Universidade Federal de Mato Grosso do Sul
Faculdade de Computação

Pedro Henrique Rocha de Oliveira

Implementação do Algoritmo Criptográfico Ascon em Aplicações Móveis com React Native e TypeScript

Campo Grande - MS
27 de novembro de 2024

Pedro Henrique Rocha de Oliveira

Implementação do Algoritmo Criptográfico Ascon em Aplicações Móveis com React Native e TypeScript

Trabalho de Conclusão de Curso de Ciência da Computação da Faculdade de Computação da Universidade Federal de Mato Grosso do Sul.

Orientador(a): Dra. Ana Karina Dourado Salina de Oliveira

Campo Grande - MS
27 de novembro de 2024

Resumo

Este trabalho apresenta a implementação do algoritmo de criptografia leve Ascon em um ambiente móbile, utilizando TypeScript e React Native. O foco é demonstrar a aplicabilidade prática do Ascon em cenários de segurança, incluindo criptografia autenticada e cálculo de hashes de strings e arquivos. A biblioteca `ascon-js` foi utilizada para viabilizar as operações, enquanto bibliotecas auxiliares permitiram a seleção e leitura de arquivos. Testes de desempenho foram realizados em três dispositivos com diferentes capacidades de hardware, avaliando os tempos de execução das operações de criptografia e hashing. Os resultados mostraram que o Ascon apresenta eficiência em dispositivos com recursos limitados, destacando seu potencial como uma solução viável para criptografia leve em ambientes móveis. Apesar das vantagens do React Native, foram identificadas limitações em termos de desempenho comparado ao código nativo e acesso direto ao hardware. Este trabalho conclui que, embora o desenvolvimento nativo possa ser mais adequado para aplicações que exigem otimizações finas, o Ascon se mostra uma alternativa promissora para segurança em dispositivos móveis.

Sumário

1 Introdução **4**

2 Trabalhos Relacionados **4**

3 Criptografia **5**
3.1 Criptografia leve . . . 5
3.2 Criptografia simétrica . . . 6
3.2.1 Cifras de bloco e crifas de fluxo . . . 6
3.3 Criptografia assimétrica . . . 6
3.4 Funções hash . . . 7

4 Assinatura digital **7**

5 Construção Esponja **8**

6 Ascon **8**
6.1 Authenticated encryption with associated data (AEAD) . . . 9
6.1.1 Inicialização . . . 9
6.1.2 Processamento de dados associados . . . 10
6.1.3 Processamento de texto simples . . . 10
6.1.4 Finalização . . . 11
6.2 Hashing . . . 11
6.2.1 Inicialização . . . 12
6.3 Permutação . . . 13
6.3.1 Adição de constantes . . . 13
6.3.2 Camada de substituição . . . 14
6.3.3 Camada de difusão linear . . . 14

7 Implementação **15**
7.1 React native . . . 15
7.2 Ascon AEAD . . . 15
7.3 Ascon Hash . . . 16
7.4 Hashing de arquivos . . . 17

8 Resultados **17**

9 Limitações **19**

10 Conclusão **20**

1 Introdução

A Segurança da Informação tem se tornado um tema cada vez mais relevante à medida que a sociedade avança para um mundo mais digital. Nesse cenário, considerando o grande volume de troca e armazenamento de dados, a demanda por melhores medidas de proteção aumenta. Uma dessas medidas é a criptografia, que segundo a Kaspersky, é a conversão de dados de um formato legível para um formato codificado com intuito de proteger sua confidencialidade e autenticidade[1]. Essa técnica tem sido poderosa para garantir sistemas mais seguros, assegurando que apenas usuários autenticados e autorizados tenham acesso aos dados enviados.

Para introduzir a criptografia, é necessário o conhecimento geral sobre segurança da informação. A segurança da informação se adapta aos requisitos específicos de cada situação. Independentemente dos agentes envolvidos, todas as partes em uma transação devem ter a certeza de que certos objetivos foram alcançados.[2] Alguns dos objetivos incluem privacidade ou confidencialidade, integridade dos dados, disponibilidade, autenticação ou identificação de entidade, e não repúdio.

À medida que novos dispositivos e tecnologias, como a Internet das Coisas (IoT) e dispositivos móveis, se tornam mais comuns, surge a necessidade de adaptar as técnicas criptográficas para contextos com recursos computacionais limitados. A **criptografia leve** surge como uma solução otimizada, projetada para dispositivos com restrições de processamento, memória e energia. Diferente da criptografia tradicional, amplamente utilizada em servidores robustos, a criptografia leve minimiza o custo computacional e o consumo de energia, sem comprometer a segurança[3].

Este trabalho propõe a implementação do algoritmo de criptografia leve Ascon. Foi implementado a função ASCON Hash e a função de criptografia simétrica AEAD na plataforma React Native, com o objetivo de testar o desempenho dos algoritmos em dispositivos móveis e viabilizar um módulo de geração de hashes de arquivos para os usuários.Nos resultados, mostramos os testes feitos em três dispositivos móveis com configurações diferentes e os tempos de execução em cada um destes dispositivos.

A estrutura deste trabalho está organizada da seguinte forma: a Seção 1 apresenta a introdução ao tema da criptografia e a sua importância na segurança da informação. A Seção 2 discute os trabalhos relacionados na área de criptografia leve. A Seção 3 fornece uma visão geral dos conceitos fundamentais da criptografia, incluindo criptografia simétrica e assimétrica, e a função de hash. A Seção 4 aborda as assinaturas digitais e sua relevância. A Seção 5 descreve a construção esponja e suas aplicações. A Seção 6 explora o algoritmo Ascon, incluindo suas características e implementações. A Seção 7 discute a implementação do Ascon em React Native e suas variantes. A Seção 8 apresenta os resultados obtidos na pesquisa. A Seção 9 aborda as limitações encontradas e, por último, a Seção 10 apresenta as conclusões.

2 Trabalhos Relacionados

Nessa seção, apresentamos uma revisão de trabalhos que abordam o uso de algoritmos de criptografia em dispositivos com recursos limitados, focando na implementação e otimização desses algoritmos em plataformas móveis.

Martins (2024) explora algoritmos de criptografia projetados para dispositivos com recursos limitados em seu estudo intitulado "Encryption algorithms for resource-constrained devices."O autor compara diversas opções, incluindo AES e Ascon, em diferentes plata-

formas de hardware. Os resultados revelam que o Ascon se destaca pela sua eficiência em ambientes com baixo poder de processamento, como o Raspberry Pi Pico, onde apresentou um desempenho superior ao do AES em termos de tempo de execução para operações de criptografia. Essa análise sublinha a importância do Ascon em contextos onde a eficiência energética e a velocidade de processamento são críticas, tornando-o uma escolha adequada para aplicações em dispositivos IoT e sistemas embarcados [4].

Outro estudo relevante é o trabalho de Avery et al. (2022), "Analysis of Practical Application of Lightweight Cryptographic Algorithm ASCON", que examina o uso prático do Ascon em um ambiente IoT. O estudo foca na integração do Ascon ao protocolo de mensagens MQTT, amplamente utilizado para comunicação em dispositivos IoT. A pesquisa demonstra que o Ascon oferece um equilíbrio eficaz entre segurança e desempenho, sendo comparável ao AES em termos de velocidade, porém utilizando menos memória, essencial para dispositivos com restrições de recursos. Este trabalho valida ainda mais a escolha do Ascon para cenários que demandam tanto segurança quanto eficiência em sistemas restritos [5].

Finalmente, o trabalho de Khan et al. (2022), "Evaluating the Performance of Ascon Lightweight Authenticated Encryption for AI-Enabled IoT Devices", avalia o desempenho do algoritmo Ascon em dispositivos IoT habilitados para inteligência artificial. O estudo destaca como Ascon pode ser implementado em plataformas FPGA, que são comuns em aplicações de IoT. Os autores ressaltam a importância da criptografia leve para a segurança dos dados transmitidos por esses dispositivos, mostrando que o Ascon se destaca em cenários restritos em recursos. Esta pesquisa complementa a análise do desempenho do Ascon em dispositivos móveis, evidenciando sua aplicabilidade em uma variedade de contextos de computação restrita [6].

3 Criptografia

A criptografia é uma técnica que protege dados através de algoritmos codificados, hashes e assinaturas, tornando-os indecifráveis para quem não tem a autorização correta [7]. É fundamental que os algoritmos de criptografia garantam a confidencialidade e a integridade dos dados, impedindo que informações sejam acessadas ou alteradas por agentes externos não autorizados.

3.1 Criptografia leve

A criptografia leve é uma subcategoria da criptografia desenvolvida especificamente para dispositivos com recursos limitados, como smartphones, dispositivos IoT e sensores. Tais dispositivos geralmente enfrentam restrições de energia, memória e capacidade de processamento, o que torna as soluções criptográficas convencionais inadequadas. Nesse contexto, surgem algoritmos que equilibram segurança e eficiência de forma otimizada. Um exemplo é o Ascon, projetado para fornecer proteção robusta em dispositivos de baixa potência, com eficiência tanto em hardware quanto em software. Selecionado pelo NIST como o padrão para criptografia leve, o Ascon oferece segurança contra ataques enquanto mantém excelente desempenho em plataformas com capacidade computacional limitada [8].

Desenvolvido em 2014 por uma equipe de pesquisadores para a Caesar Competition, o Ascon foi criado com o objetivo de atender às necessidades de autenticação e criptografia em dispositivos com restrições de recursos. Sua implementação minimiza o uso de

memória e ciclos de clock, tornando-o ideal para ambientes de baixa capacidade computacional[9]. Versátil, o Ascon pode ser usado tanto para criptografia autenticada quanto para a geração de hashes criptográficos. A adoção do Ascon pelo NIST como um dos algoritmos recomendados para criptografia leve reforça sua relevância e eficiência em aplicações como a Internet das Coisas (IoT)[10].

3.2 Criptografia simétrica

A criptografia simétrica, também conhecida como criptografia de chave privada, é um algoritmo que utiliza a mesma chave privada para cifrar e decifrar uma informação. Essa chave é compartilhada entre duas partes que querem se comunicar. Uma parte envia uma mensagem, ou texto simples, à outra parte, utilizando a chave secreta para criptografar a mensagem e obter o texto cifrado que é enviado ao destinatário. O destinatário utiliza a mesma chave para descriptografar o texto cifrado e recuperar a mensagem original[11].

3.2.1 Cifras de bloco e crifas de fluxo

Os sistemas de criptografia são comumente divididos em duas categorias principais: cifras de bloco e cifras de fluxo.

- **Cifras de Bloco:** O cifrador de bloco divide o texto original em blocos de tamanho fixo. Cada bloco é uma sequência de bits, composta por 0's e 1's, com um comprimento predefinido. A cifra de bloco processa cada bloco individualmente, criptografando ou descriptografando um bloco de cada vez.

- **Cifras de fluxo:** O cifrador de fluxo utiliza uma chave para gerar um fluxo de chaves (*keystream*), que é uma sequência de bits com o mesmo comprimento do texto simples de tamanho arbitrário. A cifragem é feita realizando um XOR entre o texto simples e a *keystream*, resultando no texto cifrado. Para reverter, aplica-se novamente a operação XOR entre o texto cifrado e a *keystream*.

3.3 Criptografia assimétrica

A criptografia assimétrica, também conhecida como criptografia de chave pública, é um método de criptografia de dados que usa duas chaves: uma chave pública e uma chave privada. A chave pública pode ser distribuída e a chave privada somente o proprietário tem acesso. O que cifrar com uma das chaves, somente será possível decifrar com a chave correspondente do par. [12]

A ideia da criptografia de chave pública foi concebida na década de 1970 por Diffie e Hellman. A ideia deles era que poderia ser possível criar um sistema criptográfico no qual existem duas chaves distintas. Neste sistema, a chave pública, que pode ser amplamente conhecida, pode ser utilizada para criptografar o texto simples. Por outro lado, a chave privada, que é mantida em segredo pelo seu proprietário, é usada para decifrar o texto cifrado. Portanto, um sistema criptográfico de chave pública permitiria que qualquer pessoa criptografasse uma mensagem a ser transmitida para Bob, digamos, e apenas Bob poderia decifrar a mensagem, obtendo assim, confidencialidade das informações. [13].

3.4 Funções hash

Uma função hash criptográfica é um algoritmo que mapeia uma mensagem de comprimento arbitrário para um resumo de comprimento fixo. Os valores retornados por uma função hash são chamados valores hash ou resumo. A função hash é uma função pública que pode ser conhecida por qualquer pessoa. As funções de hash não podem ser usadas para criptografia, por duas razões fundamentais. Primeiro, o fato de que as funções de hash não possuem uma chave. O segundo é que as funções de hash não podem ser invertidas, pois são funções de caminho único, então um resumo da mensagem não pode ser "descriptografado"[13].

4 Assinatura digital

A assinatura digital é uma forma de garantir a autenticação, integridade e não repúdio dos dados, utilizando funções de hash e criptografia assimétrica. É possível aplicar uma assinatura digital a qualquer tipo de dado, incluindo e-mails, contratos e mensagens[14]. O processo de assinatura digital segue os seguintes passos, conforme ilustrado na Figura 1:

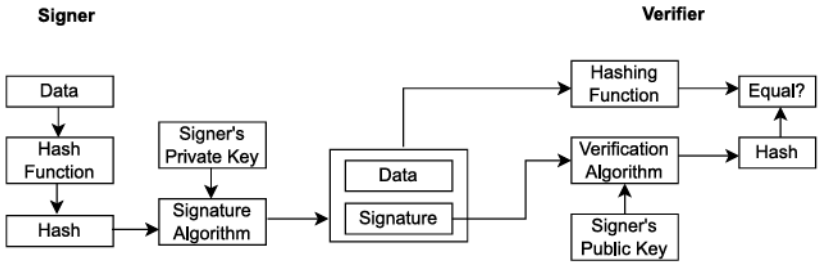

Figura 1: Assinatura digital. *Fonte: Adaptado de Dhar et al (2023).*

1. Gerar um hash dos dados usando uma função de hashing criptográfico.
2. Criptografar o hash gerado utilizando a chave privada do signatário, gerando a assinatura digital dos dados.
3. Enviar os dados e a assinatura para a parte verificadora.
4. Como parte do processo de verificação, a parte verificadora gera o hash dos dados usando a mesma função de hashing criptográfico.
5. Descriptografar a assinatura utilizando a chave pública da parte signatária, resultando em um hash descriptografado que só pode ser gerado pelo signatário.
6. Comparar o hash descriptografado com o hash calculado para determinar se a assinatura digital é válida.

Quando uma chave pública é usada com um certificado de chave pública (ou seja, um certificado que confirma a validade de uma chave pública e contém informações sobre o proprietário da chave), é possível identificar quem assinou o documento[14].

5 Construção Esponja

A função esponja, ou construção esponja, é uma função de mapeamento que recebe uma entrada de tamanho arbitrário e produz uma saída de tamanho igualmente arbitrário. Esse processo utiliza permutações de comprimento fixo e uma regra de preenchimento (padding), como ilustrado na Figura 2.

A construção esponja opera em um estado de $b = r + c$ bits. O valor r é chamado de taxa de bits (*bitrate*) e o valor c é chamado de capacidade.

- **Inicialização:** A string de entrada é preenchida com uma regra de preenchimento (padding) e dividida em blocos de r bits. Em seguida, os b bits do estado são inicializados com zero.
- **Absorção:** Na fase de absorção, uma operação XOR é aplicada entre os blocos de entrada de r bits com os primeiros r bits do estado, seguido pela permutação f.
- **Extração:** Na fase de extração, os primeiros r bits do estado são retornados como blocos de saída, intercalados com a permutação f. O número de blocos de saída é escolhido livremente pelo usuário. Os últimos c bits do estado nunca são diretamente afetados pelos blocos de entrada e nunca são produzidos como saída durante a fase de extração.

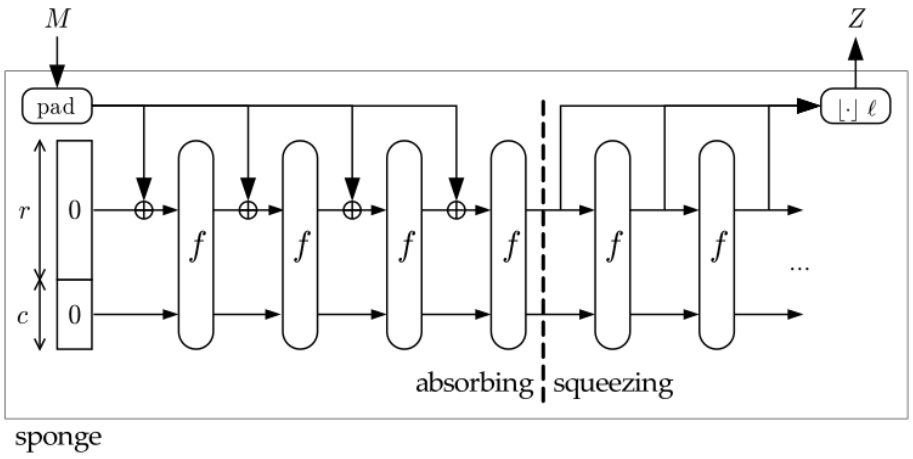

Figura 2: Função esponja. *Fonte: Adaptado de Silva (2016).*

6 Ascon

Ascon é um conjunto de algoritmos para autenticação e hashing projetados para ser leve e de fácil implementação. O algoritmo foi escolhido como opção primária na competição de Algoritmo de Criptografia Leve [15]. Ascon foi projetado para ser seguro e eficiente tanto em software quanto em hardware, ocupando pouco espaço em dispositivos [8].

Inspirado em ferramentas criptográficas amplamente analisadas e padronizadas, Ascon garante robustez e confiabilidade. É ideal para dispositivos leves que precisam realizar operações criptográficas e se comunicar com servidores de alto desempenho. Ascon é

especialmente eficiente para mensagens curtas e foi projetado para resistir a ataques do mundo real, como ataques de canal lateral, garantindo que, mesmo que um invasor consiga acesso a dados internos, isso não comprometerá a chave secreta ou permitirá falsificações sem um esforço significativo [9].

As implementações mais comuns do Ascon incluem Ascon-128, Ascon-128a, Ascon-80pq e Ascon-Hash. A Tabela 1 apresenta os parâmetros dos algoritmos de AEAD, enquanto a Tabela 2 mostra os parâmetros do algoritmo de hash.

Tabela 1: Parâmetros dos Algoritmos de AEAD

Nome	Tamanho de bits de				Rounds	
	Chave	Nonce	Tag	Data Block	p^a	p^b
Ascon-128	128	128	128	64	12	6
Ascon-128a	128	128	128	64	12	8
Ascon-80pq	160	128	128	64	12	6

Tabela 2: Parâmetros do Algoritmo de Hash

Nome	Tamanho do hash(bits)	Datablock	p^a
Ascon-Hash	256	64	12

Todos os esquemas oferecem segurança de 128 bits e utilizam internamente a mesma permutação de 320 bits, com diferentes números de rodadas. A camada de substituição emprega um equivalente de S-box, enquanto a camada de permutação utiliza funções lineares semelhantes às do Algoritmo DES. A permutação resultante é definida em palavras de 64 bits, usando apenas funções booleanas bit a bit (AND, NOT, XOR) e rotações dentro da palavra [9].

6.1 Authenticated encryption with associated data (AEAD)

O Ascon, ao ser utilizado para criptografia autenticada, usa uma chave de até 160 bits. Ele define algoritmos específicos para criptografia $E_{k,r,a,b}$ e descriptografia $D_{k,r,a,b}$. Na criptografia, a entrada inclui uma chave secreta K, um nonce N de 128 bits, dados associados A e texto simples P. A saída é o texto cifrado C e uma tag de autenticação T de 128 bits.

$$E_{k,r,a,b}(K, N, A, P) = (C, T)$$

Na descriptografia, a entrada é a chave secreta, o nonce, os dados associados, o texto cifrado e a tag. A saída é o texto simples se a tag for verificada corretamente, ou um erro $\perp$ se a verificação falhar.

$$D_{k,r,a,b}(K, N, A, C, T) \in \{P, \perp\}.$$

6.1.1 Inicialização

O estado inicial do Ascon possui 320 bits, composto pela chave secreta K de k bits, o nonce de 128 bits e um vetor de inicialização (IV). Este IV é composto pelo tamanho da chave (k), a taxa (r), o número de rodadas de inicialização e finalização (a) e o número de rodadas intermediárias (b).

$$\mathrm{IV}_{k,r,a,b} \leftarrow k\|r\|a\|b\|0^{160-k} = \begin{cases} 80400c0600000000 & \text{para Ascon-128} \\ 80800c0800000000 & \text{para Ascon-128a} \\ a0400c06 & \text{para Ascon-80pq} \end{cases}$$

$$S \leftarrow \mathrm{IV}_{k,r,a,b}\|K\|N$$

Durante a inicialização, a transformação p é aplicada a vezes no estado inicial, seguido por um XOR com a chave secreta

$$S \leftarrow p^a(S) \oplus (0^{320-k}\|K)$$

6.1.2 Processamento de dados associados

O Ascon processa os dados associados A em blocos de r bits. Ele adiciona um único bit 1 e o menor número possível de bits 0 a A para obter um múltiplo de r bits e divide em s blocos de r bits, $A_1|\ldots|A_s$. Caso A esteja vazio, nenhum preenchimento é aplicado e $s = 0$:

$$A_1, \ldots, A_s \leftarrow \begin{cases} \text{blocos de } r \text{ bits de } A\|1\|0^{r-1-(|A| \mod r)} & \text{se } |A| > 0 \\ \emptyset & \text{se } |A| = 0 \end{cases}$$

Cada bloco A_i com $i = 1, \ldots, s$ é combinado utilizando XOR com os primeiros r bits S_r do estado S, seguido pela aplicação da permutação de b rodadas p_b a S:

$$S \leftarrow p^b((S_r \oplus A_i)\|S_c), \quad 1 \le i \le s$$

Após o processamento de A_s (também se $s = 0$), uma constante de 1 bit é combinada por XOR com S:

$$S \leftarrow S \oplus (0^{319}\|1)$$

6.1.3 Processamento de texto simples

O Ascon processa o texto simples P em blocos de r bits. O processo de preenchimento (*padding*) adiciona um único bit 1 e o menor número possível de bits 0 ao texto P para que o comprimento do texto preenchido seja um múltiplo de r bits. O texto preenchido resultante é dividido em t blocos de r bits, $P_1, \ldots, P_t$:

$$P_1, \ldots, P_t \leftarrow \text{blocos de } r \text{ bits de } P\|1\|0^{r-1-(|P| \mod r)}$$

Encriptação Em cada iteração, um bloco de texto plano preenchido P_i com $i = 1, \ldots, t$ é combinado por XOR com os primeiros r bits S_r do estado interno S, seguido pela extração de um bloco de texto cifrado C_i. Para cada bloco, exceto o último, o estado interno S é transformado pela permutação p^b:

$$C_i \leftarrow S_r \oplus P_i$$

$$S \leftarrow \begin{cases} p^b b(C_i\|S_c) & \text{se } 1 \le i < t \\ C_i\|S_c & \text{se } i = t \end{cases}$$

O último bloco de texto cifrado C_t é truncado para o comprimento do último bloco de texto simples não preenchido, de modo que seu comprimento esteja entre 0 e $r-1$ bits, e o comprimento total do texto cifrado C seja exatamente o mesmo que o do texto plano original P:

$$\tilde{C}_t \leftarrow \lfloor C_t \rfloor_{|P| \mod r}$$

Decriptação: Em cada iteração, exceto a última, o bloco de texto simples P_i é calculado realizando um XOR do bloco de texto cifrado C_i com os primeiros r bits S_r do estado interno. Em seguida, os primeiros r bits do estado interno, S_r, são substituídos por C_i. Finalmente, para cada bloco de texto cifrado, exceto o último, o estado interno é transformado pela permutação p_b:

$$P_i \leftarrow S_r \oplus C_i$$

$$S \leftarrow p^b(C_i \| S_c), \quad 1 \leq i < t$$

Para o último bloco de texto cifrado truncado $\tilde{C}_t$ com $0 \leq \ell < r$ bits, o procedimento é diferente:

$$\tilde{P}_t \leftarrow \lfloor S_r \rfloor_\ell \oplus \tilde{C}_t$$

$$S \leftarrow (S_r \oplus (\tilde{P}_t \| 1 \| 0^{r-1-\ell})) \| S_c$$

6.1.4 Finalização

A chave secreta K é combinada utilizando XOR com o estado interno e o estado é transformado pela permutação p^a usando a rodadas. O tag T consiste nos últimos 128 bits do estado combinados por XOR com os últimos 128 bits da chave K:

$$S \leftarrow p^a(S \oplus (0^r \| K \| 0^{c-k}))$$

$$T \leftarrow \lceil S \rceil^{128} \oplus \lceil K \rceil^{128}$$

6.2 Hashing

A função hash é baseada na função esponja e é parametrizada pela taxa r, número de rodadas a, e um limite de comprimento de saída h (h = 0 para saída ilimitada). A função de saída extensível $X_{h,r,a}$ mapeia uma mensagem M para um hash H de comprimento $\ell \leq h$.

$$X_{h,r,a}(M, \ell) = H$$

O Ascon-Hash e o Ascon-Xof usam o mesmo algoritmo, variando os parâmetros: Ascon-Hash com $h = 256$ e Ascon-Xof com $h = 0$. O parâmetro h define o comprimento de H. Assim, chamadas com os mesmos parâmetros e entradas, mas com $\ell' < \ell''$, resultam em H' e H'' com os mesmos valores para os primeiros ℓ' bits [9].

6.2.1 Inicialização

O modo de operação para hashing consiste em uma construção esponja, conforme ilustrado na Figura 3.

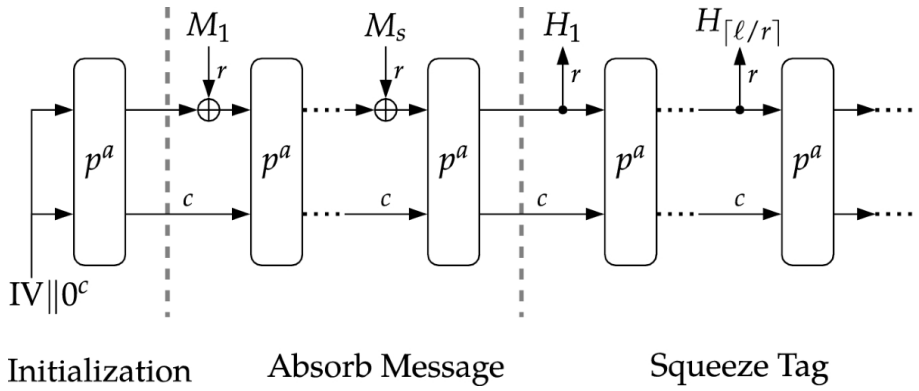

Figura 3: Modo de Hash $X_{h,r,a}$ em Ascon-Hash, Ascon-Xof. *Fonte: Adaptado de Dobraunig et al. (2020).*

Inicialização: O estado inicial de 320 bits é definido por um IV constante que especifica os parâmetros do algoritmo em um formato semelhante ao do Ascon (incluindo $k = 0$, a taxa r e os números de rodadas a e $b = 0$, cada um sendo um inteiro de 8 bits), seguido pelo comprimento máximo de saída de h bits como um inteiro de 32 bits (com $h = 256$ para o Ascon-Hash e $h = 0$ para saída ilimitada no Ascon-Xof) e um valor zero de 256 bits. A permutação de a rodadas p_a é aplicada para inicializar o estado S:

$$IV_{h,r,a} \leftarrow 0^8\|r\|a\|0^8\|h = \begin{cases} 00400c0000000000 \text{ para Ascon-Xof} \\ 00400c0000000100 \text{ para Ascon-Hash} \end{cases}$$

$$S \leftarrow p^a(IV_{h,r,a}\|0^{256})$$

Absorção: A mensagem M é processada em blocos de r bits. O processo de preenchimento é o mesmo que para o texto simples do Ascon: ele adiciona um único bit 1 e o menor número possível de bits 0 a M para que o comprimento da mensagem preenchida seja um múltiplo de r bits. A mensagem preenchida resultante é dividida em s blocos de r bits, $M_1 \ldots M_s$:

$$M_1, \ldots, M_s \leftarrow \text{blocos de } r \text{ bits de } M\|1\|0^{r-1-(|M| \mod r)}$$

Os blocos de mensagem M_i com $i = 1, \ldots, s$ são processados da seguinte forma. Cada bloco M_i é combinado por XOR com os primeiros r bits S_r do estado S, seguido pela aplicação da permutação de p_a a S:

$$S \leftarrow p^a((S_r \oplus M_i)\|S_c), \quad 1 \le i \le s$$

Extração: A saída do hash é extraída do estado em blocos de r bits H_i até que o comprimento de saída solicitado $\ell \le h$ seja completado após $t = \lfloor \ell/r \rfloor$ blocos. Após cada extração, o estado interno S é transformado pela permutação p_a:

$$H_i \leftarrow S_r$$

$$S \leftarrow p^a(S), \quad 1 \leq i \leq t = \lfloor \ell / r \rfloor$$

O último bloco de saída H_t é truncado para $\ell \mod r$ bits (a menos que r seja divisor de ℓ) e $H = H_1 \| \ldots \| \tilde{H}_t$ é retornado:

$$\tilde{H}_t \leftarrow \lfloor H_t \rfloor_{\ell \mod r}$$

6.3 Permutação

A principal operação que ocorre durante a execução do Ascon, em ambas as variações, são as permutações de 320 bits p^a e p^b. Essas permutações, por sua vez, consistem em três etapas: p_C, p_S e p_L:

$$p = p_L \circ p_S \circ p_C$$

- p_C **(Adição de Constantes)**: Esta etapa envolve a adição de constantes à entrada, o que ajuda a garantir que cada permutação tenha um comportamento único. A inclusão de constantes é fundamental para a segurança, pois impede que padrões repetitivos se formem, dificultando a análise de ataques.

- p_S **(Camada de Substituição)**: A etapa p_S aplica uma substituição não linear aos dados, utilizando uma função de mapeamento que transforma os bits de forma imprevisível. Essa substituição é crucial para aumentar a segurança do algoritmo, pois impede a identificação de padrões simples na saída, dificultando a análise criptográfica.

- p_L **(Camada de Difusão Linear)**: Por fim, a operação p_L realiza uma transformação linear que busca garantir uma maior difusão dos bits. Esta camada combina os dados de maneira linear, espalhando a influência de cada bit na saída final. A difusão linear é vital para balancear a relação entre entradas e saídas, reforçando a segurança do algoritmo.

p^a e p^b diferem apenas no número de rodadas. O número de rodadas a e o número de rodadas b são parâmetros ajustáveis.

Para a aplicação das transformações, o estado de 320 bits S é dividido em cinco palavras de 64 bits x_i:

$$S = x_0 \| x_1 \| x_2 \| x_3 \| x_4$$

6.3.1 Adição de constantes

A etapa de adição de constante p_C adiciona uma constante de rodada c_r à palavra de registro x_2 do estado S na rodada i (veja Tabela 3). Ambos os índices r e i começam do zero e usamos $r = i$ para p_a e $r = i + a - b$ para p_b:

$$x_2 \leftarrow x_2 \oplus c_r$$

p^{12}	p^8	p^6	Constante c_r	p^{12}	p^8	p^6	Constante c_r
0			00000000000000f0	6	2	0	0000000000000096
1			00000000000000e1	7	3	1	0000000000000087
2			00000000000000d2	8	4	2	0000000000000078
3			00000000000000c3	9	5	3	0000000000000069
4	0		00000000000000b4	10	6	4	000000000000005a
5	1		00000000000000a5	11	7	5	000000000000004b

Tabela 3: Constantes de rodada c_r usadas em cada rodada i de p^a e p^b. *Fonte: Adaptado de Dobraunig et al. (2020).*

6.3.2 Camada de substituição

A camada de substituição p_S atualiza o estado S com 64 aplicações paralelas da S-box de 5 bits $S(x)$ a cada bit-slice dos cinco registradores $x_0 \dots x_4$ definida na Figura 4.

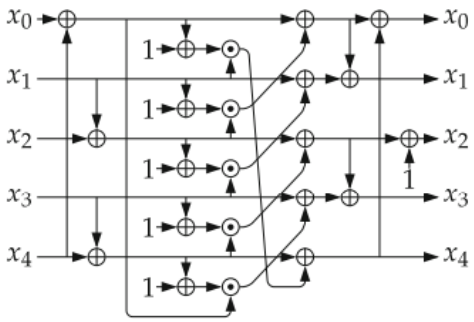

Figura 4: Ascon 5-bit S-box S. *Fonte: Adaptado de Dobraunig et al. (2020).*

6.3.3 Camada de difusão linear

A camada de difusão linear p_L é aplicada dentro de cada palavra de registro de 64 bits x_i. Geralmente, a camada de difusão linear é implementada conforme descrito na Figura 6. Ela aplica uma função linear $\Sigma_i(x_i)$ a cada palavra x_i definida na Figura 5.

$$x_i \leftarrow \Sigma_i(x_i), \quad 0 \leq i \leq 4.$$

Figura 5: Camada de difusão linear do ASCON com as funções de 64 bits $\Sigma_i(x_i)$. *Fonte: Adaptado de Dobraunig et al. (2020).*

$$\begin{aligned}
x_0 &\leftarrow \Sigma_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \\
x_1 &\leftarrow \Sigma_1(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \\
x_2 &\leftarrow \Sigma_2(x_2) = x_2 \oplus (x_2 \ggg \ 1) \oplus (x_2 \ggg \ 6) \\
x_3 &\leftarrow \Sigma_3(x_3) = x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17) \\
x_4 &\leftarrow \Sigma_4(x_4) = x_4 \oplus (x_4 \ggg \ 7) \oplus (x_4 \ggg 41)
\end{aligned}$$

```
x0 ^= x4;    x4 ^= x3;    x2 ^= x1;
t0  = x0;    t1  = x1;    t2  = x2;    t3  = x3;    t4  = x4;
t0 = ~t0;    t1 = ~t1;    t2 = ~t2;    t3 = ~t3;    t4 = ~t4;
t0 &= x1;    t1 &= x2;    t2 &= x3;    t3 &= x4;    t4 &= x0;
x0 ^= t1;    x1 ^= t2;    x2 ^= t3;    x3 ^= t4;    x4 ^= t0;
x1 ^= x0;    x0 ^= x4;    x3 ^= x2;    x2 = ~x2;
```

Figura 6: Instruções em Pipeline para implementação bitsliced da S-box de 5 bits $S(x)$. *Fonte: Adaptado de Dobraunig et al. (2020).*

7 Implementação

Neste trabalho, foi implementado o algoritmo criptográfico Ascon em um ambiente móbile, utilizando TypeScript, React Native e a biblioteca `ascon-js` para a plataforma Android. O Ascon foi aplicado em diferentes cenários de segurança de dados, como criptografia e autenticação de mensagens, cálculo de hashes de strings e arquivos. As implementações foram realizadas com foco em demonstrar a aplicabilidade prática do Ascon em dispositivos móveis. A biblioteca `ascon-js` utilizada na implementação foi recomendada e publicada no site oficial do Ascon[16], mantido pelos criadores do Ascon. Bibliotecas adicionais, como `crypto` para geração de chaves e nonces, `react-native-document-picker` para a seleção de arquivos e `rn-fetch-blob` para leitura de arquivos, foram utilizadas para auxiliar no desenvolvimento do aplicativo.

7.1 React native

React Native é um framework de código aberto criado pela Meta Platforms, Inc. que permite a criação de aplicativos nativos para plataformas iOS e Android utilizando JavaScript e React. Diferente dos paradigmas de desenvolvimento tradicionais de desenvolvimento móvel, que exigem o uso de linguagens específicas da plataforma como Swift ou Kotlin, o React Native possibilita o uso do mesmo código base JavaScript para ambas as plataformas[17]. Seu funcionamento consiste no código JavaScript sendo executado em máquina virtual e interage com componentes nativos do sistema operacional.

7.2 Ascon AEAD

No componente `AeadScreen`, foi implementada uma função que executa operações de criptografia de 1 milhão de mensagens únicas para avaliar a eficiência do Ascon em dispositivos móveis. A função `executeAeadHash`, conforme mostrado no Código 1, utiliza a biblioteca `crypto` para gerar aleatoriamente chaves e nonces. As mensagens e os dados associados são convertidos em vetores de inteiros sem sinal de 8 bits utilizando `TextEncoder`.

```
1  const n = 1000000;
2  const [totalTime, setTotalTime] = React.useState<number | null>(null);
3  const [avgTime, setAvgTime] = React.useState<number | null>(null);
4
5  function executeAeadHash() {
6    let time = 0;
7
8    for (let i = 0; i < n; i++) {
```

```
      const key = crypto.randomBytes(16);
      const nonce = crypto.randomBytes(16);
      const plaintext = new TextEncoder().encode('ascon' + i);
      const associatedData = new TextEncoder().encode('associated');

      const start = performance.now();
      Ascon.encrypt(key, nonce, plaintext, {
        associatedData,
      });
      const end = performance.now();
      time += end - start;
    }
    setTotalTime(time);
    setAvgTime(time / n);
  }
```

Listing 1: Código da função `executeAeadHash` em `AeadScreen`

Durante a execução, o Ascon realiza a criptografia dos dados com a função `Ascon.encrypt()`, utilizando as chaves, nonces e dados associados. O tempo total de execução é medido e dividido pelo número de operações, fornecendo uma média de tempo para criptografia de uma única mensagem. O tempo médio de execução é exibido na interface do usuário.

7.3 Ascon Hash

Além do algoritmo AEAD, foi implementado o cálculo de hashes de strings utilizando o Ascon. No componente `HashScreen`, foi implementado o cálculo de hashes utilizando o Ascon. A função `executeAsconHash`, mostrada no Código 2, calcula o hash de 1 milhão de strings únicas, convertidas para um vetor de inteiros sem sinal de 8 bits por meio do `TextEncoder`.

```
  const n = 1000000;
  const [totalTime, setTotalTime] = React.useState<number | null>(null);
  const [avgTime, setAvgTime] = React.useState<number | null>(null);

  function executeAsconHash() {
    let time = 0;
    for (let i = 0; i < n; i++) {
      const message = new TextEncoder().encode('ascon' + i);
      const start = performance.now();
      Ascon.hash(message);
      const end = performance.now();
      time += end - start;
    }
    setTotalTime(time);
    setAvgTime(time / n);
  }
```

Listing 2: Código da função `executeAsconHash` em `HashScreen`

O cálculo do hash é realizado pela função `Ascon.hash()`, e o tempo médio de execução é calculado dividindo o tempo total pelo número de mensagens processadas. O tempo médio de execução é exibido na interface do usuário.

7.4 Hashing de arquivos

Finalmente, foi desenvolvida uma funcionalidade para calcular o hash de arquivos. Esta funcionalidade permite ao usuário selecionar um arquivo do dispositivo e calcular seu hash utilizando o Ascon Hash. No componente `FileScreen`, conforme demonstrado no Código 3, a API `react-native-document-picker` é utilizada para selecionar arquivos, e a biblioteca `rn-fetch-blob` é usada para ler o conteúdo do arquivo. Após a leitura, a função `executeAsconHashFile` calcula o hash do conteúdo do arquivo.

```
1  const [file, setFile] = React.useState<string | null>(null);
2  const [result, setResult] = React.useState<Uint8Array | null>(null);
3
4  async function pickDocument() {
5    const document: DocumentPickerResponse = await Picker.pickSingle();
6    const fileString = await RNFetchBlob.fs.readFile(document.uri, 'utf8'
     );
7    setFile(fileString);
8  }
9
10 function executeAsconHashFile() {
11   const arquivo = new TextEncoder().encode(file!);
12   setResult(Ascon.hash(arquivo));
13 }
```

Listing 3: Código da função `executeAsconHashFile` em `FileScreen`

O resultado do hash é convertido para um formato hexadecimal utilizando a biblioteca `buffer` e exibido na interface da aplicação. Essa abordagem mostra a flexibilidade do Ascon em ser aplicado não apenas em mensagens de texto, mas também em arquivos, demonstrando sua viabilidade em cenários reais de segurança em dispositivos móveis.

8 Resultados

Para avaliar o desempenho da implementação do algoritmo Ascon em dispositivos móveis, foram realizados testes em três dispositivos distintos, com foco na medição do tempo de execução das operações de criptografia autenticada (AEAD) e hashing. Esses testes permitem analisar a eficiência do Ascon tanto na criptografia de grandes volumes de dados quanto na geração de hashes, levando em consideração diferentes níveis de capacidade de processamento dos dispositivos. Para garantir a precisão dos resultados, cada operação foi repetida um milhão de vezes, e os tempos médios de execução foram calculados, fornecendo uma métrica confiável do desempenho do algoritmo em diversos cenários.

O **Dispositivo 1** é equipado com um processador Snapdragon 778G, que possui 8 núcleos (4 de alto desempenho e 4 de eficiência), 6 GB de memória RAM e um clock máximo de 2,4 GHz. O **Dispositivo 2** possui um processador Snapdragon 730G, com 8 núcleos (2 de alto desempenho e 6 de eficiência), 4 GB de memória RAM e um clock

máximo de 2,2 GHz. Por fim, o **Dispositivo 3** utiliza um processador Snapdragon 632, com 8 núcleos (4 de alto desempenho e 4 de eficiência), 3 GB de memória RAM e um clock máximo de 1,8 GHz. Além disso, foram realizados testes em um dispositivo ESP32, que possui um processador de 240 MHz e 520 KB de SRAM, utilizando a implementação otimizada em C para dispositivos ESP32, desenvolvida pela equipe criadora do Ascon[18].

Os parâmetros empregados na implementação do algoritmo Ascon seguem os padrões estabelecidos em sua especificação. No modo de operação AEAD, foram utilizadas uma chave e um nonce de 128 bits, além de 12 rodadas de permutação a e 6 rodadas de permutação b. Para as funções de hashing, o comprimento de saída é de 256 bits, com os dados processados em blocos de 64 bits e utilizando 12 rodadas de permutação a.

Teste	ESP32	Dispositivo 1	Dispositivo 2	Dispositivo 3
AEAD média (µs)	43	756	1154	2260
AEAD máximo (µs)	-	2196	5867	7603
AEAD mínimo (µs)	-	735	986	1933
Hash média (µs)	46	1179	1632	3181
Hash máximo (µs)	-	2477	6164	8283
Hash mínimo (µs)	-	996	1350	2667

Tabela 4: Resultados de Testes

Conforme mostrado na Tabela 4, os tempos de execução variaram significativamente entre os dispositivos, principalmente devido às suas diferenças em capacidade de processamento e memória. A implementação do Ascon utilizada nos dispositivos móveis foi a ascon-js. O Dispositivo 1, com processador Snapdragon 778G e 6 GB de RAM, apresentou os melhores resultados, com tempos médios de 756µs para AEAD e 1179µs para hashing. Isso se deve ao seu processador mais rápido e maior quantidade de memória, que permitem uma execução mais eficiente. O Dispositivo 2, com um Snapdragon 730G e 4 GB de RAM, teve desempenho intermediário, com 1154µs para AEAD e 1632µs para hashing. Por outro lado, o Dispositivo 3, com um processador Snapdragon 632 e 3 GB de RAM, obteve os piores tempos, com 2260mµs para AEAD e 3181µs para hashing, refletindo seu menor poder de processamento e memória limitada. Esses resultados confirmam que dispositivos com mais recursos oferecem maior eficiência na execução do algoritmo Ascon, destacando a influência da capacidade de hardware no desempenho de operações criptográficas.

A implementação do Ascon no ESP32 foi realizada em C, com otimizações especificas para o dispostivo em questão. Os resultados mostraram tempos de execução para essa implementação é de 43µs para AEAD e 46µs para hashing, evidenciando que, mesmo com um hardware limitado, o Ascon mantém um desempenho competitivo. Esses resultados ressaltam a eficiência do algoritmo em diferentes plataformas e a sua capacidade de adaptação a ambientes com recursos restritos.

No estudo realizado por Martins (2024), foi demonstrado que o Ascon pode alcançar tempos de execução significativamente mais baixos, como os encontrados no ESP32, em plataformas de hardware de baixo consumo, como o Raspberry Pi Pico, onde o Ascon obteve um tempo de execução de apenas 40 µs para dados de 16 bytes. Comparativamente, os algoritmos AES-128, AES-192 e AES-256 exibiram tempos mais altos, de 56 µs, 66 µs e 76 µs, respectivamente. Esse desempenho superior do Ascon reforça seu potencial em ambientes de recursos limitados e destaca a eficiência do algoritmo em comparação a alternativas amplamente utilizadas [4].

Os resultados obtidos neste estudo, entretanto, indicam que as implementações do Ascon em ambientes móveis utilizando React Native, enfrentam limitações que afetam seu desempenho. Os tempos de execução em dispositivos móveis foram muito superiores aos do estudo de Martins, possivelmente devido à sobrecarga da camada de abstração e às limitações de otimização de hardware do React Native. Esses achados sugerem que, para cenários de aplicação em que o tempo de execução é um fator crítico, pode ser vantajoso explorar alternativas de implementação nativa para melhor utilizar as capacidades do dispositivo.

9 Limitações

Apesar das vantagens do React Native em termos de rapidez de desenvolvimento e reutilização de código, existem algumas limitações inerentes à sua arquitetura que podem impactar o desempenho:

- **Desempenho Comparado a Código Nativo**: Mesmo com a utilização de motores como Hermes, o desempenho de aplicativos React Native tende a ser inferior ao de aplicações escritas diretamente em código nativo, como Kotlin. Isso ocorre porque a camada de abstração entre o código JavaScript e o código nativo pode introduzir overhead [19].

- **Limitada Interação com Recursos Nativos**: Algumas operações específicas que dependem diretamente de APIs nativas podem ser mais difíceis de implementar em React Native, uma vez que pode ser necessário escrever módulos nativos para acessar funcionalidades indisponíveis [20].

- **Ausência de Acesso Direto ao Hardware**: No React Native, a camada de abstração do JavaScript impede o acesso direto ao hardware do dispositivo, dificultando a medição precisa dos ciclos de CPU. Além disso, a variação entre dispositivos e sistemas operacionais adiciona mais complexidade a essa tarefa. [21]

- **Thread Única para JavaScript**: O JavaScript no React Native é executado em uma única thread, o que significa que operações computacionalmente intensivas podem bloquear a interface do usuário e causar problemas de desempenho, como atrasos na renderização. Isso limita a execução de operações computacionalmente intensivas [22].

- **Incompatibilidade com Bibliotecas do Node.js**: O React Native não oferece suporte nativo para bibliotecas Node.js, como o crypto, que é necessária para o funcionamento do ascon-js. Para superar essa limitação, é preciso utilizar pacotes como o rn-nodify, que simulam o ambiente Node.js no React Native. No entanto, essa solução pode introduzir uma degradação no desempenho, já que o rn-nodify não consegue replicar a eficiência do código nativo Node.js em termos de criptografia e outras operações computacionalmente intensivas. [23]

10 Conclusão

A segurança e a privacidade das informações em aplicações que manipulam dados sensíveis são de extrema importância, especialmente em conformidade com a Lei Geral de Proteção de Dados (LGPD), que estabelece diretrizes rigorosas para a coleta, armazenamento e tratamento dessas informações. A implementação de mecanismos robustos de proteção é essencial para garantir a integridade e a confidencialidade dos dados dos usuários.

Nesse contexto, a criptografia torna-se uma ferramenta fundamental. Tecnologias de criptografia leve, como o Ascon, foram projetadas para fornecer segurança sem comprometer a eficiência em dispositivos com recursos limitados. O Ascon se apresenta como uma alternativa viável, assegurando a privacidade e a proteção das informações, ao mesmo tempo que mantém um desempenho adequado em ambientes móveis, alinhando-se às exigências de segurança impostas pela LGPD.

Dessa forma, a análise dos resultados validou a performance do algoritmo Ascon em um ambiente móvel usando React Native, mostrando que, embora seja eficiente para operações de criptografia autenticada e hashing, a implementação enfrenta limitações de desempenho em comparação com plataformas de baixo consumo analisadas em outros estudos. Os testes realizados demonstraram que o Ascon, implementado com a biblioteca `ascon-js` em um aplicativo React Native, apresenta tempos de execução superiores ao esperado, especialmente ao considerar resultados de estudos como o de Martins (2024), no qual o Ascon demonstrou maior eficiência em plataformas com hardware limitado.

Apesar das vantagens do React Native para o desenvolvimento multiplataforma, seus recursos limitados para otimização de desempenho e integração direta com o hardware comprometem a eficiência do Ascon em dispositivos móveis. Para aplicações que exigem máximo desempenho e acesso otimizado ao hardware, uma abordagem nativa poderia proporcionar resultados mais favoráveis. Ainda assim, este trabalho evidenciou a aplicabilidade do Ascon como uma solução promissora de criptografia leve em ambientes de recursos restritos, com potencial para evoluir à medida que as ferramentas de desenvolvimento móvel se aprimoram.

Para trabalhos futuros, propõe-se investigar a implementação do Ascon em plataformas nativas, como Kotlin/Java para Android ou Swift para iOS, comparando sua eficiência com o React Native. Além disso, sugere-se estudos sobre a otimização de bibliotecas criptográficas em ferramentas multiplataforma, visando aprimorar o suporte a algoritmos leves em dispositivos móveis.

Referências

[1] Kaspersky. *O que é criptografia de dados? Definição e explicação*. Acesso em 18 de abril de 2024. URL: https://www.kaspersky.com.br/resource-center/definitions/encryption.

[2] Alfred J. Menezes, Paul C. van Oorschot e Scott A. Vanstone. *Handbook of Applied Cryptography (Discrete Mathematics and Its Applications)*. CRC Press, 1996.

[3] D. Knuth. "An Introduction to Lightweight Cryptography". Em: *Journal of Cryptographic Research* 23.4 (2015), pp. 113–126.

[4] Pedro Afonso Ferreira Lopes Martins. "Encryption algorithms for resource-constrained devices". Diss. de mestr. DEI - FCTUC: Faculdade de Ciências e Tecnologia, jul. de 2024.

[5] Jeffrey Avery et al. "Analysis of Practical Application of Lightweight Cryptographic Algorithm ASCON". Em: (2022).

[6] Safiullah Khan, Wai-Kong Lee e Seong Oun Hwang. "Evaluating the Performance of Ascon Lightweight Authenticated Encryption for AI-Enabled IoT Devices". Em: *2022 TRON Symposium (TRONSHOW)*. 2022.

[7] IBM. *What is cryptography?* Acesso em 18 de abril de 2024. URL: https://www.ibm.com/topics/cryptography.

[8] *Ascon - Lightweight Authenticated Encryption and Hashing*. Acesso em 20 de junho de 2024. URL: https://ascon.iaik.tugraz.at.

[9] Christoph Dobraunig et al. "ASCON v1.2: Lightweight Authenticated Encryption and Hashing". Em: *Journal of Cryptology* (2020).

[10] National Institute of Standards e Technology (NIST). *Lightweight Cryptography Standardization Process: Round 3*. Rel. técn. Available at https://csrc.nist.gov/projects/lightweight-cryptography. NIST, 2021.

[11] Jonathan Katz e Yehuda Lindell. *Introduction to Modern Cryptography 2nd Edition*. Chapman e Hall/CRC, 2014.

[12] *Asymmetric encryption*. Acesso em 13 de abril de 2024. URL: https://encyclopedia.kaspersky.com/glossary/asymmetric-encryption/.

[13] Douglas R. Stinson e Maura B. Paterson. *Cryptography : theory and practice*. Chapman e Hall/CRC, 2018.

[14] Valentin Mulder et al. *Trends in Data Protection and Encryption Technologies*. Springer, 2023.

[15] *The CAESAR committee, CAESAR: competition for authenticated encryption: security, applicability and robustness (2014)*. URL: https://competitions.cr.yp.to/caesar-submissions.html.

[16] *Ascon - Implementations*. Acesso em 18 de abril de 2024. URL: https://ascon.iaik.tugraz.at/implementations.html.

[17] "React Native". Em: (). Acesso em 26 de outubro de 2024. URL: https://reactnative.dev.

[18] *Ascon C*. Acesso em 26 de outubro de 2024. URL: https://github.com/ascon/ascon-c.

[19] Oskar Svensson e Marcus Presa Käld. “React Native and native application development”. Em: (2021).

[20] *Native Modules Intro*. Acesso em 02 de outubro de 2024. URL: https://reactnative.dev/docs/native-modules-intro.

[21] *Performance*. Acesso em 02 de outubro de 2024. URL: https://archive.reactnative.dev/docs/performance.

[22] *Threading Model*. Acesso em 02 de outubro de 2024. URL: https://reactnative.dev/architecture/threading-model.

[23] Tradle. *rn-nodeify*. Acesso em 21 de outubro de 2024. 2024. URL: https://github.com/tradle/rn-nodeify?tab=readme-ov-file%5C#what-is-solves.