016. P. 3–17. Disponível em: https://link.springer.com/chapter/10.1007/978-3-319-50011-9_1. Acesso em: 26 mai. 2024.

BELLARE, Mihir; ROGAWAY, Phillip. Introduction to modern cryptography. **Lecture Notes**, 2001. Disponível em: https://web.cs.ucdavis.edu/~rogaway/classes/227/fall01/book/main.pdf. Acesso em: 28 out. 2023.

BHARDWAJ, Isha; KUMAR, Ajay; BANSAL, Manu. A review on lightweight cryptography algorithms for data security and authentication in IoTs. *In:* IEEE. 2017 4th International Conference on Signal Processing, Computing and Control (ISPCC). [*S.l.*: *s.n.*], 2017. P. 504–509. Disponível em: https://ieeexplore.ieee.org/abstract/document/8269731/. Acesso em: 27 out. 2023.

BRZICA, Hrvoje; HERCEG, Boris; STANČIĆ, Hrvoje. Long-term preservation of validity of electronically signed records. Department of Information e Communication Sciences, Faculty of Humanities ..., 2013. Disponível em: http://darhiv.ffzg.unizg.hr/id/eprint/8291/. Acesso em: 28 jun. 2024.

CLARK, Joseph; ALI, Farha. Analysis of ECDSA's Computational Impact on IoT Network Performance. *In:* PROCEEDINGS of the 2023 ACM Southeast Conference. [*S.l.*: *s.n.*], 2023. P. 196–200. Disponível em: https://dl.acm.org/doi/abs/10.1145/3564746.3587013. Acesso em: 26 mai. 2024.

ČOLAKOVIĆ, Alem; HADŽIALIĆ, Mesud. Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues. **Computer networks**, Elsevier, v. 144, p. 17–39, 2018. Disponível em:

https://www.sciencedirect.com/science/article/pii/S1389128618305243. Acesso em: 18 jun. 2024.

DHANDA, Sumit Singh; SINGH, Brahmjit; JINDAL, Poonam. Lightweight cryptography: a solution to secure IoT. **Wireless Personal Communications**, Springer, v. 112, n. 3, p. 1947–1980, 2020. Disponível em: https://link.springer.com/article/10.1007/s11277-020-07134-3. Acesso em: 28 out. 2023.

ESPRESSIF. **ESP-IDF Programming Guide**. 2023. Disponível em: https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32/index.html. Acesso em: 8 out. 2024.

_____. **ESP-IDF-C++**. 2023. Disponível em: https://components.espressif.com/components/espressif/esp-idf-cxx. Acesso em: 8 out. 2024.

_____. **ESP32 Technical Reference Manual Version 5.2**. 2024. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf. Acesso em: 6 nov. 2024.

_____. **ESP32-WROOM-32E ESP32-WROOM-32UE Datasheet Version 1.7**. 2024. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf. Acesso em: 6 nov. 2024.

_____. **Mbed TLS**. 2024. Disponível em: https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32/api-reference/protocols/mbedtls.html. Acesso em: 4 out. 2024.

FAUZAN, Andaresta; SUKARNO, Parman; WARDANA, Aulia Arif. Overhead Analysis of the Use of Digital Signature in MQTT Protocol for Constrained Device in the Internet of Things System. *In:* IEEE. 2020 3rd International Conference on Computer and Informatics Engineering (IC2IE). [*S.l.*: *s.n.*], 2020. P. 415–420. Disponível em: https://ieeexplore.ieee.org/abstract/document/9274651. Acesso em: 28 out. 2023.

AL-FUQAHA, Ala *et al.* Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE communications surveys & tutorials**, Ieee, v. 17, n. 4, p. 2347–2376, 2015. Disponível em: https://ieeexplore.ieee.org/abstract/document/7123563. Acesso em: 18 jun. 2024.

HA, Duy An; NGUYEN, Kha Tho; ZAO, John K. Efficient authentication of resource-constrained IoT devices based on ECQV implicit certificates and datagram transport layer security protocol. *In:* PROCEEDINGS of the 7th Symposium on

Information and Communication Technology. [*S.l.*: *s.n.*], 2016. P. 173–179. Disponível em: https://dl.acm.org/doi/abs/10.1145/3011077.3011108. Acesso em: 18 abr. 2024.

HOMADI, Salwan Rafa; DAWOOD, Omar A. Effective Mutual Authentication Scheme and Message Authentication for Internet of Things (IoT) using Fog Computing Technology. *In:* IEEE. 2023 16th International Conference on Developments in eSystems Engineering (DeSE). [*S.l.*: *s.n.*], 2023. P. 858–863. Disponível em: https://ieeexplore.ieee.org/abstract/document/10468905. Acesso em: 12 abr. 2024.

HOU, Yingzhe *et al.* Certificate-based parallel key-insulated aggregate signature against fully chosen key attacks for industrial Internet of Things. **IEEE Internet of Things Journal**, IEEE, v. 8, n. 11, p. 8935–8948, 2021. Disponível em: https://ieeexplore.ieee.org/abstract/document/9344710. Acesso em: 2 mai. 2024.

ICP-BRASIL. **PADRÕES E ALGORITMOS CRIPTOGRÁFICOS DA ICP-BRASIL**. 2022. Disponível em: https://www.gov.br/iti/pt-br/assuntos/legislacao/documentos-principais/IN2022_22_DOC_ICP_01.01_assinado.pdf. Acesso em: 8 out. 2024.

ICPBRASIL. **VISÃO GERAL SOBRE ASSINATURAS DIGITAIS NA ICP-BRASIL**. 2015. Disponível em: https://www.gov.br/iti/pt-br/central-de-conteudo/doc-icp-15-v-3-0-visao-geral-sobre-assin-dig-na-icp-brasil-pdf. Acesso em: 28 jun. 2024.

INMETRO. **Inmetro aprova primeira bomba de combustível com assinatura eletrônica**. 2023. Disponível em: https://www.gov.br/inmetro/pt-br/centrais-de-conteudo/noticias/inmetro-aprova-primeira-bomba-de-combustivel-com-assinatura-eletronica. Acesso em: 28 jun. 2024.

ITU-T. **Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks**. [*S.l.*], 2019. Disponível em: https://www.itu.int/rec/T-REC-X.509-201910-I/en. Acesso em: 30 jun. 2024.

JOLTWALLET. **littlefs**. 2024. Disponível em: https://components.espressif.com/components/joltwallet/littlefs. Acesso em: 8 out. 2024.

JOSEFSSON, S.; LIUSVAARA, I. **Edwards-Curve Digital Signature Algorithm (EdDSA)**. 2017. Disponível em: https://www.rfc-editor.org/rfc/rfc8032#page-43. Acesso em: 10 nov. 2024.

KAÂNICHE, Nesrine; JUNG, Eunjin; GEHANI, Ashish. Efficiently validating aggregated IoT data integrity. *In:* IEEE. 2018 IEEE Fourth International Conference on Big Data

Computing Service and Applications (BigDataService). [*S.l.*: *s.n.*], 2018. P. 260–265. Disponível em: https://ieeexplore.ieee.org/abstract/document/8405721. Acesso em: 13 mai. 2024.

KATZ, Jonathan. **Digital signatures**. [*S.l.*]: Springer, 2010. v. 1. Disponível em: https://link.springer.com/book/10.1007/978-0-387-27712-7. Acesso em: 28 out. 2023.

LI, Bing *et al.* A Lightweight Authentication and Key Agreement Protocol for IoT Based on ECC. *In:* IEEE. 2021 International Conference on Advanced Computing and Endogenous Security. [*S.l.*: *s.n.*], 2022. P. 1–5. Disponível em: https://ieeexplore.ieee.org/abstract/document/10013341. Acesso em: 12 abr. 2024.

LI, He *et al.* Cumulative message authentication codes for resource-constrained IoT networks. **IEEE Internet of Things Journal**, IEEE, v. 8, n. 15, p. 11847–11859, 2021. Disponível em: https://ieeexplore.ieee.org/abstract/document/9408605. Acesso em: 18 abr. 2024.

LI, Jian *et al.* A novel message authentication scheme with absolute privacy for the internet of things networks. **IEEE Access**, IEEE, v. 8, p. 39689–39699, 2020. Disponível em: https://ieeexplore.ieee.org/abstract/document/9007751. Acesso em: 12 abr. 2024.

LI, Shancang; XU, Li Da; ZHAO, Shanshan. The internet of things: a survey. **Information systems frontiers**, Springer, v. 17, p. 243–259, 2015. Disponível em: https://link.springer.com/article/10.1007/S10796-014-9492-7. Acesso em: 28 out. 2023.

LIU, Jianghua *et al.* Lightweight authentication scheme for data dissemination in cloud-assisted healthcare IoT. **IEEE Transactions on Computers**, IEEE, v. 72, n. 5, p. 1384–1395, 2022. Disponível em: https://ieeexplore.ieee.org/abstract/document/9893346. Acesso em: 28 out. 2023.

LOLI, Gabriel B. **esp32-crypto-api**. 2024. Disponível em: https://github.com/siegjor/esp32-crypto-api. Acesso em: 9 dez. 2024.

MACKAY, Ken. **micro-ecc**. 2024. Disponível em: https://github.com/kmackay/micro-ecc. Acesso em: 4 out. 2024.

MALIK, Manisha; DUTTA, Maitreyee; GRANJAL, Jorge *et al.* L-ecqv: Lightweight ecqv implicit certificates for authentication in the internet of things. **IEEE Access**, IEEE, v. 11, p. 35517–35540, 2023. Disponível em: https://ieeexplore.ieee.org/abstract/document/10080960. Acesso em: 18 abr. 2024.

MAMUN, Quazi; RANA, Muhammad. A Key Management Scheme for Establishing an Encryption-Based Trusted IoT System. *In:* IEEE. 2019 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE). [*S.l.*: *s.n.*], 2019. P. 41–46. Disponível em: https://ieeexplore.ieee.org/abstract/document/8908650. Acesso em: 28 out. 2023.

MBEDTLS. **mbedtls**. 2024. Disponível em: https://github.com/Mbed-TLS/mbedtls. Acesso em: 4 out. 2024.

MORIARTY, K. *et al.* **PKCS 1: RSA Cryptography Specifications Version 2.2**. 2016. Disponível em: https://www.rfc-editor.org/rfc/rfc8017.html#appendix-A.1.1. Acesso em: 15 nov. 2024.

MÖSSINGER, Max *et al.* Towards quantifying the cost of a secure IoT: Overhead and energy consumption of ECC signatures on an ARM-based device. *In:* IEEE. 2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM). [*S.l.*: *s.n.*], 2016. P. 1–6. Disponível em: https://ieeexplore.ieee.org/abstract/document/7523559. Acesso em: 4 jun. 2024.

NIST. **Digital Signature Standard (DSS)**. 2023. Disponível em: https://csrc.nist.gov/pubs/fips/186-5/final. Acesso em: 14 nov. 2024.

_____. **NIST Retires SHA-1 Cryptographic Algorithm**. 2022. Disponível em: https://www.nist.gov/news-events/news/2022/12/nist-retires-sha-1-cryptographic-algorithm. Acesso em: 8 out. 2024.

PANDA, Prabhat Kumar; CHATTOPADHYAY, Sudipta. A secure mutual authentication protocol for IoT environment. **Journal of Reliable Intelligent Environments**, Springer, v. 6, n. 2, p. 79–94, 2020. Disponível em: https://link.springer.com/article/10.1007/s40860-020-00098-y. Acesso em: 8 abr. 2024.

PÖHLS, Henrich C. JSON sensor signatures (JSS): End-to-end integrity protection from constrained device to IoT application. *In:* IEEE. 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. [*S.l.*: *s.n.*], 2015. P. 306–312. Disponível em: https://ieeexplore.ieee.org/abstract/document/7284966. Acesso em: 2 jun. 2024.

PORAMBAGE, Pawani *et al.* Two-phase authentication protocol for wireless sensor networks in distributed IoT applications. *In:* IEEE. 2014 IEEE Wireless Communications and Networking Conference (WCNC). [*S.l.*: *s.n.*], 2014. P. 2728–2733. Disponível em: https://ieeexplore.ieee.org/abstract/document/6952860. Acesso em: 28 out. 2023.

RAO, Vidya; PREMA, KV. Comparative study of lightweight hashing functions for resource constrained devices of IoT. *In:* IEEE. 2019 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS). [*S.l.*: *s.n.*], 2019. P. 1–5. Disponível em: https://ieeexplore.ieee.org/abstract/document/9031038. Acesso em: 27 out. 2023.

_____. Light-weight hashing method for user authentication in Internet-of-Things. **Ad Hoc Networks**, Elsevier, v. 89, p. 97–106, 2019. Disponível em: https://www.sciencedirect.com/science/article/pii/S1570870518307716. Acesso em: 26 mai. 2024.

SÁ, Acácia de; SOUZA, Evandio. **Assinatura eletrônica pelo portal 'gov.br': regulamentação e aplicação prática**. 2024. Disponível em: https://www.conjur.com.br/2024-mar-17/assinatura-eletronica-pelo-portal-gov-br-regulamentacao-e-aplicacao-pratica/. Acesso em: 14 nov. 2024.

SACHAN, Anuj; KUMAR, Neetesh; ADWITEEYA, Adwiteeya. Light Weighted Mutual Authentication and Dynamic Key Encryption for IoT Devices Applications. *In:* IEEE. 2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT). [*S.l.*: *s.n.*], 2019. P. 1–6. Disponível em: https://ieeexplore.ieee.org/abstract/document/8977672. Acesso em: 12 abr. 2024.

SCIANCALEPORE, Savio *et al.* Public key authentication and key agreement in IoT devices with minimal airtime consumption. **IEEE Embedded Systems Letters**, IEEE, v. 9, n. 1, p. 1–4, 2016. Disponível em: https://ieeexplore.ieee.org/abstract/document/7748553. Acesso em: 12 abr. 2024.

STALLINGS, William. **Criptography and Network Security: Principles and Practice**. 8. ed. Harlow: Pearson Education, 2020.

STINSON, Douglas R; PATERSON, Maura B. **Cryptography: Theory and Practice**. 4. ed. Boca Raton: CRC Press, 2019.

SUÁREZ-ALBELA, Manuel; FRAGA-LAMAS, Paula; FERNÁNDEZ-CARAMÉS, Tiago M. A Practical Evaluation on RSA and ECC-Based Cipher Suites for IoT High-Security Energy-Efficient Fog and Mist Computing Devices. **Sensors**, v. 18, n. 11, 2018. ISSN 1424-8220. DOI: 10.3390/s18113868. Disponível em: https://www.mdpi.com/1424-8220/18/11/3868. Acesso em: 14 nov. 2024.

SUN, Jiangfeng *et al.* Mutual authentication scheme for the device-to-server communication in the internet of medical things. **IEEE Internet of Things Journal**, IEEE, v. 8, n. 21, p. 15663–15671, 2021. Disponível em: https://ieeexplore.ieee.org/abstract/document/9426898. Acesso em: 8 abr. 2024.

UFSC. **Assinatura Eletrônica Avançada do GOV.BR**. 2024. Disponível em: https://e.ufsc.br/assinatura-eletronica-avancada-do-gov-br/. Acesso em: 14 nov. 2024.

ULLAH, Ikram; MERATNIA, Nirvana; HAVINGA, Paul JM. IMAC: Implicit message authentication code for IoT devices. *In:* IEEE. 2020 IEEE 6th World Forum on Internet of Things (WF-IoT). [*S.l.*: *s.n.*], 2020. P. 1–6. Disponível em: https://ieeexplore.ieee.org/abstract/document/9221331. Acesso em: 2 mai. 2024.

UNIÃO EUROPEIA, Jornal Oficial da. **REGULAMENTO (UE) N.o 910/2014 DO PARLAMENTO EUROPEU E DO CONSELHO**. 2014. Disponível em: https://eur-lex.europa.eu/legal-content/PT/TXT/HTML/?uri=CELEX:32014R0910#d1e3079-73-1. Acesso em: 28 jun. 2024.

VERMA, Girraj Kumar; KUMAR, Neeraj *et al.* SCBS: a short certificate-based signature scheme with efficient aggregation for industrial-internet-of-things environment. **IEEE Internet of Things Journal**, IEEE, v. 8, n. 11, p. 9305–9316, 2021. Disponível em: https://ieeexplore.ieee.org/abstract/document/9343265. Acesso em: 11 mai. 2024.

VERMA, Girraj Kumar; SINGH, BB *et al.* CB-CAS: Certificate-based efficient signature scheme with compact aggregation for industrial Internet of Things environment. **IEEE Internet of Things Journal**, IEEE, v. 7, n. 4, p. 2563–2572, 2019. Disponível em: https://ieeexplore.ieee.org/abstract/document/8854296. Acesso em: 2 mai. 2024.

WANG, Xingmiao *et al.* A new RFID ultra-lightweight authentication protocol for medical privacy protection in smart living. **Computer Communications**, Elsevier, v. 186, p. 121–132, 2022. Disponível em: https://www.sciencedirect.com/science/article/pii/S0140366422000202. Acesso em: 28 out. 2023.

WEI, Jiannan; PHUONG, Tran Viet Xuan; YANG, Guomin. An efficient privacy preserving message authentication scheme for internet-of-things. **IEEE Transactions on Industrial Informatics**, IEEE, v. 17, n. 1, p. 617–626, 2020. Disponível em: https://ieeexplore.ieee.org/abstract/document/8988199. Acesso em: 1 jun. 2024.

WOLFSSL. **Building wolfSSL**. 2023. Disponível em: https://www.wolfssl.com/documentation/manuals/wolfssl/chapter02.html#use_fast_math. Acesso em: 8 out. 2024.

_____. **Embedded TLS Library**. 2024. Disponível em: https://www.wolfssl.com/. Acesso em: 4 out. 2024.

_____. **wolfssl**. 2024. Disponível em: https://github.com/wolfSSL/wolfssl. Acesso em: 4 out. 2024.

YAN, Haotian *et al.* Spmac: Scalable prefix verifiable message authentication code for internet of things. **IEEE Transactions on Network and Service Management**, IEEE, v. 19, n. 3, p. 3453–3464, 2022. Disponível em: https://ieeexplore.ieee.org/abstract/document/9761985. Acesso em: 6 mai. 2024.

YAVUZ, Attila Altay; OZMEN, Muslum Ozgur. Ultra lightweight multiple-time digital signature for the internet of things devices. **IEEE Transactions on Services Computing**, IEEE, v. 15, n. 1, p. 215–227, 2019. Disponível em: https://ieeexplore.ieee.org/abstract/document/8762164. Acesso em: 28 out. 2023.

YOUNIS, Mohamed; FARRAG, Osama; ALTHOUSE, Bryan. TAM: a tiered authentication of multicast protocol for ad-hoc networks. **IEEE Transactions on Network and Service Management**, IEEE, v. 9, n. 1, p. 100–113, 2011. Disponível em: https://ieeexplore.ieee.org/abstract/document/6094287. Acesso em: 2 mai. 2024.

ZHAO, Guanglei *et al.* A novel mutual authentication scheme for Internet of Things. *In:* IEEE. PROCEEDINGS of 2011 international conference on modelling, identification and control. [*S.l.*: *s.n.*], 2011. P. 563–566. Disponível em: https://ieeexplore.ieee.org/abstract/document/5973767/. Acesso em: 22 abr. 2024.

ZHU, Fei *et al.* Certificate-based anonymous authentication with efficient aggregation for wireless medical sensor networks. **IEEE Internet of Things Journal**, IEEE, v. 9, n. 14, p. 12209–12218, 2021. Disponível em: https://ieeexplore.ieee.org/abstract/document/9647005. Acesso em: 11 mai. 2024.

APÊNDICE A – ARTIGO SBC

Certificação Digital em Dispositivos da Internet das Coisas

Gabriel Bristot Loli, Thaís Bardini Idalino, Gabriel Estevam de Oliveira

Dep. De Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

bristotgl@gmail.com, thais.bardini@ufsc.br,
gabriel.estevam@posgrad.ufsc.br

Abstract. *The evolution of technology has enabled the Internet of Things (IoT), where everyday devices like watches and refrigerators communicate through data-sharing networks. While beneficial, these networks face security challenges due to the limited computational power of IoT devices, making it harder to ensure protection. This work proposes a digital certification solution to ensure the authenticity, integrity, and legitimacy of public data transmitted by IoT devices, as existing efforts often focus on privacy and confidentiality, which are unnecessary in public data contexts.*

Resumo. *A evolução tecnológica possibilitou a Internet das Coisas (IoT), onde dispositivos comuns, como relógios e geladeiras, comunicam-se por redes de dados. Apesar dos benefícios, garantir a segurança é um desafio, pois os dispositivos têm baixo poder computacional. Este trabalho propõe uma solução de certificação digital para garantir a autenticidade, integridade e legitimidade dos dados públicos transmitidos por dispositivos IoT, já que o foco atual em privacidade e confidencialidade não se aplica a esse contexto.*

1. Introdução

O avanço de paradigmas tecnológicos, como a Internet das Coisas (IoT), tem transformado a forma como interagimos com o mundo. A IoT consiste em uma rede global de dispositivos conectados, capazes de coletar e compartilhar dados por meio de sensores, aplicando-se em áreas como saúde, indústria e cidades inteligentes i (Li; Xu; Zhao, 2015; Atzori; Iera; Morabito, 2010).

Apesar dos benefícios, a interconexão de bilhões de dispositivos amplia a superfície para ataques, trazendo desafios de segurança relacionados à integridade e autenticidade dos dados e à identificação do dispositivo remetente (Atzori; Iera; Morabito, 2010). Soluções com assinaturas e certificados digitais são viáveis, mas limitadas pela restrição de recursos computacionais típicas dos dispositivos IoT (Bellare; Rogaway, 2001; Katz, 2010; Ahmed; Barukab, 2022)

Estudos existentes focam em garantir privacidade e confidencialidade, atributos irrelevantes para dados públicos. Exemplos como bombas de combustível regulamentadas pelo INMETRO mostram a importância de assegurar que os dados são íntegros, autênticos e oriundos de dispositivos legítimos, utilizando assinaturas digitais (INMETRO, 2023).

Assim, este trabalho propõe uma solução de certificação digital para dispositivos IoT em contextos de dados públicos, utilizando algoritmos aceitos pelo ITI (ICP-BRASIL, 2022), garantindo validade jurídica, integridade e autenticidade na transmissão de dados capturados por sensores.

Pergunta de Pesquisa: *Como é possível atribuir autenticidade e integridade dos dados emitidos por dispositivos IoT?*

Objetivo Geral: Propor uma solução de certificação digital em dispositivos IoT de medição, que garanta a integridade e autenticidade dos dados transmitidos.

Objetivos Específicos:

- Selecionar algoritmos de assinatura digital que consigam ser executados dentro de dispositivos com limitações de recursos, como processamento e memória.
- Implementar uma prova de conceito de certificação digital em dispositivos IoT, sendo que o dispositivo deve ser capaz de assinar dados digitalmente.
- Realizar experimentos com os algoritmos escolhidos em um dispositivo IoT para medir o tempo de execução, consumo de memória e ciclos de clock.

3. Revisão sistemática da literatura

3.1. Metodologia

Com o intuito de entender o estado atual da literatura relacionada ao uso de certificação digital em dispositivos IoT, realizou-se uma revisão sistemática da literatura sobre o tema. A busca foi executada em quatro bases de artigos acadêmicos: IEEEXplore, ACM Digital Library, Springer Link e Google Scholar. A *string* de busca utilizada no campo de busca de cada um destes sites foi:

(“iot” OR “internet of things” OR “embedded systems” OR “arduino” OR “esp”') AND (“integrity” OR “authentic”) AND (“digital signature” OR “digital certificate” OR “message authentication code” OR “asymmetric'” OR “public-key”' OR “cryptography” OR “certification”)*

Os resultados das buscas foram filtrados por relevância. O processo de seleção dos trabalhos relacionados para revisão encontra-se resumido na Tabela 1. A Busca Inicial representa a quantidade inicial total de publicações resultantes da pesquisa pela *string* de busca supracitada. Na Seleção 1, selecionou-se os trabalhos acadêmicos possivelmente relacionados com o tema pelo título, até que a busca não retornasse nenhum resultado interessante por algumas páginas; na Seleção 2, filtrou-se os artigos mais relevantes pelo *abstract*; por fim, na Seleção 3, os trabalhos resultantes foram lidos: alguns parcialmente, até notar-se pouca afinidade com o objetivo deste trabalho; e os que demonstravam maior afinidade foram lidos por completo. Destes que foram lidos na íntegra, escolheu-se os mais relevantes.

Tabela 1. Resultados da revisão sistemática da literatura

Base de dados	Busca Inicial	Seleção 1	Seleção 2	Seleção 3
IEEXplore	6.671	175	25	2
ACM DL	3.153	29	3	1
Spinger Link	2251	49	5	0
Google Scholar	111.000	47	10	4
Total	**123.075**	**300**	**48**	**7**

3.2. Trabalhos relacionados

Os estudos escolhidos, de forma similar a este trabalho, exploram a utilização de assinaturas digitais em dispositivos IoT, analisando o impacto computacional, eficiência e a aplicabilidade prática de diferentes algoritmos. Entretanto, diferentemente deste trabalho, todos eles acabam utilizando-se, em maior ou menor grau, de algoritmos ou protocolos que conferem confidencialidade aos dados transmitidos, algo que não é o foco deste trabalho.

Fauzan, Sukarno e Wardana (2020) investigam o overhead de recursos como tempo de verificação, memória e envio de mensagens no protocolo MQTT utilizando EdDSA com AES para criptografia ponta-a-ponta, mostrando que dispositivos com mais memória, como o ESP32, têm desempenho significativamente superior. Rao e Prema (2019) focam em um esquema de autenticação com cBLAKE2b e ECGDSA em Raspberry Pi 3, obtendo ganhos expressivos no tempo de hashing, geração e verificação de assinaturas em comparação com BLAKE2b. Enquanto isso, Bauer et al. (2016) analisam a integridade e autenticidade de ponta-a-ponta com ECDSA e SHA-256, mostrando que otimizações de hardware e uso do protocolo CoAP resultam em comunicação mais robusta em redes instáveis, sem afetar a duração da bateria. Wei, Phuong e Yang (2020) propõem uma versão aprimorada de um esquema de autenticação chamado SAMA, que combina assinaturas em anel e o paradigma offline/online para reduzir o custo de operações críticas.

Já em experimentos focados no ECDSA, Clark e Ali (2023) demonstram que, apesar do overhead de até 300ms no envio de pacotes usando ESP32 e ESP-NOW, a perda de pacotes não é significativamente afetada. Pöhls (2015) introduz o formato JSON Sensor Signatures (JSS) para proteger a integridade as informações na cadeia de processamento de dados IoT de ponta-a-ponta. De forma complementar, Mössinger et al. (2016) avaliam o overhead de bibliotecas ECC no consumo energético, tempo de execução e transmissão de mensagens em dispositivos ARM, mostrando que a biblioteca MicroECC oferece o menor overhead.

4. Desenvolvimento

Neste trabalho, foi desenvolvida uma prova de conceito de certificação digital no dispositivo IoT ESP32, em que utilizou-se algoritmos de assinaturas digitais aceitos pelo ITI (ICP-BRASIL, 2022). Esta prova de conceito se materializou na forma de uma biblioteca chamada CryptoAPI, de forma que ela possa servir como uma interface para

as operações criptográficas implementadas por outras bibliotecas. A CryptoAPI é capaz de realizar: a geração e o armazenamento de um par de chaves criptográficas; a geração e o armazenamento de uma assinatura digital; e a verificação de uma assinatura.

4.1. Escolha das bibliotecas

Para a escolha das bibliotecas criptográficas a serem utilizadas, levou-se em consideração três principais fatores: (1) se a biblioteca é confiável; (2) se ela implementa algum algoritmo de assinatura digital; e (3) se é possível utilizá-la no ESP32 sem precisar portá-la. Considerou-se que bibliotecas confiáveis devem possuir um número próximo ou superior à 1000 estrelas em sua página do GitHub (se houver) ou ter sido utilizada por algum dos artigos revisados. Após diversas pesquisas, as bibliotecas encontradas e consideradas para a análise foram: mbedtls, wolfssl, micro-ecc, arduinolibs, psa-crypto-arduino, TweetNaCl, arduino-crypto e tinyECC. Essas bibliotecas foram avaliadas conforme os critérios de seleção previamente elencados, e as únicas bibliotecas que cumprem os três requisitos elencados são a mbedtls, a wolfssl e a micro-ecc. Portanto, estas foram as três bibliotecas escolhidas para serem utilizadas na CryptoAPI.

4.2. Escolha dos algoritmos

Os algoritmos de assinatura digital a serem utilizados foram escolhidos com base na versão 5.0 do documento “Padrões e algoritmos criptográficos da ICP-Brasil” (ICP-BRASIL, 2022). em que são listados os algoritmos de geração de chaves assimétricas aceitos pelo ITI. Embora o ideal seria conseguir testar todos eles, ficamos limitados pelas implementações disponíveis nas bibliotecas criptográficas escolhidas. Os algoritmos de assinatura digital que foram integrados na CryptoAPI estão listados a seguir:

- ECDSA com as curvas brainpool:
  - brainpool256r1, com chaves de tamanho 256 bits;
  - Brainpool521r1, com chaves de tamanho 512 bits;
- ECDSA com as curvas NIST:
  - secp256r1 (ou P-256), com chaves de tamanho 256 bits;
  - secp521r1 (ou P-521), com chaves de tamanho 521 bits;
- RSA com chaves de tamanho 2048 e 4096 bits;
- Ed25519 com chaves de tamanho 256 bits;
- Ed448 com chaves de tamanho 448 bits.

É importante notar que embora o ITI não aceite as curvas da NIST, como elas estão disponíveis em todas as bibliotecas criptográficas escolhidas, elas foram integradas à CryptoAPI e fizeram parte dos testes, pois é de grande valor conseguir comparar seu desempenho com a dos algoritmos aceitos.

4.3. Testes

O dispositivo utilizado para os testes foi o ESP32, modelo ESP-WROOM-32, que apresenta aceleração de hardware para os algoritmos SHA256, SHA512 e RSA.

Para a realização dos testes, para a maioria dos algoritmos foram realizadas 10 execuções, e calculou-se a média do tempo de execução, consumo de memória e ciclos de clock de cada etapa. Para o algoritmo RSA com chaves de tamanho 4096 bits das bibliotecas MbedTLS e wolfSSL, no entanto, como o tempo de execução possuía alto desvio padrão, realizou-se 20 execuções.

O valor do expoente público do RSA utilizado foi de 65537, uma escolha padrão amplamente adotada em implementações de RSA por seu equilíbrio entre eficiência computacional e segurança, conforme descrito na RFC 8017 (Moriarty et al., 2016). A função de hash utilizada nas execuções dos algoritmos de assinatura digital foi a SHA512, exceto para o algoritmo Ed448, em que deve ser utilizado o algoritmo de hash SHAKE-256, segundo especifica a RFC 8032 (Josefsson; Liusvaara, 2017) documento que define os algoritmos EdDSA. Em todos os testes, a mensagem utilizada para gerar o hash, que foi posteriormente assinado e utilizado na verificação, tem tamanho de 575 bytes.

5. Resultados

Aqui são apresentados e discutidos os resultados obtidos dos testes.

5.1. Geração de chaves

Conforme se observa na Tabela 3, A biblioteca MbedTLS foi a mais rápida em termos de tempo de execução ao usar ECDSA com as curvas NIST, enquanto a wolfSSL foi mais rápida com os outros algoritmos. O algoritmo RSA, mesmo com aceleração de hardware, foi o mais lento, especialmente com chaves de 4096 bits. Em termos de memória, como detalha a Tabela 4, a wolfSSL consumiu muito mais memória que as outras bibliotecas, exceto para algoritmos EdDSA. A micro-ecc foi a biblioteca que menos consumiu memória, e isto provavelmente está relacionado ao fato de que ela usa arrays para guardar as chaves, enquanto que as outras bibliotecas se utilizam de objetos complexos com muitas informações, que requerem mais memória. Por fim, como mostra a Tabela 5, o uso de ciclos de clock seguiu o padrão do desempenho observado: as curvas NIST da MbedTLS exigem menos ciclos que as da wolfSSL, e o RSA continua sendo o mais custoso, com o algoritmo ed25519 sendo o mais eficiente.

Tabela 3. Média do tempo de geração de chaves, em milissegundos

Algoritmo	MbedTLS	wolfSSL	micro-ecc
secp256r1	94,6	530,6	168,1
secp521r1	173,8	1.954,1	-
brainpool256r1	1.231,7	736,1	-
brainpool512r1	3.980,5	2.026,1	-
RSA 2048	6.200,4	5.463,6	-
RSA 4096	43.646,3	37.643,9	-
Ed448	-	106,5	-
Ed25519	-	26,9	-

Tabela 4. Média do consumo de memória para a geração de chaves, em bytes

Algoritmo	MbedTLS	wolfSSL	micro-ecc
secp256r1	715,6	28.784,0	104,0
secp521r1	3.470,0	28.784,0	-
brainpool256r1	946,4	29.852,0	-
brainpool512r1	1.787,2	29.852,0	-
RSA 2048	3.466,0	21.002,6	-
RSA 4096	6.786,2	21.166,2	-
Ed448	-	252,8	-
Ed25519	-	135,2	-

Tabela 5. Média do número de ciclos de clock da geração de chaves, em ciclos

Algoritmo	MbedTLS	wolfSSL	micro-ecc
secp256r1	15.141.233,2	84.908.554,2	26.896.810,5
secp521r1	27.777.557,0	312.663.820,2	-
brainpool256r1	197.061.815,4	117.792.179,2	-
brainpool512r1	636.903.947,4	324.184.226,8	-
RSA 2048	992.075.602,0	874.210.748,5	-
RSA 4096	1.829.431.266,0	1.728.064.595	-
Ed448	-	17.041.338,8	-
Ed25519	-	4.301.285,8	-

5.2. Geração da assinatura

Em relação ao tempo para gerar a assinatura, como mostra a Tabela 6, a biblioteca MbedTLS foi a mais rápida com o ECDSA com curvas NIST, enquanto a wolfSSL foi superior com os outros algoritmos. O ed25519 manteve-se o mais rápido. O RSA da MbedTLS foi mais eficiente do que seu ECDSA com as curvas brainpool, com a aceleração de hardware influenciando seu desempenho. Em termos de consumo de memória, a Tabela 7 nos diz que a biblioteca wolfSSL foi a mais custosa novamente, com exceção dos seus algoritmos EdDSA, em principal com o ed25519, que não apresentou consumo de memória adicional. O RSA da wolfSSL, em particular, consumiu menos memória em comparação com seus outros algoritmos. Quanto aos ciclos de clock, a MbedTLS mostrou melhores resultados com as curvas NIST, enquanto o RSA foi mais eficiente que o ECDSA com a curva brainpool, conforme elucida a Tabela 8.

Tabela 6. Média do tempo para assinar a mensagem, em milissegundos

Algoritmo	MbedTLS	wolfSSL	micro-ecc
secp256r1	108,7	536,5	175,7
secp521r1	207,9	1.967,1	-
brainpool256r1	1.246,0	741,4	-
brainpool512r1	4.010,4	2.035,3	-
RSA 2048	629,6	433,7	-
RSA 4096	2.707,9	2.070,25	-
Ed448	-	113,4	-
Ed25519	-	30,0	-

Tabela 7. Média do consumo de memória para assinar uma mensagem, em bytes

Algoritmo	MbedTLS	wolfSSL	micro-ecc
secp256r1	1.722,8	38.420,0	0
secp521r1	2.893,6	38.420,0	-
brainpool256r1	1.701,2	39.488,0	-
brainpool512r1	2.820,8	39.488,0	-
RSA 2048	6.882,0	13.055,8	-
RSA 4096	13.533,6	11.738,4	-
Ed448	-	252,0	-
Ed25519	-	0	-

Tabela 8. Média do número de ciclos de clock para assinar uma mensagem, em ciclos

Algoritmo	MbedTLS	wolfSSL	micro-ecc
secp256r1	17.408.477,0	85.822.371,2	28.106.546,3
secp521r1	33.258.746,8	314.734.243,6	-
brainpool256r1	199.378.077,6	118.732.881,2	-
brainpool512r1	641.610.814,2	325.638.116,4	-
RSA 2048	100.740.178,5	19.004.084,3	-
RSA 4096	433.269.037,4	331.247.838,6	-

Ed448	-	18.146.413,0	-
Ed25519	-	4.764.296,0	-

5.3.Verificação da assinatura

Na verificação de assinaturas, para a curva secp256r1 a micro-ecc apresentou a implementação mais rápida, como mostra a Tabela 9. Para as curvas brainpool, a wolfSSL foi mais eficiente do que a MbedTLS. O algoritmo RSA da MbedTLS teve o menor tempo de verificação de todos. Em termos de memória, segundo a Tabela 10, a biblioteca wolfSSL continuou a ser a mais custosa novamente, com exceção dos algoritmos EdDSA. O ECDSA da micro-ecc aparenta não ter consumido memória adicional. Para a MbedTLS, o RSA foi o algoritmo que menos consumiu memória. Finalmente, quanto aos ciclos de clock, a Tabela 11 mostra que o ECDSA com a curva brainpool512r1 da MbedTLS foi o menos eficiente, e o RSA da MbedTLS com chaves de 2048 bits foi o mais eficiente. O ed25519 da wolfSSL também se destacou entre os demais algoritmos.

Tabela 9. Média do tempo para verificar uma assinatura, em milissegundos

Algoritmo	MbedTLS	wolfSSL	micro-ecc
secp256r1	376,8	1.021,2	200,2
secp521r1	755,4	3.759,6	-
brainpool256r1	5.001,5	1.424,3	-
brainpool512r1	18.430,3	3.909,1	-
RSA 2048	1,9	118,7	-
RSA 4096	4,0	888,9	-
Ed448	-	286,4	-
Ed25519	-	59,7	-

Tabela 10. Média do consumo de memória para verificar uma assinatura, em bytes

Algoritmo	MbedTLS	wolfSSL	micro-ecc
secp256r1	3.508,8	36.324,0	0
secp521r1	6.116,4	36.324,0	-
brainpool256r1	3.319,2	37.392,0	-
brainpool512r1	6.095,2	37.392,0	-
RSA 2048	1.302,6	7.788,0	-

RSA 4096	2.581,6	8.044,0	-
Ed448	-	4.692,0	-
Ed25519	-	2.364,0	-

Tabela 11. Média do número de ciclos de clock para verificar uma mensagem, em ciclos

Algoritmo	**MbedTLS**	**wolfSSL**	**micro-ecc**
secp256r1	60.286.655,8	163.391.980,0	32.028.084,2
secp521r1	120.867.253,4	120.867.253,4	-
brainpool256r1	800.253.074,4	227.880.363,6	-
brainpool512r1	2.948.825.804,0	625.456.416,2	-
RSA 2048	291.297	19.004.084,3	-
RSA 4096	644.955,9	142.257.788,1	-
Ed448	-	45.786.648,2	-
Ed25519	-	9.565.873,3	-

6. Conclusão

Este trabalho teve como objetivo avaliar a viabilidade de realizar operações de certificação digital no dispositivo IoT ESP32, visando garantir a autenticidade e integridade dos dados em sua transmissão. Inicialmente realizou-se uma revisão sistemática da literatura, que revelou que poucos trabalhos focam exclusivamente na autenticidade e integridade dos dados transmitidos por dispositivos IoT, com a maioria concentrando-se na confidencialidade.

Após isso, foram analisadas diversas bibliotecas criptográficas, das quais três foram escolhidas para integrar a biblioteca CryptoAPI, desenvolvida no estudo. A análise de tempo de execução, consumo de memória e ciclos de clock demonstrou que a realização dessas operações é viável, especialmente com o algoritmo Ed25519 da biblioteca WolfSSL, que apresentou boa performance e baixo consumo de recursos, além de ser aceito pelo ITI.

Com esses resultados, o trabalho contribui para o desenvolvimento de soluções de certificação digital para dispositivos IoT em contextos públicos, permitindo verificar a integridade e autenticidade dos dados. Para trabalhos futuros, propõem-se a adaptação da CryptoAPI para suportar algoritmos pós-quânticos, otimizações para casos específicos e a utilização de certificados digitais dentro dos dispositivos. Outra possibilidade seria a adoção de padrões avançados de assinatura, como o CMS, para garantir maior segurança e conformidade.

Referências

AHMED, Adel A.; BARUKAB, Omar M. Unforgeable Digital Signature Integrated into Lightweight Encryption Based on Effective ECDH for Cybersecurity Mechanism in Internet of Things. *Processes, Multidisciplinary Digital Publishing Institute*, v. 10, n. 12, p. 2631, 2022. Disponível em: https://www.mdpi.com/2227-9717/10/12/2631?type=check_updateversion%3D3. Acesso em: 28 out. 2023.

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. The internet of things: A survey. *Computer networks, Elsevier*, v. 54, n. 15, p. 2787–2805, 2010. Disponível em: https://www.sciencedirect.com/science/article/pii/S1389128610001568. Acesso em: 28 out. 2023.

BAUER, Johannes et al. ECDSA on things: IoT integrity protection in practise. In: SPRINGER. *INFORMATION and Communications Security: 18th International Conference, ICICS 2016, Singapore, Singapore, November 29–December 2, 2016, Proceedings 18*. [S.l.]: Springer, Cham, 2016. p. 3–17. Disponível em: https://link.springer.com/chapter/10.1007/978-3-319-50011-9_1. Acesso em: 26 mai. 2024.

BELLARE, Mihir; ROGAWAY, Phillip. Introduction to modern cryptography. *Lecture Notes*, 2001. Disponível em: https://web.cs.ucdavis.edu/~rogaway/classes/227/fall01/book/main.pdf. Acesso em: 28 out. 2023.

CLARK, Joseph; ALI, Farha. Analysis of ECDSA's Computational Impact on IoT Network Performance. In: *PROCEEDINGS of the 2023 ACM Southeast Conference*. [S.l.: s.n.], 2023. p. 196–200. Disponível em: https://dl.acm.org/doi/abs/10.1145/3564746.3587013. Acesso em: 26 mai. 2024.

DHANDA, Sumit Singh; SINGH, Brahmjit; JINDAL, Poonam. Lightweight cryptography: a solution to secure IoT. *Wireless Personal Communications, Springer*, v. 112, n. 3, p. 1947–1980, 2020. Disponível em: https://link.springer.com/article/10.1007/s11277-020-07134-3. Acesso em: 28 out. 2023.

FAUZAN, Andaresta; SUKARNO, Parman; WARDANA, Aulia Arif. Overhead Analysis of the Use of Digital Signature in MQTT Protocol for Constrained Device in the Internet of Things System. In: *IEEE*. 2020 3rd International Conference on Computer and Informatics Engineering (IC2IE). [S.l.: s.n.], 2020. p. 415–420. Disponível em: https://ieeexplore.ieee.org/abstract/document/9274651. Acesso em: 28 out. 2023.

ICP-BRASIL**.** Padrões e algoritmos criptográficos da ICP-BRASIL. 2022. Disponível em: https://www.gov.br/iti/pt-br/assuntos/legislacao/documentos-principais/IN2022_22_DOC_ICP_01.01_assinado.pdf. Acesso em: 8 out. 2024.

INMETRO**.** Inmetro aprova primeira bomba de combustível com assinatura eletrônica. 2023. Disponível em: https://www.gov.br/inmetro/pt-br/centrais-de-conteudo/noticias/inmetro-aprova-primeira-bomba-de-combustivel-com-assinatura-eletronica. Acesso em: 28 jun. 2024.

JOSEFSSON, S.; LIUSVAARA, I. Edwards-Curve Digital Signature Algorithm (EdDSA). 2017. Disponível em: https://www.rfc-editor.org/rfc/rfc8032#page-43. Acesso em: 10 nov. 2024.

KATZ, Jonathan. Digital signatures. [S.l.]: Springer, 2010. v. 1. Disponível em: https://link.springer.com/book/10.1007/978-0-387-27712-7. Acesso em: 28 out. 2023.

LI, Shancang; XU, Li Da; ZHAO, Shanshan. The internet of things: a survey. *Information systems frontiers, Springer*, v. 17, p. 243–259, 2015. Disponível em: https://link.springer.com/article/10.1007/S10796-014-9492-7. Acesso em: 28 out. 2023.

MORIARTY, K. et al. PKCS 1: RSA Cryptography Specifications Version 2.2. 2016. Disponível em: https://www.rfc-editor.org/rfc/rfc8017.html#appendix-A.1.1. Acesso em: 15 nov. 2024.

MÖSSINGER, Max et al. Towards quantifying the cost of a secure IoT: Overhead and energy consumption of ECC signatures on an ARM-based device. In: *IEEE*. 2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM). [S.l.: s.n.], 2016. p. 1–6. Disponível em: https://ieeexplore.ieee.org/abstract/document/7523559. Acesso em: 4 jun. 2024.

PÖHLS, Henrich C. JSON sensor signatures (JSS): End-to-end integrity protection from constrained device to IoT application. In: *IEEE*. 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. [S.l.: s.n.], 2015. p. 306–312. Disponível em: https://ieeexplore.ieee.org/abstract/document/7284966. Acesso em: 2 jun. 2024.

RAO, Vidya; PREMA, KV. Comparative study of lightweight hashing functions for resource constrained devices of IoT. In: *IEEE*. 2019 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS). [S.l.: s.n.], 2019. p. 1–5. Disponível em: https://ieeexplore.ieee.org/abstract/document/9031038. Acesso em: 27 out. 2023.

WEI, Jiannan; PHUONG, Tran Viet Xuan; YANG, Guomin. An efficient privacy preserving message authentication scheme for internet-of-things. *IEEE Transactions on Industrial Informatics, IEEE*, v. 17, n. 1, p. 617–626, 2020. Disponível em: https://ieeexplore.ieee.org/abstract/document/8988199. Acesso em: 1 jun. 2024.