CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA FACULDADE DE TECNOLOGIA DE MAUÁ

FACULDADE DE TECNOLOGIA DE MAUÁ

CURSO DE INFORMÁTICA PARA NEGÓCIOS

JAILTON MENDES OLIVEIRA

VERIFICAÇÃO DA SEGURANÇA DA CRIPTOGRAFIA AES E RSA EM RELAÇÃO AO TAMANHO DAS CHAVES

MAUÁ
2024

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA FACULDADE DE TECNOLOGIA DE MAUÁ

VERIFICAÇÃO DA SEGURANÇA DA CRIPTOGRAFIA AES E RSA EM RELAÇÃO AO TAMANHO DAS CHAVES

Trabalho de Conclusão de Curso (TCC) apresentado à FATEC Mauá, como parte dos requisitos para obtenção do Título de Tecnólogo em Informática para Negócios.

Orientador: Prof. Me. Ivan Carlos Pavão.

Ms. Orientador : Prof. Me. Ivan Carlos Pavão.

MAUÁ - SP
2024

Catalogação-na-Publicação – Biblioteca Fatec Mauá

005.82
O48v Oliveira, Jailton Mendes.
Verificação da segurança da criptografia AES e RSA em relação ao tamanho das chaves / Jailton Mendes Oliveira. – 2024.
65 p. : il. ; 30 cm.

Orientador: Prof. Me. Ivan Carlos Pavão.
Trabalho de conclusão de curso (Curso Superior de Tecnologia em Informática para Negócios) – Faculdade de Tecnologia de Mauá.
Referências: p. 61.

1. Criptografia. 2. AES (*Advanced Encryption Standard*). 3. RSA (*Rivest-Shamir-Adleman*). 4. Segurança. 5. Cifra. I. Pavão, Ivan Carlos. II. Título.

CDD 23. : Criptografia de dados (informática) 005.82
Criptografia de dados (tecnologia) 652.8
Elaborada por Tatiana Sambinelli CRB-8 SP-011003/O

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA FACULDADE DE TECNOLOGIA DE MAUÁ

VERIFICAÇÃO DA SEGURANÇA DA CRIPTOGRAFIA AES E RSA EM RELAÇÃO AO TAMANHO DAS CHAVES

Trabalho de Conclusão de Curso (TCC) apresentado à FATEC Mauá, como parte dos requisitos para obtenção do Título de Tecnólogo em Informática para Negócios.

Aprovação em: __/__/ 2024.

Prof. Me. Ivan Carlos Pavão
FATEC Mauá
Orientador (a)

Prof. João Carlos de Souza
FATEC Mauá
Avaliador

Prof. (a). Luana Lourenço
FATEC Mauá
Avaliadora

AGRADECIMENTOS

Desejo expressar meus agradecimentos.

Ao Professor e orientador Ivan Carlos Pavão, pela dedicação, paciência e incentivo para transformar minhas preocupações em pesquisa.

Aos professores da Graduação que me abriram horizontes para conclusão dessa etapa de estudos.

E, por fim, aos meus amigos de sala, pois, buscamos os mesmos objetivos.

RESUMO

O objetivo principal deste estudo é verificar a resistência dos algoritmos criptográficos AES (Advanced Encryption Standard) e RSA (Rivest-Shamir-Adleman) para diferentes tamanhos de chave para a ataques de força bruta, com o objetivo de investigar a viabilidade desses ataques com base no poder computacional atual. Esta pesquisa é de natureza empírica e foi desenvolvida por meio de estudo de caso. A relevância do estudo reside na crescente necessidade de assegurar a segurança da informação diante do avanço tecnológico e da capacidade computacional, que constantemente desafiam os padrões de criptografia existentes. Para atingir os objetivos propostos, a pesquisa foi estruturada em uma metodologia que envolveu a implementação de algoritmos em JavaScript para testar a resistência do AES e RSA. O processo experimental consistiu na geração de chaves criptográficas de diferentes tamanhos e na medição do tempo necessário para testar essas chaves. Foram realizadas várias rodadas de testes com tamanhos de chave variados, tanto para o AES quanto para o RSA, registrando-se o tempo de execução em milissegundos. Este método permitiu uma análise comparativa detalhada entre os dois algoritmos, destacando suas respectivas resistências a ataques de força bruta. Notou-se que os resultados obtidos indicam que o AES e o RSA apresentam uma resistência significativa a ataques de força bruta, especialmente quando se utilizam chaves de tamanhos maiores. No caso do AES, as chaves de 256 bits demonstraram uma robustez considerável, com tempos de teste relativamente altos, o que torna inviável a quebra do algoritmo por força bruta dentro de um prazo razoável com o poder computacional disponível atualmente. O RSA, por sua vez, também apresentou forte resistência, especialmente com chaves de 2048 bits, mostrado tempos de geração e teste de chaves mais elevados em comparação ao AES. A comparação entre os algoritmos revelou que o AES tende a ser mais eficiente em termos de tempo de execução para chaves de tamanho equivalente, o que reforça sua aplicabilidade em sistemas que requerem maior velocidade e eficiência. As conclusões deste trabalho destacam a inviabilidade prática de ataques de força bruta contra ambos os algoritmos com as configurações de chave atuais, contribuindo para a validação contínua do uso dessas técnicas em segurança da informação. As limitações do estudo incluem as configurações do computador usado para rodar os testes, que não representavam o poder computacional mais avançado disponível, e o próprio algoritmo de teste utilizado. Sugere-se que futuras

pesquisas explorem o impacto de avanços em hardware e técnicas de otimização para continuar avaliando a segurança dos algoritmos AES e RSA em cenários cada vez mais desafiadores.
Palavras chaves: criptografia; AES; RSA; segurança; cifra.

ABSTRACT:

The main objective of this study is to verify the resistance of the cryptographic algorithms AES (Advanced Encryption Standard) and RSA (Rivest-Shamir-Adleman) to different key sizes against brute-force attacks, aiming to investigate the feasibility of these attacks based on current computational power. This research is empirical in nature and was developed through a case study approach. The relevance of the study lies in the growing need to ensure information security in the face of technological advancement and computational capacity, which constantly challenge existing encryption standards. To achieve the proposed objectives, the research was structured using a methodology involving the implementation of algorithms in JavaScript to evaluate the resistance of AES and RSA. The experimental process involved generating cryptographic keys of varied sizes and measuring the time required to test these keys. Several rounds of tests were conducted with varying key sizes for both AES and RSA, recording the execution time in milliseconds. This method allowed for a detailed comparative analysis between the two algorithms, highlighting their respective resistance to brute-force attacks. It was observed that the results indicate AES and RSA exhibit significant resistance to brute-force attacks, especially when using larger key sizes. In the case of AES, 256-bit keys demonstrated considerable robustness, with high-test times, making it impractical to break the algorithm by brute force within a reasonable period with currently available computational power. RSA, on the other hand, also showed strong resistance, especially with 2048-bit keys, exhibiting higher key generation and testing times compared to AES. The comparison between the algorithms revealed that AES tends to be more efficient in terms of execution time for equivalent key sizes, reinforcing its applicability in systems requiring greater speed and efficiency. The conclusions of this work highlight the practical infeasibility of brute-force attacks against both algorithms with current key configurations, contributing to the ongoing validation of the use of these techniques in information security. Study limitations include the computer settings used to run the tests, which did not represent the most advanced computational power available, and the test algorithm itself. It is suggested that future research explore the impact of hardware advancements and optimization techniques to continue evaluating the security of AES and RSA algorithms in increasingly challenging scenarios.

Key words: cryptography; AES; RSA; security; cipher.

LISTA DE FIGURAS

.

SUMÁRIO

1. INTRODUÇÃO

A criptografia desempenha um papel fundamental na proteção da privacidade e na segurança das comunicações digitais em um mundo cada vez mais conectado. Ao longo dos anos, diversos algoritmos criptográficos foram desenvolvidos com o objetivo de garantir a confidencialidade, autenticidade e integridade dos dados transmitidos pela internet e armazenados em dispositivos eletrônicos. Entre esses algoritmos, destacam-se o AES (Advanced Encryption Standard) e o RSA (Rivest-Shamir-Adleman), que se tornaram amplamente utilizados em uma variedade de aplicações, desde transações bancárias online até a comunicação entre dispositivos IoT (Internet das Coisas).

O tema deste trabalho de conclusão de curso (TCC) é a Verificação da Segurança da Criptografia AES e RSA em relação ao tamanho das chaves, visando uma análise da segurança contra ataques de força bruta. Este tipo de ataque, embora simples em sua abordagem, pode representar uma ameaça significativa à segurança dos dados caso não sejam implementadas medidas de proteção adequadas. O objetivo principal deste estudo é investigar como esses algoritmos se comportam sob diferentes tamanhos de chaves e avaliar a eficácia de suas medidas de segurança na proteção contra esse tipo de ameaça.

Ao longo deste trabalho, serão abordados os princípios fundamentais da criptografia simétrica e assimétrica, com foco especial nos algoritmos AES e RSA. Serão discutidos os conceitos de força bruta e suas implicações na quebra de chaves criptográficas, bem como as estratégias de defesa disponíveis para mitigar esse tipo de ameaça. Além disso, serão apresentadas análises comparativas, utilizando ferramentas computacionais para simular e avaliar o desempenho dos algoritmos em cenários de ataque de força bruta.

Por fim, espera-se que este trabalho contribua para uma melhor compreensão dos desafios enfrentados na proteção de dados sensíveis em ambientes digitais e forneça dados valiosos sobre as melhores práticas de segurança no uso de algoritmos criptográficos. A análise detalhada dos algoritmos AES e RSA para diferentes tamanhos de chaves oferecerá informações importantes para profissionais de segurança da informação e pesquisadores interessados em fortalecer a segurança dos sistemas digitais contra ameaças emergentes.

1.1. Definição do problema

Este trabalho aborda a análise da resistência dos algoritmos AES e RSA em relação ao tamanho da chave, levando em consideração o poder computacional atual. O objetivo é investigar como esses algoritmos se comportam com diferentes tamanhos de chave e como avanços tecnológicos influenciam a segurança dos sistemas criptográficos, proporcionando percepções valiosas para o desenvolvimento de estratégias de segurança mais eficazes e adaptáveis aos desafios contemporâneos da cibersegurança.

1.2. Objetivos

1.2.1. Objetivos Geral

Este trabalho tem como objetivo principal analisar a segurança dos algoritmos de criptografia AES e RSA para diferentes tamanhos de chave a ataques de força bruta. Pretende-se investigar como o poder computacional atual influencia a eficácia desses algoritmos na proteção de dados sensíveis e confidenciais, bem como compreender as possíveis vulnerabilidades que possam comprometer a segurança dos sistemas criptográficos.

1.2.2 Objetivos Específicos

- Avaliar a robustez do algoritmo AES e RSA em cenários de ataques de força bruta, considerando diferentes tamanhos de chave e configurações.
- Comparar o desempenho dos algoritmos AES e RSA em termos de resistência a ataques de força bruta, identificando suas principais diferenças e pontos fortes.
- Analisar as implicações práticas dos resultados obtidos para o desenvolvimento de estratégias de segurança mais eficazes e adaptáveis às demandas da cibersegurança contemporânea.

1.3 Justificativa

A crescente complexidade e sofisticação dos ataques cibernéticos tornam imperativo o aprimoramento contínuo das técnicas e algoritmos de criptografia utilizados para proteger informações sensíveis. Nesse contexto, compreender a resistência dos algoritmos AES e RSA a ataques de força bruta é fundamental para fortalecer a segurança dos sistemas criptográficos e garantir a integridade e confidencialidade dos dados**.**

1.4 Estruturação do trabalho

Este trabalho está estruturado em cinco capítulos principais, além das referências bibliográficas e apêndices, conforme detalhado a seguir.

- No primeiro capítulo, Introdução, apresenta uma visão geral do tema abordado, começando pela definição do problema que motivou a pesquisa, seguida pelos objetivos gerais e específicos do estudo. A justificativa explica a relevância do tema, e a estruturação do trabalho oferece uma visão do conteúdo abordado em cada capítulo.
- O segundo capítulo, Fundamentação Teórica, explora os conceitos essenciais e históricos da criptografia. A seção inicia com uma introdução à criptografia, seguida por uma descrição detalhada dos diferentes tipos de cifras, como transposição, substituição, cifras simétricas e assimétricas. Em seguida, é apresentada uma revisão histórica das principais cifras, incluindo a Cifra de César, Cifra de Vigenère, Enigma, DES, AES e RSA, com foco especial nas características, funcionamento e segurança de cada uma.
- Prossegue-se no terceiro capítulo, Metodologia da Pesquisa, detalha os métodos e procedimentos adotados para conduzir a pesquisa. Inclui a definição e o tipo da pesquisa, o delineamento dos experimentos realizados e a descrição das ferramentas e algoritmos implementados em JavaScript para testar a segurança dos algoritmos AES e RSA de acordo com o tamanho da chave.
- No quarto capítulo, Descrição e Análise dos Resultados, são apresentados os resultados dos testes realizados, comparando o tempo necessário para gerar diferentes quantidades de chaves de diversos tamanhos para ambos os algoritmos, destacando as diferenças observadas e a inviabilidade de ataques de força bruta contra essas técnicas criptográficas.

- Finalmente, o quinto capítulo, Considerações Finais, discute as conclusões do estudo, as contribuições do trabalho, as limitações encontradas e as possibilidades para pesquisas futuras, proporcionando uma visão abrangente e conclusiva sobre a resistência dos algoritmos RSA e AES.

2 – Fundamentação Teórica

2.1 Criptografia

A palavra criptografia é definida por Figueiredo (1913, p. 543) como “escrita secreta, em cifra. O mesmo que ocultismo”. Já Singh (2001, p. 01) explica que “cifras são técnicas para disfarçar uma mensagem de forma que apenas o destinatário pretendido consiga lê-la”. Assunção (2002, p. 64), discorre sobre criptografia sugerindo que:

> Criptografia é a arte da escrita oculta usada desde a antiguidade, por exemplo, pelos egípcios na sua antiga escrita. Ela é muito importante hoje em dia na internet. Mandar um e-mail confidencial da maneira convencional é muito inseguro ele pode ser interceptado no meio da transmissão ou posteriormente, por isto a necessidade do uso de programas eficientes. Esses programas possibilitam uma espécie de “código especial” entre você e o receptor da mensagem, fazendo com que mesmo que alguém consiga obtê-la no meio do caminho, ela será impossível de se ler.

Levy (2001, p. 01) defende que criptografia é:

> O uso de códigos secretos e cifras para embaralhar informações de modo que se tornem inúteis para qualquer pessoa além dos destinatários pretendidos. E é através da magia da criptografia que muitas convenções de comunicação do mundo real - como assinaturas, contratos, recibos e até mesmo jogos de pôquer - encontrarão seu caminho para o comum eletrônico ubíquo.

A criptografia foi evoluindo ao longo do tempo e de acordo com Singh (2001, p 01) “a história dos códigos e cifras é a narrativa da batalha centenária entre criadores de códigos e quebradores de códigos, uma corrida intelectual armamentista que teve um impacto dramático no curso da história”. Singh (2001, p. 02) também discorre que:

> Um código está constantemente sob ataque de quebradores de códigos. Quando os quebradores de códigos desenvolvem uma nova arma que revela uma fraqueza do código, então o código não é mais útil. Ele se torna extinto ou evolui para um novo código mais forte. Por sua vez, este novo código prospera apenas até que os quebradores de códigos identifiquem sua fraqueza, e assim por diante. Isso é semelhante à situação enfrentada, por exemplo, por uma cepa de

> bactérias infecciosas. As bactérias vivem, prosperam e sobrevivem até que os médicos descubram um antibiótico que exponha uma fraqueza nas bactérias e as mate. As bactérias são forçadas a evoluir e enganar o antibiótico, e se bem-sucedidas, prosperarão novamente e se reestabelecerão.

O autor Singh (2001) ainda diz que criptografia em si pode ser dividida em dois ramos, conhecidos como transposição e substituição.

2.2 - Tipos de criptografia

2.2.1 – Transposição

Na transposição, as letras da mensagem são simplesmente rearranjadas, gerando efetivamente um anagrama. Para mensagens muito curtas, como uma única palavra, este método é relativamente inseguro porque existem apenas um número limitado de maneiras de rearranjar um punhado de letras. Por exemplo, três letras podem ser arranjadas de apenas seis maneiras diferentes, por exemplo, vaca, cva, acv, avc, vca, vca. No entanto, à medida que o número de letras aumenta gradualmente, o número de arranjos possíveis rapidamente explode, tornando impossível voltar à mensagem original a menos que o processo de embaralhamento exato seja conhecido. *Por exemplo, considere esta pequena frase.* Contém apenas trinta e cinco letras, e ainda existem mais de 50.000.000.000.000.000.000.000.000.000.000 arranjos distintos entre elas. Se uma pessoa pudesse verificar um arranjo por segundo, e se todas as pessoas do mundo trabalhassem dia e noite, ainda levaria mais de mil vezes a vida útil do universo para verificar todos os arranjos (SINGH, 2001).

2.2.2 Substituição

Stallings diz que uma técnica de substituição é aquela em que as letras do texto claro são substituídas por outras letras, números ou símbolos. Se o texto claro for visto como uma sequência de bits, então a substituição envolve trocar padrões de bits de texto claro por padrões de bits de texto cifrado. O uso mais antigo que conhecemos de uma cifra de substituição, e o mais simples, foi feito por Júlio César. A cifra de César envolve substituir cada letra do alfabeto por aquela que fica três posições adiante (STALLINGS, 2014).

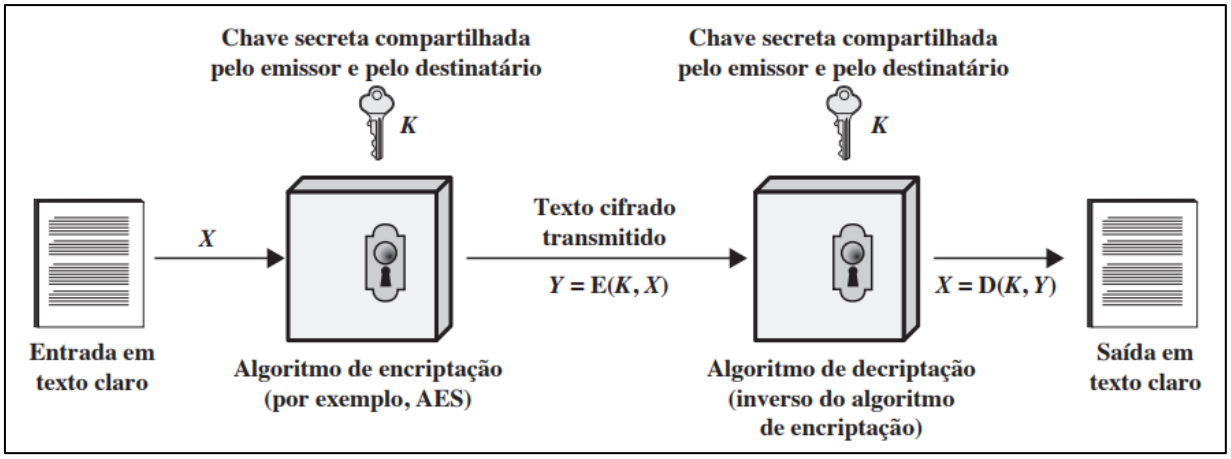
2.2.3 Cifra Simétrica

Segundo Stallings (2014) a encriptação simétrica, também chamada de encriptação convencional ou encriptação de chave única, era o único tipo em uso antes do desenvolvimento da encriptação por chave pública na década de 1970. O autor define o modelo se cifra simétrica cinco itens:

- **Texto claro:** essa é a mensagem ou dados originais, inteligíveis, que servem como entrada do algoritmo de encriptação.
- **Algoritmo de encriptação:** realiza diversas substituições e transformações no texto claro.
- **Chave secreta:** também é uma entrada para o algoritmo de encriptação. A chave é um valor independente do texto claro e do algoritmo. O algoritmo produzirá uma saída diferente, dependendo da chave usada no momento. As substituições e transformações exatas realizadas pelo algoritmo dependem da chave.
- **Texto cifrado:** essa é a mensagem embaralhada, produzida como saída do algoritmo de encriptação. Ela depende do texto claro e da chave secreta. Para determinada mensagem, duas chaves diferentes produzirão dois textos cifrados distintos. O texto cifrado é um conjunto de dados aparentemente aleatório e, nesse formato, ininteligível.
- **Algoritmo de decriptação:** esse é basicamente o algoritmo de encriptação executado de modo inverso. Ele apanha o texto cifrado e a chave secreta e produz o texto claro original.

Singh (2001), diz que quando uma cifra é simétrica, isso significa que o processo de descriptografia é simplesmente o oposto do processo de criptografia. Por exemplo, uma máquina que utiliza uma determinada configuração de chave para cifrar uma mensagem, e o receptor utiliza uma máquina idêntica com a mesma configuração de chave para decifrá-la. Tanto o remetente quanto o receptor possuem conhecimento equivalente e ambos utilizam a mesma chave para criptografar e descriptografar — sua relação é simétrica. A figura 1 demonstra o modelo simples de cifra simétrica.

Figura 1 – Modelo simplificado da encriptação simétrica



Fonte: Adaptado de Stallings (2014)

2.2.4 Cifra Assimétrica

Em um sistema de chave assimétrica, segundo Singh (2001), como o próprio nome sugere, a chave de criptografia e de descriptografia não são idênticas. Em um sistema de cifra assimétrica, se o emissor da mensagem conhece a chave de criptografia, ele pode criptografar uma mensagem, mas não pode descriptografar. Para descriptografar, ele precisa ter acesso à chave de descriptografia. Essa distinção entre chaves de criptografia e descriptografia é o que torna uma cifra assimétrica especial.
Com um sistema de chave pública, outro nome para sistema de chave assimétrica, segundo Levy (2001), cada pessoa poderia gerar um par de chaves único por conta própria, um par consistindo em uma chave pública e uma chave privada, e nenhuma pessoa externa teria acesso às partes secretas das chaves. Então, a comunicação privada poderia começar.
Stallings (2014) discorre que os algoritmos assimétricos contam com uma chave para encriptação e uma chave diferente, porém relacionada, para a decriptação. Eles têm as seguintes características importantes:

- É computacionalmente inviável determinar a chave de decriptação dado apenas o conhecimento do algoritmo de criptografia e da chave de encriptação.

Além disso, alguns algoritmos, como RSA, também exibem esta característica:

- Qualquer uma das duas chaves relacionadas pode ser usada para encriptação, com a outra para a decriptação.

Um esquema de encriptação de chave pública possui cinco elementos:

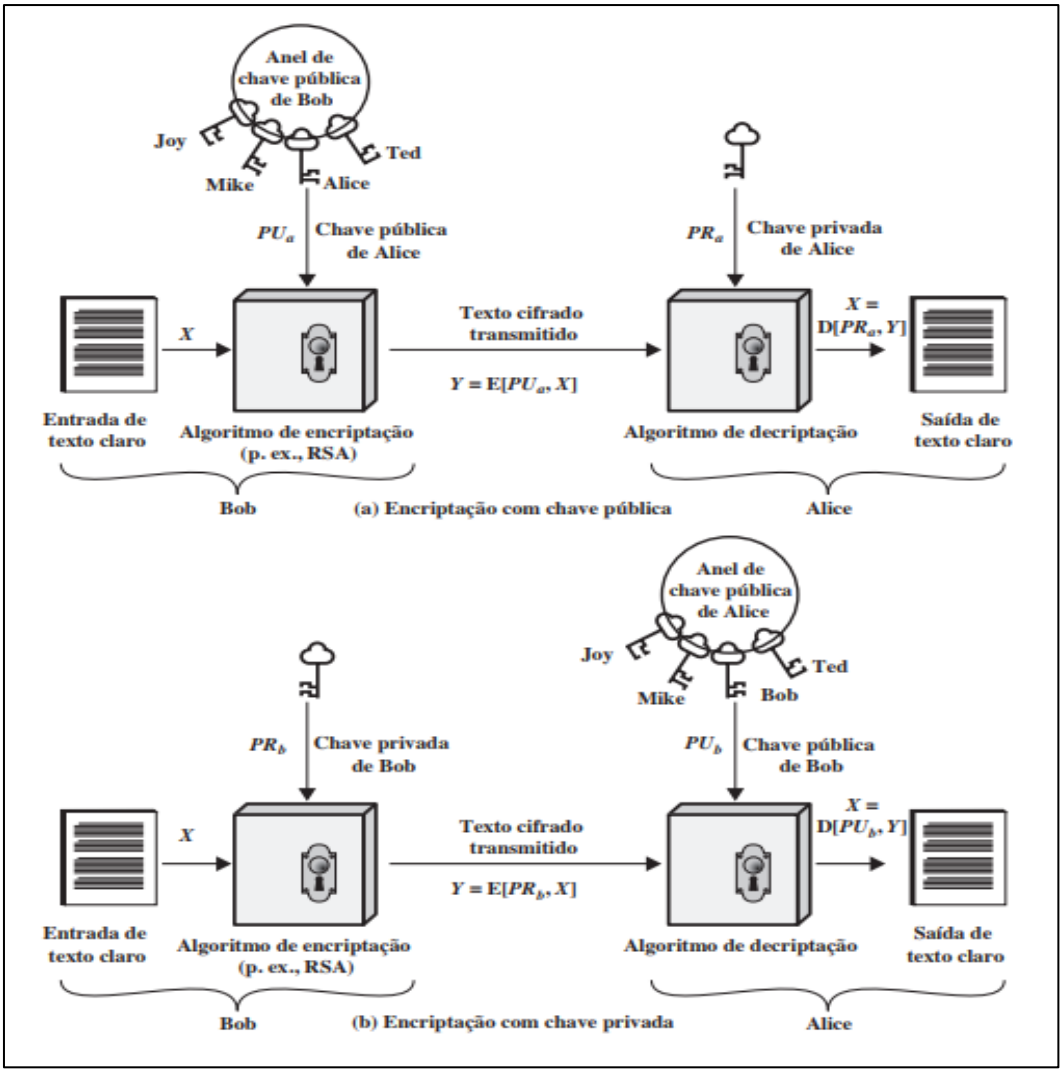
- **Texto claro:** essa é a mensagem ou dados legíveis que são alimentados no algoritmo como entrada.
- **Algoritmo de encriptação:** realiza várias transformações no texto claro.
- **Chaves pública e privada:** esse é um par de chaves que foi selecionado de modo que, se uma for usada para encriptação, a outra é usada para decriptação. As transformações exatas realizadas pelo algoritmo dependem da chave pública ou privada que é fornecida como entrada.
- **Texto cifrado:** essa é a mensagem embaralhada produzida como saída. Ela depende do texto claro e da chave. Para determinada mensagem, duas chaves diferentes produzirão dois textos cifrados diferentes.
- **Algoritmo de decriptação:** aceita o texto cifrado e a chave correspondente e produz o texto claro original.

2.3 História da criptografia

Prieto (2020) diz que o primeiro uso conhecido da criptografia remonta a quase 4.000 anos atrás, por volta do ano 1900 a.C. Foi encontrado em um objeto do antigo Egito, uma inscrição em pedra na tumba de um nobre da cidade de Menet Khufu, perto do Nilo. Esta inscrição detalha os momentos mais significativos de sua vida. Embora os símbolos hieroglíficos usuais da época estejam presentes, foram modificados em certa medida, com algumas substituições entre eles. É pouco provável que o objetivo fosse ocultar uma mensagem, já que não faria sentido descrever a vida de um homem em sua tumba com símbolos incompreensíveis para todos. Prieto (2020) também ressalta a influência da criptográfica dizendo que:

> A corrida dos segredos, como muitas outras, perdura através dos séculos, pois diante de um novo método de ocultação de informações ou de uma nova forma de comunicação segura, surgem novas maneiras de quebrar esses avanços e, portanto, torná-los inúteis. Se o uso da criptografia ao longo da história tivesse gerado uma solução definitiva e plenamente confiável, a história seria diferente. É muito mais relevante para o curso dos eventos a influência que a criptografia teve quando seu uso falhou e as mensagens que se acreditava serem secretas na verdade não eram, do que quando funcionou como esperado. Essas situações provocaram desequilíbrios em um conflito e influenciaram, às vezes de maneira crucial, na história.

Figura 2 – Criptografia de chave pública



Fonte: Adaptado de Stallings (2014)

2.3.1 Cifra de Cesar

Segundo Stallings (2014), o uso mais antigo que conhecemos de uma cifra de substituição, e o mais simples, foi feito por Júlio César. A cifra de César envolve substituir cada letra do alfabeto por aquela que fica três posições adiante. Por exemplo, “meet me after toga party” se torna “PHHW PH DIWHU WKH WRJD SDUWB”

Observe que o alfabeto recomeça no final, de modo que a letra após Z é A. Podemos definir a transformação listando todas as alternativas, da seguinte forma:

claro: a b c d e f g h i j k l m n o p q r s t u v w x y z

cifra: d e f g h i j k l m n o p q r s t u v w x y z a b c

2.3.2 Cifra de Vigenère

A força da cifra de Vigenère reside em seu uso de não uma ou duas, mas vinte e seis alfabetos cifrados distintos para criptografar uma mensagem. O primeiro passo é elaborar uma chamada tabela de Vigenère, um alfabeto de texto simples seguido por vinte e seis alfabetos cifrados, cada um deslocado em uma letra em relação ao alfabeto anterior. Portanto, conforme a figura 2, a linha 1 representa um alfabeto cifrado com um deslocamento de César de 1, o que significa que poderia ser usado para implementar um cifra de César no qual cada letra do texto simples é substituída pela letra um lugar adiante no alfabeto. Da mesma forma, a linha 2 representa um alfabeto cifrado com um deslocamento de César de 2, e assim por diante. A linha superior da tabela, em minúsculas, representa as letras do texto simples. Você pode cifrar cada letra do texto simples de acordo com qualquer um dos vinte e seis alfabetos cifrados. Por exemplo, se o alfabeto cifrado número 2 for usado, então a letra ***a*** é cifrada como ***C***, mas se o alfabeto cifrado número 12 for usado, então ***a*** é cifrada como ***M*** (SINGH, 2001).

Para descriptografar a mensagem, o destinatário pretendido precisa saber qual linha da tabela de Vigenère foi usada para cifrar cada letra, então deve haver um sistema acordado de alternância entre as linhas. Isso é alcançado usando uma palavra-chave. Para ilustrar como uma palavra-chave é usada com a tabela de Vigenère para criptografar uma mensagem de exemplo, vamos cifrar “divert troops to east ridge”, usando a palavra-chave “white” Primeiro, a palavra-chave é escrita acima da mensagem e repetida várias vezes, de modo que cada letra na mensagem é associada a uma letra da palavra-chave. O texto cifrado é então gerado da seguinte forma. Para cifrar a primeira letra, "d", comece identificando a letra-chave acima dela, W, que por sua vez define uma linha específica na tabela de Vigenère. A linha que começa com W, linha 22, é o alfabeto cifrado que será usado para encontrar a letra substituta para o texto simples d. Observamos onde a coluna encabeçada por d intersecta a linha que começa com W, que resulta ser na letra Z. Consequentemente, a letra d no texto simples é representada por Z no texto cifrado (SINGH, 2001).

Figura 03 – Tabela de Vigenère

Plain	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
26	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Fonte: Adaptado de Singh (2001)

Chave	**W H I T E W H I T E W H I T E W H I T E W H I**
Texto claro	**d i v e r t t r o o p s t o e a s t r i d g e**
Cifrado	**Z P D X V P A Z H S L Z B H I W Z B K M Z N M**

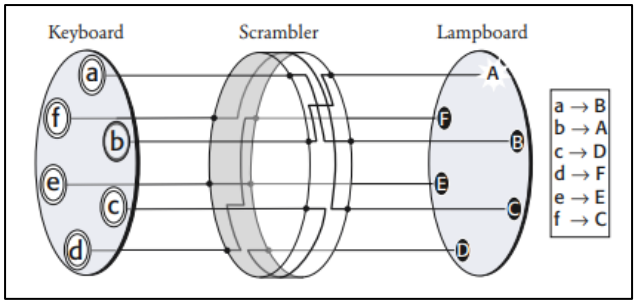
Para cifrar a segunda letra da mensagem, i, o processo é repetido. A letra-chave acima de i é H, então ela é cifrada através de uma linha diferente na tabela de Vigenère (figura 3), a linha H (linha 7), que é um novo alfabeto cifrado. Para cifrar i, observamos onde a coluna encabeçada por i intersecta a linha que começa com H, que resulta ser na letra P. Consequentemente, a letra i no texto simples é representada por P no texto cifrado. Cada letra da palavra-chave indica um alfabeto cifrado específico dentro da tabela de Vigenère, e como a palavra-chave contém cinco letras, o remetente cifra a mensagem atravessando cinco linhas da tabela de Vigenère. A quinta letra da mensagem é cifrada de acordo com a quinta letra da palavra-chave, E, mas para cifrar a sexta letra da mensagem, temos que retornar à primeira letra da palavra-chave. Uma palavra-chave mais longa, ou talvez uma frase-chave, traria mais linhas para o processo de criptografia e aumentaria a complexidade do cifra (SINGH, 2001).

2.3.3 Enigma

Em 1918, o inventor alemão Arthur Scherbius e seu amigo próximo Richard Ritter fundaram a empresa Scherbius & Ritter, uma empresa de engenharia inovadora que se aventurava em tudo, desde turbinas até travesseiros aquecidos. Scherbius estava encarregado da pesquisa e desenvolvimento, e estava constantemente procurando por novas oportunidades. Um de seus projetos favoritos era substituir os sistemas inadequados de criptografia usados na Primeira Guerra Mundial, trocando códigos e cifras tradicionais por uma forma de criptografia que explorasse a tecnologia do século XX. Tendo estudado engenharia elétrica em Hanover e Munique, ele desenvolveu uma máquina criptográfica que era essencialmente uma versão elétrica do disco de cifra de Alberti. Chamada de Enigma, a invenção de Scherbius se tornaria o sistema de criptografia mais temido da história (SINGH, 2001).

De forma esquemática, uma máquina Enigma possui três componentes principais. Primeiramente, um teclado no qual o operador digita o texto em claro, ou seja, digita da mesma forma que faria em uma máquina de escrever tradicional. Em segundo lugar, o mecanismo de cifragem propriamente dito, que recebe a letra pressionada pelo operador e a transforma em outra completamente diferente. Este segundo elemento é o cerne da máquina, é onde a cifragem realmente acontece e onde estão suas forças e fraquezas. O terceiro e último componente seria o painel onde uma série de luzes indica ao operador qual é o resultado da cifragem. Ou seja, o operador pressiona, por exemplo, a tecla A no teclado, o mecanismo de cifragem é acionado e gera a letra de saída, a letra cifrada, que é indicada ao operador através de uma luz no painel de cifragem. O mecanismo funciona através de impulsos elétricos e possui a grande vantagem de que a cada letra cifrada, ele próprio se altera, gerando assim uma pseudo-aleatoriedade contínua na cifragem que torna difícil para um criptoanalista descobrir facilmente a mensagem original (PRIETO, 2020).

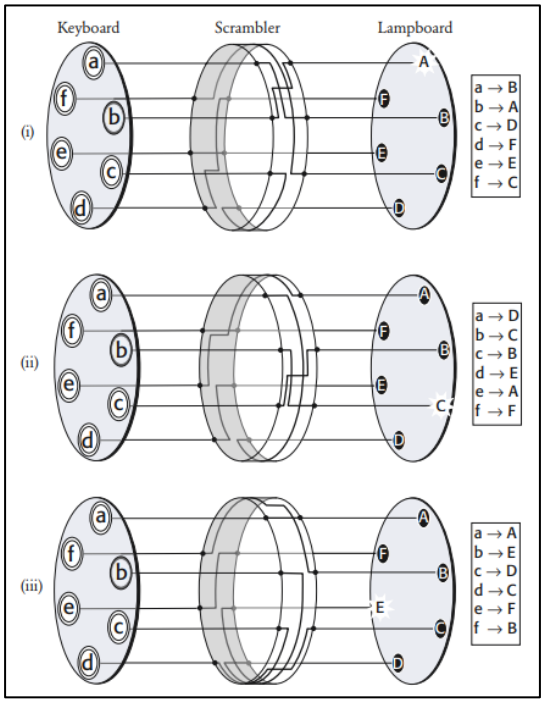
Figura 04 – Versão simplificada do Enigma



Fonte: Adaptado de Singh (2001)

O elemento mais importante da máquina é o embaralhador. Ao digitar b no teclado, uma corrente passa para o embaralhador, segue o caminho da fiação interna e então emerge para iluminar a lâmpada A. Em resumo, b é criptografado como A. A caixa à direita da figura 04 indica como cada uma das seis letras é criptografada (SINGH, 2001).

Figura 05 – Versão simplificada do Enigma dois



Fonte: Adaptado de Singh (2001)

2.3.4 DES

O algoritmo de bloco Data Encryption Standard (DES) é um dos mais conhecidos e foi

padrão mundial por mais de um quarto de século. Em 1972, o instituto norte-americano de padrões, o National Bureau of Standards (NBS), hoje chamado de National Institute of Standards and Technology (NIST), iniciou um projeto para proteger dados digitais de computador e de comunicação. Em maio de 1973, o NBS publicou uma nota solicitando o envio de propostas para um novo algoritmo criptográfico que seria adotado como padrão. Até agosto de 1974, nenhuma das poucas sugestões enviadas preenchia os requisitos preestabelecidos. Numa última tentativa, o NBS publicou uma nova convocação. Só então a IBM enviou um trabalho de Horst Feistel. Por se tratar de material patenteado, a IBM precisou licenciar o algoritmo registrado como LUCIFER. O NBS aprovou o algoritmo e solicitou a ajuda da National Security Agency (NSA), a agência responsável pela segurança nacional norte-americana, para testar o sistema e determinar se era adequado como padrão federal. Depois de algumas modificações, o LUCIFER deu lugar ao DES (TKOTZ, 2005).

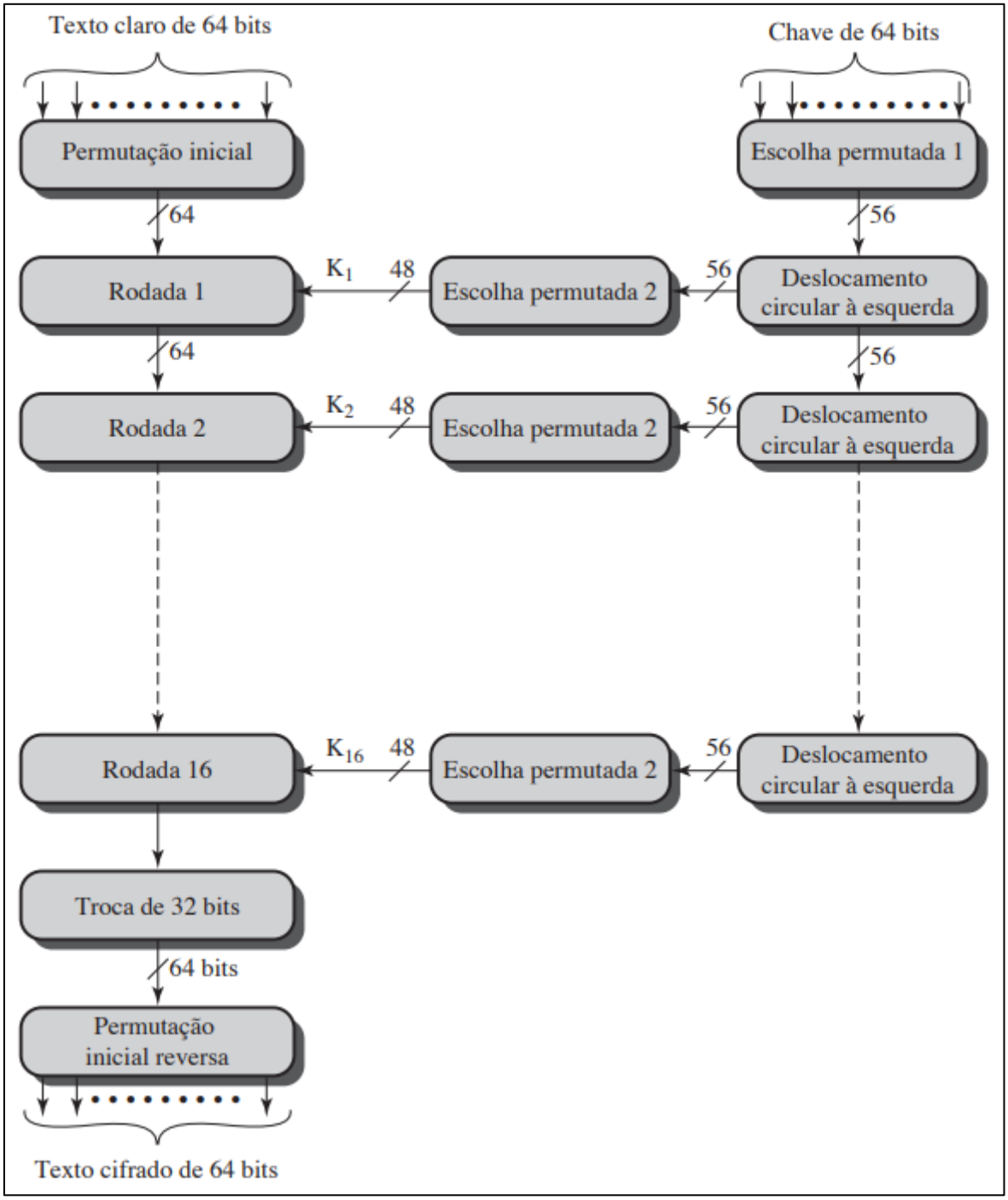
2.3.4.1 Encriptação DES

Assim como em qualquer esquema de encriptação, existem duas entradas na função: o texto claro a ser encriptado e a chave. Nesse caso, o texto claro precisa ter 64 bits de extensão, e a chave tem 56 bits de extensão. O processamento do texto claro prossegue em três fases. Primeiro, o texto claro de 64 bits passa por uma permutação inicial (IP, do acrônimo em inglês para initial permutation), que reorganiza os bits a fim de produzir a entrada permutada. Isso é seguido por uma fase consistindo em 16 rodadas da mesma função, que envolve funções de permutação e substituição. A saída da última (décima sexta) rodada baseia-se em 64 bits que são uma função do texto claro de entrada e da chave. As metades esquerda e direita da saída são trocadas para produzir a pré-saída. Finalmente, a pré-saída é passada por uma permutação [IP–1], que é o inverso da função de permutação inicial, a fim de produzir o texto cifrado de 64 bits (STALLINGS, 2014).

2.3.4.2 Decriptação DES

A decriptação usa o mesmo algoritmo da encriptação, exceto que a aplicação das subchaves é invertida. Além disso, as permutações inicial e final são invertidas (STALLINGS, 2014).

Figura 06 – Representação geral do algoritmo de encriptação DES



Fonte: Adaptado de Stallings (2014)

2.3.4.3 Exemplo do DES

Para este exemplo, o texto claro é um palíndromo hexadecimal. O texto claro, a chave e o texto cifrado resultante são os seguintes:

Texto claro: 02468aceeca86420

Chave: 0f1571c947d9e859

Texto cifrado: da02ce3a89ecac3b

Tabela 01 – Exemplo de DES

Rodada	K_i	L_i	R_i
IP		5a005a00	3cf03c0f
1	1e030f03080d2930	3cf03c0f	bad22845
2	0a31293432242318	bad22845	99e9b723
3	23072318201d0c1d	99e9b723	0bae3b9e
4	05261d3824311a20	0bae3b9e	42415649
5	3325340136002c25	42415649	18b3fa41
6	123a2d0d04262a1c	18b3fa41	9616fe23
7	021f120b1c130611	9616fe23	67117cf2
8	1c10372a2832002b	67117cf2	c11bfc09
9	04292a380c341f03	c11bfc09	887fbc6c
10	2703212607280400	887fbc6c	600f7e8b
11	2826390c31261504	600f7e8b	f596506e
12	12071c241a0a0f08	f596506e	738538b8
13	300935393c0d100b	738538b8	c6a62c4e
14	311e09231321182a	c6a62c4e	56b0bd75
15	283d3e0227072528	56b0bd75	75e8fd8f
16	2921080b13143025	75e8fd8f	25896490
IP-1		da02ce3a	89ecac3b

Fonte: Adaptado de Stallings (2014)

A Tabela 1 mostra a progressão do algoritmo. A primeira linha apresenta os valores de 32 bits das metades esquerda e direita dos dados após a permutação inicial. As 16 linhas seguintes indicam os resultados após cada rodada. Também aparece o valor da subchave de 48 bits gerada para cada rodada. Observe que Li = Ri – 1. A última linha mostra os valores da esquerda e da direita após a permutação inicial inversa. Esses dois valores combinados formam o texto cifrado (STALLINGS 2014).

2.3.4.4 Desuso de chaves de 56 bits

Com um tamanho de chave de 56 bits, existem 256 chaves possíveis, o que é aproximadamente 7,2 × 1016 chaves. Assim, um ataque de força bruta parece ser impraticável. Supondo que, em média, metade do espaço de chave tenha que ser pesquisado, uma única máquina realizando uma encriptação DES por microssegundo

levaria mais de mil anos para quebrar a cifra (STALLINGS, 2014).

Porém, a suposição de uma encriptação por microssegundo é bastante conservadora. Desde 1977, Diffie e Hellman postularam que existia tecnologia para montar uma máquina paralela com 1 milhão de dispositivos de encriptação, cada um podendo realizar uma encriptação por microssegundo. Isso traria o tempo médio de busca para cerca de 10 horas. Os autores estimaram que o custo seria de aproximadamente US$ 20 milhões em dólares de 1977 (STALLINGS, 2014).

Com a tecnologia atual (2014), nem sequer é preciso usar hardware especial. Em vez disso, a velocidade dos processadores comerciais ameaça a segurança do DES. Um artigo da Seagate Technology sugere que uma taxa de 1 bilhão (10^9) de combinações de chaves por segundo é razoável para os computadores multicore de 2014. Tanto a Intel quanto a AMD agora oferecem instruções baseadas em hardware para acelerar o uso do AES. Testes executados em uma máquina Intel multicore resultaram em uma taxa de cerca de meio bilhão de encriptações por segundo. Outra análise sugere que, com a tecnologia de supercomputador contemporânea, uma taxa de 1013 encriptações por segundo é razoável (STALLINGS, 2014).

Tabela 02 - Tempo médio exigido para uma busca exaustiva no espaço de chaves.

Tamanho de chave (bits)	**Cifra**	**número de chaves alternativas**	**Tempo exigido a 10^9 decriptações/s**	**Tempo exigido a 10^{13} decriptações/s**
56	DES	$2^{56} \approx 7{,}2 \times 10^{16}$	2^{55} ns = 1,125 ano	1 hora
128	AES	$2^{128} \approx 3{,}4 \times 10^{38}$	2^{127} ns = 5,3 × 1021 anos	$5{,}3 \times 10^{17}$ anos
168	Triple DES	$2^{168} \approx 3{,}7 \times 10^{50}$	2^{167} ns = 5,8 × 1033 anos	$5{,}8 \times 10^{29}$ anos
192	AES	$2^{192} \approx 6{,}3 \times 10^{57}$	2^{191} ns = 9,8 × 1040 anos	$9{,}8 \times 10^{36}$ anos
256	AES	$2^{256} \approx 1{,}2 \times 10^{77}$	2^{255} ns = 1,8 × 1060 anos	$1{,}8 \times 10^{56}$ ano
26 caracteres (permutação)	Monoalfabétivo	$2! = 4 \times 10^{26}$	2×10^{26} ns = $6{,}3 \times 10^9$ anos	$6{,}3 \times 10^6$ anos

Fonte: Adaptado de Stallings (2014)

A Tabela 02 mostra quanto tempo é necessário a um ataque de força bruta para diversos tamanhos de chave. Como podemos ver, um único PC pode quebrar o DES em cerca de um ano; se vários PCs trabalharem em paralelo, o tempo é reduzido drasticamente. E os supercomputadores de hoje devem ser capazes de descobrir

uma chave em cerca de uma hora. Os tamanhos de chave de 128 bits ou mais são efetivamente inquebráveis usando apenas uma técnica de força bruta. Mesmo que conseguíssemos agilizar o sistema de ataque por um fator de 1 trilhão (1012), ainda seriam necessários 100 mil anos para quebrar um código usando uma chave de 128 bits (STALLINGS 2014).

2.3.4.5 Número de rodadas

Quanto maior o número de rodadas, mais difícil é realizar a criptoanálise, mesmo para uma função F relativamente fraca. Em geral, o critério deverá ser de que o número de rodadas seja escolhido de modo que os esforços criptoanalíticos conhecidos exijam maior ação do que um ataque de busca de chave por força bruta. Esse critério certamente foi usado no projeto do DES. Schneier observa que, para o DES com 16 rodadas, um ataque de criptoanálise diferencial é ligeiramente menos eficiente do que a força bruta: o ataque de criptoanálise diferencial exige 255,1 operações,10 enquanto o de força bruta, 255. Se o DES tivesse 15 ou menos rodadas, a criptoanálise diferencial exigiria menos esforço do que a busca de chave por força bruta (SCHNEIER,1996).

Esse critério é atraente porque facilita julgar a força de um algoritmo e comparar diferentes algoritmos. Na ausência de uma descoberta revolucionária em criptoanálise, a força de qualquer algoritmo que satisfaça o critério acima pode ser julgada unicamente a partir do tamanho da chave (STALLINGS 2014).

2.3.5 AES

2.3.5.1 História

Em janeiro de 1997, o Instituto Nacional de Padrões e Tecnologia dos Estados Unidos (NIST) anunciou o início de uma iniciativa para desenvolver um novo padrão de criptografia: o AES. O novo padrão de criptografia se tornaria um Padrão Federal de Processamento de Informações (FIPS), substituindo o antigo Padrão de Criptografia de Dados (DES) e o Triple DES (DAEMEN, RIJMEN, 2020).

Ao contrário do processo de seleção para o DES, o Algoritmo de Hash Seguro (SHA-1) e o Algoritmo de Assinatura Digital (DSA), o NIST anunciou que o processo de

seleção do AES seria aberto. Qualquer pessoa poderia enviar uma cifra como candidato. Cada envio, desde que atendesse aos requisitos, seria considerado com base em seus méritos. O NIST não realizaria nenhuma avaliação de segurança ou eficiência por si só, mas, em vez disso, convidava a comunidade de criptografia a lançar ataques e tentar analisar os diferentes candidatos, e qualquer pessoa interessada a avaliar o custo de implementação. Todos os resultados poderiam ser enviados ao NIST como comentários públicos para publicação no site do NIST AES ou serem enviados para apresentação em conferências AES. O NIST simplesmente coletaria as contribuições, usando-as como base para sua seleção. O NIST justificaria suas escolhas em relatórios de avaliação (DAEMEN, RIJMEN, 2020).

O escopo oficial de um padrão FIPS é bastante limitado: o FIPS se aplica apenas à Administração Federal dos Estados Unidos. Além disso, o novo AES só seria usado para documentos que contenham informações sensíveis, mas não classificadas. No entanto, previa-se que o impacto do AES seria muito maior do que isso: o AES é o sucessor do DES, que desde sua adoção tem sido utilizado como padrão criptográfico de fato em todo o mundo por bancos, administrações e indústria (DAEMEN, RIJMEN, 2020). Os principais fatores para a rápida aceitação do Rijndael são que ele é um software livre e pode ser implementado facilmente em uma ampla gama de plataformas sem reduzir significativamente a largura de banda (DAEMEN, RIJMEN, 2020).

2.3.5.2 Rijndael

Rijndael é um cifrador de bloco iterado por chave: consiste na aplicação repetida de uma rodada de transformação no estado. O número de rodadas é denotado por N_r e depende do comprimento do bloco e do comprimento da chave (DAEMEN, RIJMEN, 2020).

Uma criptografia com Rijndael consiste em uma adição inicial de chave, denominada AddRoundKey, seguida por N_r-1 aplicações da transformação Round, e finalmente uma aplicação chamada de FinalRound (round final). A adição inicial de chave e cada rodada recebem como entrada o Estado e uma chave de rodada. A chave de rodada para a rodada i é denotada por ExpandedKey[i], e ExpandedKey[0] denota a entrada da adição inicial de chave. A derivação de ExpandedKey a partir da CipherKey é denominada KeyExpansion (DAEMEN, RIJMEN, 2020).

2.3.5.3 Estrutura Geral AES

A cifra recebe como entrada um bloco de texto sem formatação de tamanho 128 bits, ou 16 bytes. O comprimento da chave pode ser 16, 24 ou 32 bytes (128, 192 ou 256 bits). O algoritmo é denominado AES-128, AES-192 ou AES-256, dependendo do tamanho da chave (STALLINGS, 2014).

A entrada para os algoritmos de encriptação e decriptação é um único bloco de 128 bits. No FIPS PUB 197, esse bloco é indicado como uma matriz quadrada de bytes 4 × 4. Esse bloco é copiado para um array Estado, que é modificado a cada etapa de encriptação ou decriptação. Após a etapa final, Estado é copiado para uma matriz de saída. Essas operações são descritas na Figura 07a. De modo semelhante, a chave é apresentada como uma matriz quadrada de bytes. Essa chave é, então, expandida para um conjunto de palavras de chave. A Figura 07b mostra a expansão para a chave de 128 bits. Cada palavra tem quatro bytes, e o conjunto total é de 44 palavras, para a chave de 128 bits. Observe que a ordenação de bytes dentro de uma matriz de chaves é por coluna. Assim, por exemplo, os primeiros quatro bytes de entrada de texto claro de 128 bits para a cifra de encriptação ocupam a primeira coluna da matriz in, os próximos quatro bytes ocupam a segunda coluna, e assim por diante. Da mesma forma, os primeiros quatro bytes da chave expandida, que forma uma palavra, ficam na primeira coluna da matriz w (STALLINGS, 2014).

Tabela 03 – Parâmetros do AES.

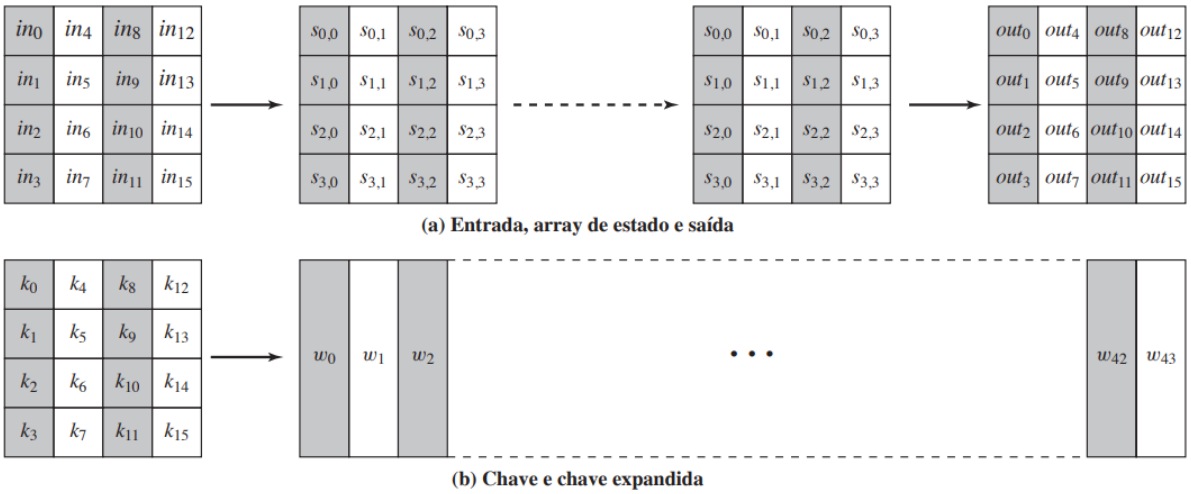
Tamanho da chave (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Tamanho do bloco de texto claro (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Número de rodadas	10	12	14
Tamanho da chave de rodada (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Tamanho da chave expandida (words/bytes)	44/176	52/208	60/240

Fonte: Stallings (2014)

A cifra consiste em N rodadas, e o número delas depende do comprimento da chave: 10 rodadas para uma chave de 16 bytes, 12 para uma chave de 24 bytes e 14 para uma chave de 32 bytes (Tabela 3). As primeiras N – 1 rodadas consistem em quatro funções de transformação distintas: SubBytes, ShiftRows, MixColumns e AddRoundKey. A rodada final contém apenas três transformações, e há uma transformação inicial única (AddRoundKey) antes da primeira rodada, o que pode ser

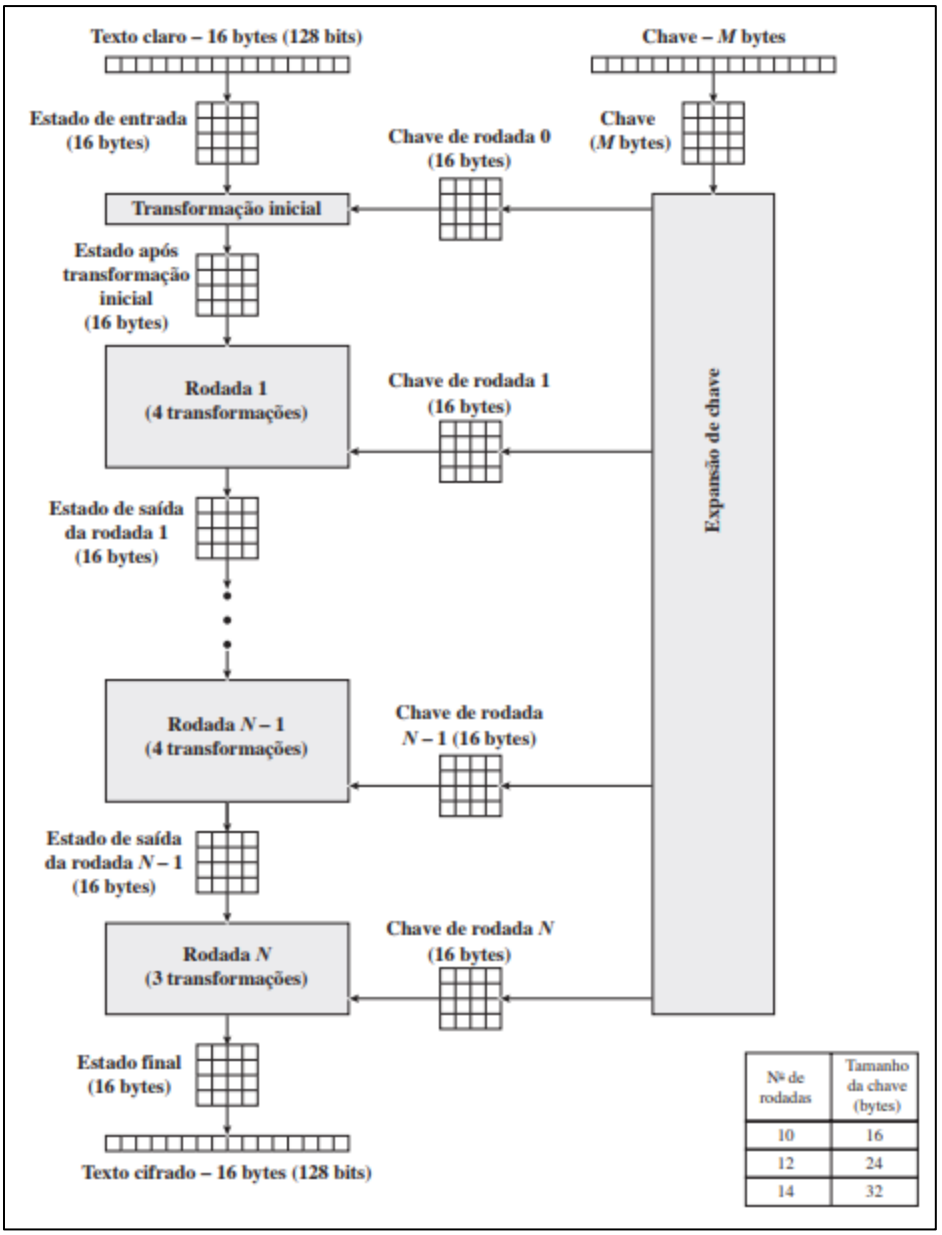
considerado rodada 0 (STALLINGS, 2014).

Figura 07 – Estrutura de dados do AES



Fonte: Adaptado de Stallings (2014)

Figura 08 – Processo de encriptação do AES



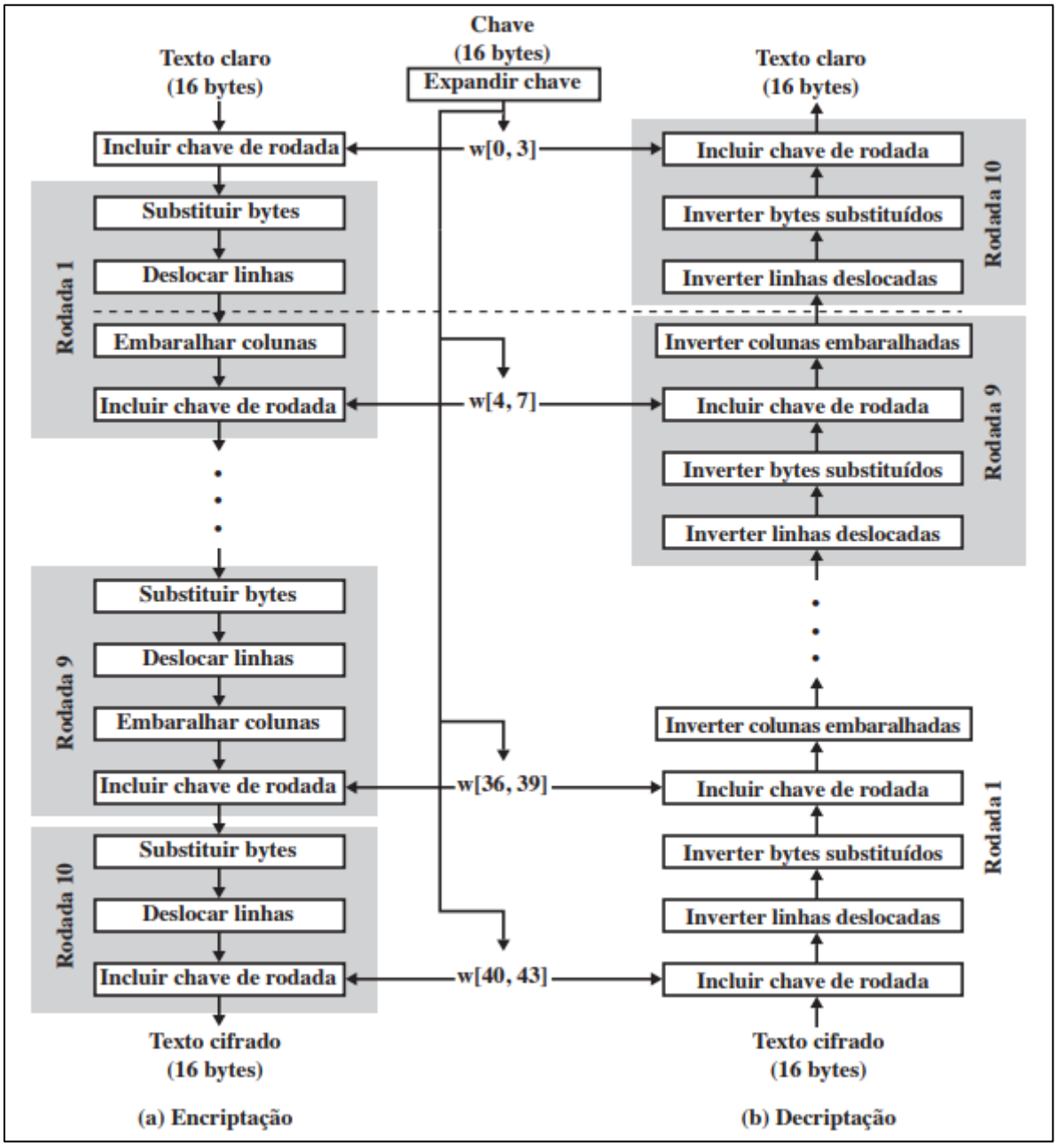
Fonte: Adaptado de Stallings (2014)

Cada transformação usa uma ou mais matrizes de 4×4 como entrada e produz uma de 4×4 como saída. A Figura 08 mostra que a saída de cada rodada é uma matriz de 4×4, com a saída da rodada final sendo o texto cifrado. Além disso, a função de expansão de chave gera N + 1 chaves de rodada, cada uma das quais é uma matriz distinta de 4×4. Cada chave de rodada serve como uma das entradas para a transformação AddRoundKey em cada rodada (STALLINGS, 2014).

2.3.5.4 Estrutura detalhada

A Figura 09 mostra a cifra AES com mais detalhes, indicando a sequência de transformações em cada rodada e a função de decriptação correspondente.

Figura 09 – Encriptação e decriptação do AES



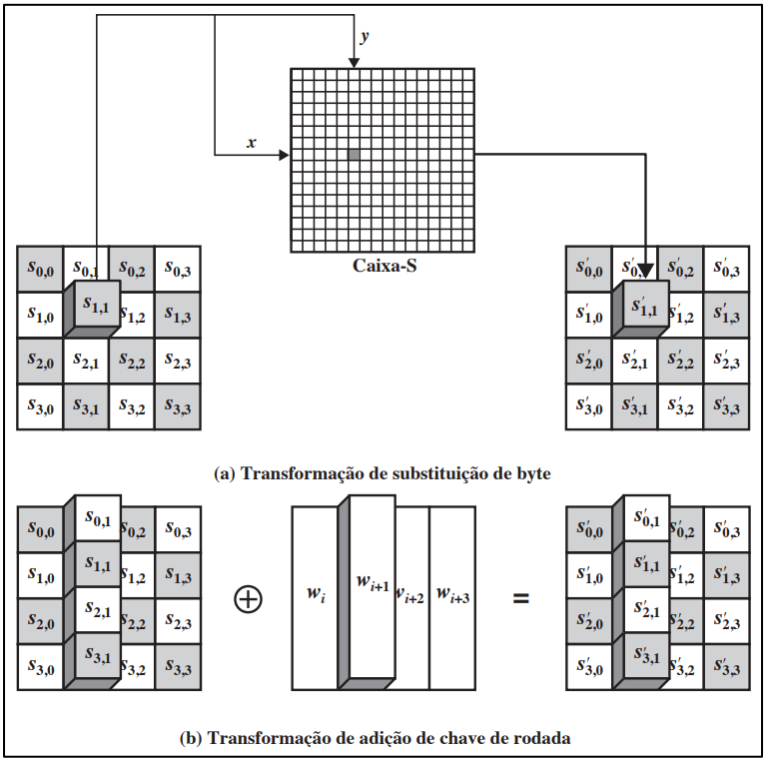
Fonte: Adaptado de Stallings (2014)

2.3.5.5 Funções De Transformação Do AES

2.3.5.5.1 SubBytes

A transformação direta de substituição de byte, chamada SubBytes, consiste em uma simples pesquisa em tabela (Figura 10a). AES define uma matriz de 16 × 16 de valores de byte, chamada de S-box (Tabela 4), que contém uma permutação de todos os 256 valores possíveis de 8 bits. Cada byte individual de Estado é mapeado para um novo byte da seguinte maneira: os 4 bits mais à esquerda do byte são usados como um valor de linha e os 4 bits mais à direita, como um valor de coluna. Esses valores de linha e coluna servem como índices para a S-box a fim de selecionar um valor de saída de 8 bits. Por exemplo, o valor hexadecimal {95} referência a linha 9, coluna 5 da S-box, que contém o valor {2A}. De acordo com isso, o valor {95} é mapeado para o {2A} (STALLINGS, 2014).

Figura 10 – Operação em nível de byte AES



Fonte: Adaptado de Stallings (2014)

Tabela 04 – S-box do AES

		y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a)

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b)

Fonte: Stallings (2014)

2.3.5.5.2 ShiftRows

A transformação direta de deslocamento de linhas, chamada ShiftRows, é representada na Figura 11a. A primeira linha de Estado não é alterada. Para a segunda linha, é realizado um deslocamento circular à esquerda por 1 byte. Para a terceira linha, é feito um deslocamento circular à esquerda por 2 bytes. Para a quarta

linha, ocorre um deslocamento circular à esquerda por 3 bytes. A seguir vemos um exemplo de ShiftRows.

87	F2	4D	97	>>	87	F2	4D	97
EC	6E	4C	90		6E	4C	90	EC
4A	C3	46	E7		46	E7	4A	C3
8C	D8	95	A6		A6	8C	D8	95

A transformação inversa de deslocamento de linhas, chamada InvShiftRows, realiza os deslocamentos circulares na direção oposta para cada uma das três últimas linhas, com um deslocamento circular à direita por um byte para a segunda linha, e assim por diante (STALLINGS, 2014).

A transformação de deslocamento de linhas é mais substancial do que pode parecer a princípio. Isso porque o Estado, além da entrada e saída da cifra, é tratado como um array de quatro colunas de 4 bytes. Assim, na encriptação, os quatro primeiros bytes do texto claro são copiados para a primeira coluna de Estado, e assim por diante. Além disso, conforme veremos, a chave da rodada é aplicada ao Estado coluna por coluna. Assim, um deslocamento de linha move um único byte de uma coluna para outra, que está a uma distância linear de um múltiplo de 4 bytes. Observe também que a transformação garante que os 4 bytes de uma coluna são espalhados para quatro colunas diferentes (STALLINGS, 2014).

2.3.5.5.3 MixColumns

A transformação direta de embaralhamento de colunas, chamada MixColumns, opera sobre cada coluna individualmente. Cada byte de uma coluna é mapeado para um novo valor que é determinado em função de todos os quatro bytes nessa coluna. A transformação pode ser definida pela seguinte multiplicação de matriz sobre Estado (Figura 11) (STALLINGS, 2014).

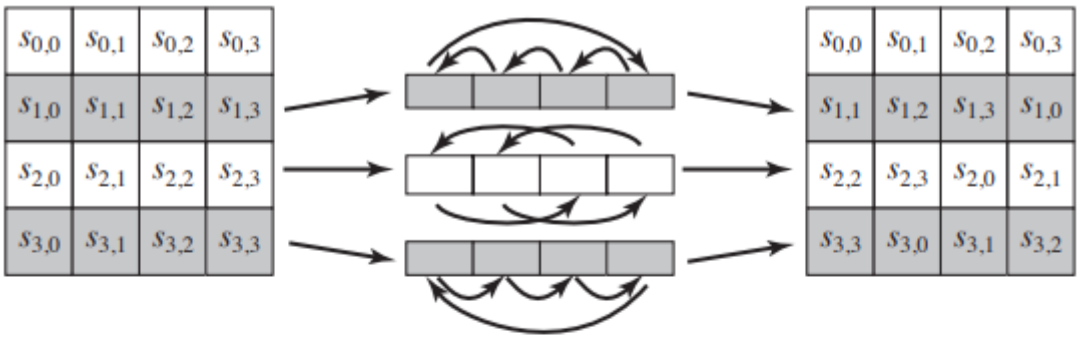
$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

(5.3)

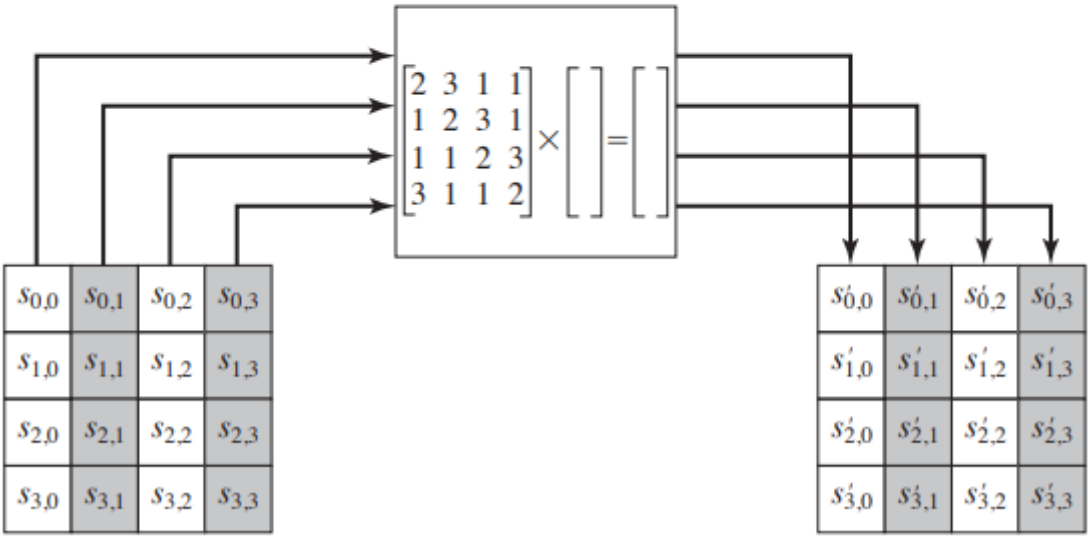
Cada elemento na matriz de produtos é a soma dos produtos dos elementos de uma linha e uma coluna. Nesse caso, as adições e multiplicações individuais são realizadas em GF(2^8). A transformação MixColumns sobre uma única coluna de Estado pode ser expressa como:

$$s'_{0,j} = (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$
$$s'_{1,j} = s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j}$$
$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j})$$
$$s'_{3,j} = (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j})$$

Figura 11 – Operações de linha e coluna do AES



(a) Transformação de deslocamento de linhas



(b) Transformação de embaralhamento de colunas

Fonte: Stallings (2014)

A seguir vemos um exemplo de MixColumns:

87	F2	4D	97	>>	47	40	A3	4C
6E	4C	90	EC		37	D4	70	9F
46	E7	4A	C3		94	E4	3A	42
A6	8C	D8	95		ED	A5	A6	BC

A transformação inversa de embaralhamento de colunas, chamada InvMixColumns, é definida pela seguinte multiplicação de matrizes:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

(5.5)

Fica imediatamente claro que a Equação 5.5 é o inverso da Equação 5.3. Precisamos mostrar que:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

que é equivalente a mostrar que:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(STALLINGS, 2014)

Os coeficientes da matriz na Equação 5.3 são baseados em um código linear com máxima distância entre as palavras, o que garante um bom embaralhamento entre os bytes de cada coluna. A transformação de embaralhamento de colunas combinada com a de deslocamento de linhas garante que, após algumas rodadas, todos os bits da saída dependam de todos os bits da entrada (DAEMEN, RIJMEN, 1999)

2.3.5.5.4 Transformação addroundKey

Na transformação direta de adição de chave da rodada, chamada AddRoundKey, os 128 bits de Estado passam por um XOR bit a bit com os 128 bits da chave da rodada. A operação é vista como uma do tipo coluna por coluna entre os 4 bytes da coluna Estado e uma word da chave da rodada; ela também pode ser vista como uma operação em nível de byte. A seguir está um exemplo de AddRoundKey:

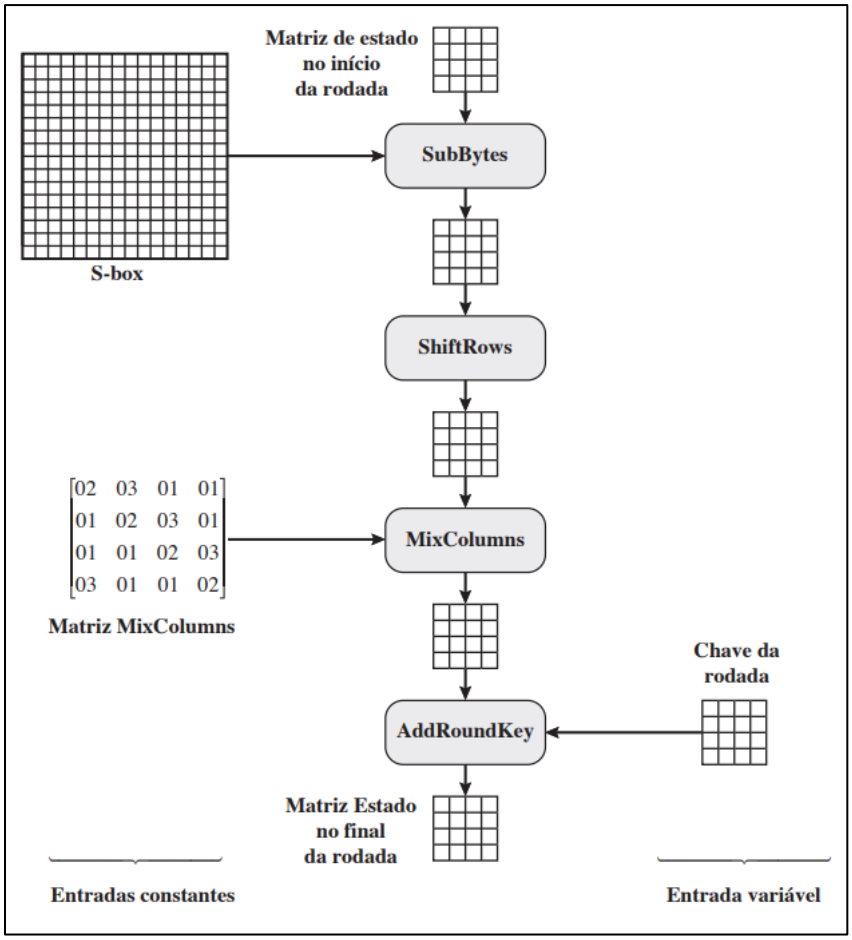
47	40	A3	4C	⊕	AC	19	28	57	⊖	EB	59	8B	1B
37	D4	70	9F		77	FA	D1	5C		40	2E	A1	C3
94	E4	3A	42		66	DC	29	0		F2	38	13	42
ED	A5	A6	BC		F3	21	41	6A		1E	84	E7	D6

A primeira matriz é o Estado, e a segunda, a chave da rodada.

A transformação inversa de adição de chave da rodada é idêntica à direta de adição de chave da rodada, pois a operação XOR é o seu próprio inverso (STALLINGS, 2014).

A transformação de adição de chave da rodada é a mais simples possível, e afeta cada bit de Estado. A complexidade da expansão de chave da rodada, mais a dos outros estágios do AES, garantem a sua segurança. A Figura 12 apresenta outra visão de uma rodada única do AES, enfatizando os mecanismos e entradas de cada transformação (STALLINGS, 2014).

Figura 12 - Entradas para rodada única do AES.



Fonte: Stallings (2014)

2.3.5.6 Expansão de chave do AES

O algoritmo de expansão de chave do AES, de acordo com Stallings (STALLINGS, 2014), utiliza como entrada uma chave de 4 words (16 bytes) e produz um array linear de 44 words (176 bytes). Isso é suficiente para oferecer uma chave da rodada de 4 words para o estágio AddRoundKey inicial e para cada uma das 10 rodadas da cifra. O pseudocódigo a seguir descreve a expansão:

```
KeyExpansion (byte key[16], word w[44])
{
   word temp
   for (i = 0; i < 4; i++)    w[i] = (key[4*i], key[4*i+1],
                                      key[4*i+2],
                                      key[4*i+3]);
    for (i = 4; i < 44; i++)
    {
     temp = w[i – 1];
     if (i mod 4 = 0)    temp = SubWord (RotWord (temp))
                                ⊕ Rcon[i/4];
     w[i] = w[i-4] ⊕ temp
    }
}
```

A chave é copiada para as quatro primeiras words da chave expandida. O restante da chave expandida é preenchido com quatro words de cada vez. Cada word incluída w[i] depende da imediatamente anterior, w[i – 1], e da word quatro posições atrás, w[i – 4]. Em três dentre quatro casos, um simples XOR é usado. Para uma word cuja posição no array w é um múltiplo de 4, uma função mais complexa é empregada. A Figura 5.9 ilustra a geração das oito primeiras words da chave expandida, com o símbolo g para representar essa função complexa. A função g consiste nas seguintes subfunções:

1. RotWord realiza um deslocamento circular de um byte à esquerda em uma word. Isso significa que uma word de entrada [B0, B1, B2, B3] é transformada em [B1, B2, B3, B0].
2. SubWord realiza uma substituição byte a byte de sua word de entrada, usando a S-box (Tabela 5.2a).
3. O resultado das etapas 1 e 2 passa por um XOR com a constante da rodada, Rcon[j].

A constante da rodada é uma word em que os três bytes mais à direita são sempre 0. Assim, o efeito de um XOR de uma word com Rcon se resume a realizar um XOR no byte mais à esquerda da word. A constante da rodada é diferente para cada uma delas, e é definida como Rcon[j] = (RC[j], 0, 0, 0), com RC[1] = 1, RC[j] = 2 · RC[j – 1] e com a multiplicação definida sobre o corpo GF(28). Os valores de RC(j) em hexadecimal são:

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

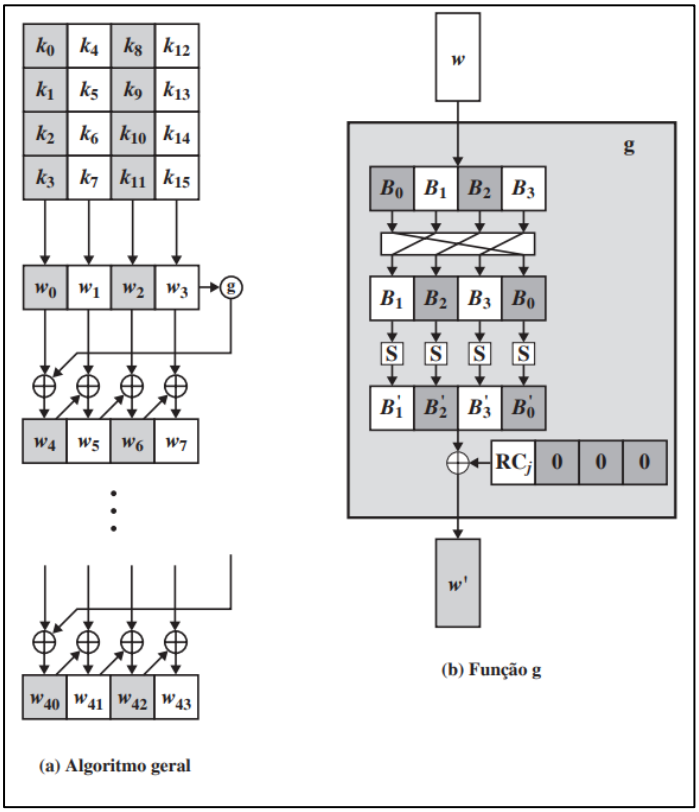
Por exemplo, suponha que a chave da rodada 8 seja

“EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F”

Então, os 4 primeiros bytes (primeira coluna) da chave da rodada 9 são calculados da seguinte forma:

i (decimal)	temp	Após RotWord	Após SubWord	Rcon(9)	Após XOR com Rcon	w[i-4]	w[i] = temp ⊕ w[i-4]
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

Figura 13 – Expansão de chave do AES



Fonte: Stallings (2014)

2.3.5.7 Exemplo de AES

Para este exemplo, o texto claro é um palíndromo hexadecimal. O texto claro, a chave e o texto cifrado resultante são:

Texto claro:	0 1 2 3 4 5 6 7 8 9 a b c d e f f e d c b a 9 8 7 6 5 4 3 2 1 0
Chave:	0 f 1 5 7 1 c 9 4 7 d 9 e 8 5 9 0 c b 7 a d d 6 a f 7 f 6 7 9 8
Texto cifrado:	f f 0 b 8 4 4 a 0 8 5 3 b f 7 c 6 9 3 4 a b 4 3 6 4 1 4 8 f b 9

Tabela 5 – Expansão de chave para o exemplo de AES

Words de chave	Função auxiliar
w0 = 0f 15 71 c9 w1 = 47 d9 e8 59 w2 = 0c b7 ad d6 w3 = af 7f 67 98	RotWord (w3) = 7f 67 98 af = x1 SubWord (x1) = d2 85 46 79 = y1 Rcon (1) = 01 00 00 00 y1 ⊕ Rcon (1) = d3 85 46 79 = z1
w4 = w0 ⊕ z1 = dc 90 37 b0 w5 = w4 ⊕ w1 = 9b 49 df e9 w6 = w5 ⊕ w2 = 97 fe 72 3f w7 = w6 ⊕ w3 = 38 81 15 a7	RotWord (w7) = 81 15 a7 38 = x2 SubWord (x2) = 0c 59 5c 07 = y2 Rcon (2) = 02 00 00 00 y2 ⊕ Rcon (2) = 0e 59 5c 07 = z2
w8 = w4 ⊕ z2 = d2 c9 6b b7 w9 = w8 ⊕ w5 = 49 80 b4 5e w10 = w9 ⊕ w6 = de 7e c6 61 w11 = w10 ⊕ w7 = e6 ff d3 c6	RotWord (w11) = ff d3 c6 e6 = x3 SubWord (x3) = 16 66 b4 83 = y3 Rcon (3) = 04 00 00 00 y3 ⊕ Rcon (3) = 12 66 b4 8e = z3
w12 = w8 ⊕ z3 = c0 af df 39 w13 = w12 ⊕ w9 = 89 2f 6b 67 w14 = w13 ⊕ w10 = 57 51 ad 06 w15 = w14 ⊕ w11 = b1 ae 7e c0	RotWord (w15) = ae 7e c0 b1 = x4 SubWord (x4) = e4 f3 ba c8 = y4 Rcon (4) = 08 00 00 00 y4 ⊕ Rcon (4) = ec f3 ba c8 = 4
w16 = w12 ⊕ z4 = 2c 5c 65 f1 w17 = w16 ⊕ w13 = a5 73 0e 96 w18 = w17 ⊕ w14 = f2 22 a3 90 w19 = w18 ⊕ w15 = 43 8c dd 50	RotWord(w19) = 8c dd 50 43 = x5 SubWord(x5) = 64 c1 53 1a = y5 Rcon(5) = 10 00 00 00 y5 ⊕ Rcon(5) = 74 c1 53 1a = z5
w20 = w16 ⊕ z5 = 58 9d 36 eb w21 = w20 ⊕ w17 = fd ee 38 7d w22 = w21 ⊕ w18 = 0f cc 9b ed w23 = w22 ⊕ w19 = 4c 40 46 bd	RotWord (w23) = 40 46 bd 4c = x6 SubWord (x6) = 09 5a 7a 29 = y6 Rcon(6) = 20 00 00 00 y6 ⊕ Rcon(6) = 29 5a 7a 29 = z6
w24 = w20 ⊕ z6 = 71 c7 4c c2 w25 = w24 ⊕ w21 = 8c 29 74 bf w26 = w25 ⊕ w22 = 83 e5 ef 52 w27 = w26 ⊕ w23 = cf a5 a9 ef	RotWord (w27) = a5 a9 ef cf = x7 SubWord (x7) = 06 d3 bf 8a = y7 Rcon (7) = 40 00 00 00 y7 ⊕ Rcon(7) = 46 d3 df 8a = z7
w28 = w24 ⊕ z7 = 37 14 93 48 w29 = w28 ⊕ w25 = bb 3d e7 f7 w30 = w29 ⊕ w26 = 38 d8 08 a5 w31 = w30 ⊕ w27 = f7 7d a1 4a	RotWord (w31) = 7d a1 4a f7 = x8 SubWord (x8) = ff 32 d6 68 = y8 Rcon (8) = 80 00 00 00 y8 ⊕ Rcon(8) = 7f 32 d6 68 = z8
w32 = w28 ⊕ z8 = 48 26 45 20 w33 = w32 ⊕ w29 = f3 1b a2 d7 w34 = w33 ⊕ w30 = cb c3 aa 72 w35 = w34 ⊕ w32 = 3c be 0b 3	RotWord (w35) = be 0b 38 3c = x9 SubWord (x9) = ae 2b 07 eb = y9 Rcon (9) = 1B 00 00 00 y9 ⊕ Rcon (9) = b5 2b 07 eb = z9
w36 = w32 ⊕ z9 = fd 0d 42 cb w37 = w36 ⊕ w33 = 0e 16 e0 1c w38 = w37 ⊕ w34 = c5 d5 4a 6e w39 = w38 ⊕ w35 = f9 6b 41 56	RotWord (w39) = 6b 41 56 f9 = x10 SubWord (x10) = 7f 83 b1 99 = y10 Rcon (10) = 36 00 00 00 y10 ⊕ Rcon (10) = 49 83 b1 99 = z10
w40 = w36 ⊕ z10 = b4 8e f3 52 w41 = w40 ⊕ w37 = ba 98 13 4e w42 = w41 ⊕ w38 = 7f 4d 59 20 w43 = w42 ⊕ w39 = 86 26 18 76	

Fonte: Stallings (2014)

A Tabela 5 mostra a expansão da chave de 16 bytes para 10 chaves de rodada. Como explicado anteriormente, esse processo é realizado word a word, com cada uma de quatro bytes ocupando uma coluna da matriz de chave de rodada. A coluna da esquerda mostra as quatro words de chave de rodada geradas para cada rodada. Já a coluna da direita mostra os passos usados para gerar a word auxiliar empregada na expansão da chave. Começamos, é claro, com a chave em si servindo como uma de rodada para a rodada 0.

Tabela 6 – Exemplo do AES

Início da rodada	Após SubBytes	Após ShiftRows	Após MixColumns	Chave de rodada
01 89 fe 76 23 ab dc 54 45 cd ba 32 67 ef 98 10				0f 47 0c af 15 d9 b7 7f 71 e8 ad 67 c9 59 d6 98
0e ce f2 d9 36 72 6b 2b 34 25 17 55 ae b6 4e 88	ab 8b 89 35 05 40 7f f1 18 3f f0 fc e4 4e 2f c4	ab 8b 89 35 40 7f f1 05 f0 fc 18 3f c4 e4 4e 2f	b9 94 57 75 e4 8e 16 51 47 20 9a 3f c5 d6 f5 3b	dc 9b 97 38 90 49 fe 81 37 df 72 15 b0 e9 3f a7
65 0f c0 4d 74 c7 e8 d0 70 ff e8 2a 75 3f ca 9c	4d 76 ba e3 92 c6 9b 70 51 16 9b e5 9d 75 74 de	4d 76 ba e3 c6 9b 70 92 9b e5 51 16 de 9d 75 74	8e 22 db 12 b2 f2 dc 92 df 80 f7 c1 2d c5 1e 52	d2 49 de e6 c9 80 7e ff 6b b4 c6 d3 b7 5e 61 c6
5c 6b 05 f4 7b 72 a2 6d b4 34 31 12 9a 9b 7f 94	4a 7f 6b bf 21 40 3a 3c 8d 18 c7 c9 b8 14 d2 22	4a 7f 6b bf 40 3a 3c 21 c7 c9 8d 18 22 b8 14 d2	b1 c1 0b cc ba f3 8b 07 f9 1f 6a c3 1d 19 24 5c	c0 89 57 b1 af 2f 51 ae df 6b ad 7e 39 67 06 c0
71 48 5c 7d 15 dc da a9 26 74 c7 bd 24 7e 22 9c	a3 52 4a ff 59 86 57 d3 f7 92 c6 7a 36 f3 93 de	a3 52 4a ff 86 57 d3 59 c6 7a f7 92 de 36 f3 93	d4 11 fe 0f 3b 44 06 73 cb ab 62 37 19 b7 07 ec	2c a5 f2 43 5c 73 22 8c 65 0e a3 dd f1 96 90 50
f8 b4 0c 4c 67 37 24 ff ae a5 c1 ea e8 21 97 bc	41 8d fe 29 85 9a 36 16 e4 06 78 87 9b fd 88 65	41 8d fe 29 9a 36 16 85 78 87 e4 06 65 9b fd 88	2a 47 c4 48 83 e8 18 ba 84 18 27 23 eb 10 0a f3	58 fd 0f 4c 9d ee cc 40 36 38 9b 46 eb 7d ed bd
72 ba cb 04 1e 06 d4 fa b2 20 bc 65 00 6d e7 4e	40 f4 1f f2 72 6f 48 2d 37 b7 65 4d 63 3c 94 2f	40 f4 1f f2 6f 48 2d 72 65 4d 37 b7 2f 63 3c 94	7b 05 42 4a 1e d0 20 40 94 83 18 52 94 c4 43 fb	71 8c 83 cf c7 29 e5 a5 4c 74 ef a9 c2 bf 52 ef
0a 89 c1 85 d9 f9 c5 e5 d8 f7 f7 fb 56 7b 11 14	67 a7 78 97 35 99 a6 d9 61 68 68 0f b1 21 82 fa	67 a7 78 97 99 a6 d9 35 68 0f 61 68 fa b1 21 82	ec 1a c0 80 0c 50 53 c7 3b d7 00 ef b7 22 72 e0	37 bb 38 f7 14 3d d8 7d 93 e7 08 a1 48 f7 a5 4a
db a1 f8 77 18 6d 8b ba a8 30 08 4e ff d5 d7 aa	b9 32 41 f5 ad 3c 3d f4 c2 04 30 2f 16 03 0e ac	b9 32 41 f5 3c 3d f4 ad 30 2f c2 04 ac 16 03 0e	b1 1a 44 17 3d 2f ec b6 0a 6b 2f 42 9f 68 f3 b1	48 f3 cb 3c 26 1b c3 be 45 a2 aa 0b 20 d7 72 38
f9 e9 8f 2b 1b 34 2f 08 4f c9 85 49 bf bf 81 89	99 1e 73 f1 af 18 15 30 84 dd 97 3b 08 08 0c a7	99 1e 73 f1 18 15 30 af 97 3b 84 dd a7 08 08 0c	31 30 3a c2 ac 71 8c c4 46 65 48 eb 6a 1c 31 62	fd 0e c5 f9 0d 16 d5 6b 42 e0 4a 41 cb 1c 6e 56
cc 3e ff 3b a1 67 59 af 04 85 02 aa a1 00 5f 34	4b b2 16 e2 32 85 cb 79 f2 97 77 ac 32 63 cf 18	4b b2 16 e2 85 cb 79 32 77 ac f2 97 18 32 63 cf	4b 86 8a 36 b1 cb 27 5a fb f2 f2 af cc 5a 5b cf	b4 ba 7f 86 8e 98 4d 26 f3 13 59 18 52 4e 20 76
ff 08 69 64 0b 53 34 14 84 bf ab 8f 4a 7c 43 b9				

Fonte: Stallings (2014)

A Tabela 6 mostra a progressão de Estado através do processo de encriptação do AES. A primeira coluna indica o valor de Estado no início de uma rodada. Para a primeira linha, Estado é apenas a disposição em forma matricial do texto claro. A segunda, a terceira e a quarta colunas apresentam o valor de Estado para esta rodada após as transformações SubBytes, ShiftRows e MixColumns, respectivamente. A quinta coluna mostra a chave de rodada. Você pode verificar que essas chaves de rodada se equiparam com aquelas na Tabela 5. A primeira coluna exibe o valor de Estado resultante do XOR bit a bit de Estado após os MixColumns posteriores com a chave de rodada para a anterior (STALLINGS, 2014).

2.3.6 RSA

O artigo pioneiro de Diffie e Hellman introduziu uma nova técnica para criptografia e, com efeito, desafiou os criptologistas a encontrarem um algoritmo criptográfico que atendesse os requisitos para os sistemas de chave pública. Diversos algoritmos foram propostos. Alguns deles, embora inicialmente promissores, provaram ser falhos (DIFFIE, HELLMAN, 1976).

Uma das primeiras respostas ao desafio foi desenvolvida em 1977 por Ron Rivest, Adi Shamir e Len Adleman, no MIT, e publicada em 1978. O esquema Rivest-Shamir-Adleman (RSA), desde essa época, tem reinado soberano como a técnica de uso geral mais aceita e implementada para a encriptação de chave pública (RIVEST; SHAMIR e ADLEMAN, 1978).

O esquema RSA é uma cifra de bloco em que o texto claro e o cifrado são inteiros entre 0 e n − 1, para algum n. Um tamanho típico para n é 1024 bits, ou 309 dígitos decimais. Ou seja, n é menor que 2^{1024}. RSA utiliza uma expressão com exponenciais. O texto claro é encriptado em blocos, com cada um tendo um valor binário menor que algum número n. Ou seja, o tamanho do bloco precisa ser menor ou igual a log2(n) + 1; na prática, o tamanho do bloco é de i bits, onde $2^i < n \leq 2^{i+1}$. A encriptação e a decriptação têm a seguinte forma, para algum bloco de texto claro M e bloco de texto cifrado C:

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Tanto o emissor quanto o receptor precisam conhecer o valor de n. O emissor conhece o valor de e, e somente o receptor sabe do valor de d. Assim, esse é um algoritmo de encriptação de chave pública com uma chave pública PU = {e, n} e uma chave privada PR = {d, n}. Para que esse algoritmo seja satisfatório à encriptação de chave pública, os seguintes requisitos precisam ser atendidos:

1. É possível encontrar valores de *e*, *d* e *n*, tais que M^{ed} mod *n* = *M* para todo *M* < *n*.
2. É relativamente fácil calcular M^e mod *n* e C^d mod *n* para todos os valores de *M* < *n*.
3. Conhecendo *e* e *n*, é inviável determinar *d*.

Precisamos encontrar um relacionamento na forma M^{ed} mod *n* = *M*. O relacionamento mostrado se mantém se *e* e *d* forem inversos multiplicativos módulo *Φ(n)*, onde *Φ(n)* é a função totiente de Euler. Para *p, q* primos, *Φ(pq) = (p − 1)(q − 1)*. O relacionamento entre *e* e *d* pode ser expresso como

$$ed \bmod \Phi(n) = 1$$

Isso é equivalente a dizer

$$ed \equiv 1 \bmod \Phi(n)$$

$$d \equiv e^{-1} \bmod \Phi(n)$$

Ou seja, *e* e *d* são inversos multiplicativos mod *Φ(n)*. Observe que, de acordo com as regras da aritmética modular, isso é verdadeiro somente se *d* (e, portanto, *e*) for relativamente primo de *Φ(n)*. De modo equivalente, mdc(-*Φ(n)*, *d*) = 1. Agora é possível formular o RSA, a partir dos itens:

p, q, dois números primos	(privados, escolhidos)
n = pq	(público, calculado)
e, com mdc(*Φ(n)*, *e*) = 1; 1 < *e* < *Φ(n)*	(público, escolhido)
$d \equiv e^{-1} \pmod{\Phi(n)}$	(privado, calculado)

A chave privada consiste em {*d, n*}, e a chave pública, em {*e, n*}. Suponha que o usuário A tenha publicado sua chave pública e que o usuário B queira enviar a mensagem M para A. Então, B calcula C = M^e mod *n* e transmite C. Ao receber esse texto cifrado, o usuário A decripta calculando *M* = C^d mod *n.*

A Figura 14 resume o algoritmo RSA: Alice gera um par de chaves pública/privada; Bob encripta usando a chave pública de Alice; e Alice decripta usando sua chave privada. Um exemplo, de Singh (SINGH,2001), aparece na Figura 15. Para ele, as chaves foram geradas da seguinte forma:

1. Selecione dois números primos, *p* = 17 e *q* = 11.

2. Calcule $n = pq = 17 \times 11 = 187$.

3. Calcule $\Phi(n) = (p - 1)(q - 1) = 16 \times 10 = 160$.

4. Selecione e, tal que e seja relativamente primo de $\Phi(n) = 160$ e menor que $\Phi(n)$; escolhemos $e = 7$.

5. Determine d, tal que de $\equiv 1 \pmod{160}$ e $d < 160$. O valor correto é $d = 23$, pois $23 \times 7 = 161 = (1 \times 160) + 1$;

d pode ser calculado usando o algoritmo de Euclides estendido.

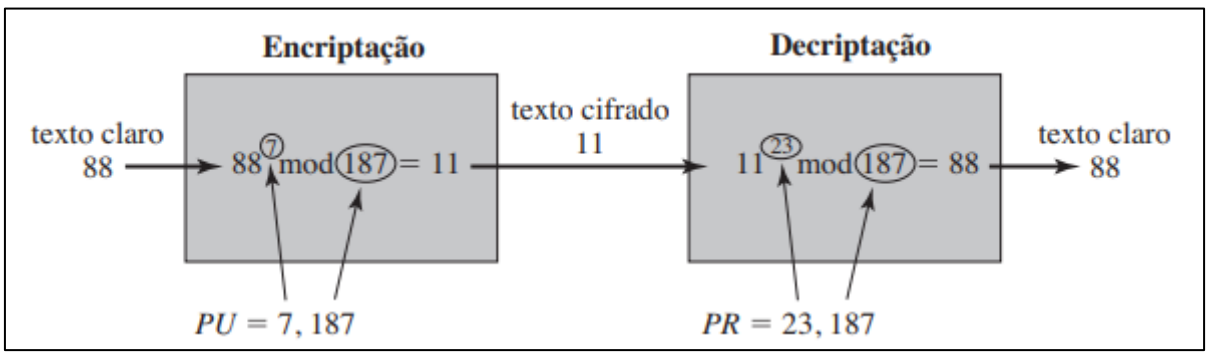
As chaves resultantes são a pública PU = {7, 187} e a privada PR = {23, 187}. O exemplo mostra o uso dessas chaves para uma entrada de texto claro $M = 88$. Para a encriptação, temos que calcular $C = 88^7 \mod 187$.

Figura 14 – O algoritmo RSA

Geração de chave por Alice	
Selecione p, q	p e q são primos, $p \neq q$
Calcule $n = p \times q$	
Calcule $\phi(n) = (p-1)(q-1)$	
Selecione o inteiro e	$\text{mdc}(\phi(n), e) = 1; 1 < e < \phi(n)$
Calcule d	$d \equiv e^{-1} \pmod{\phi(n)}$
Chave pública	$PU = \{e, n\}$
Chave privada	$PR = \{d, n\}$
Encriptação por Bob com chave pública de Alice	
Texto claro:	$M < n$
Texto cifrado:	$C = M^e \bmod n$
Decriptação por Alice com a chave privada de Alice	
Texto cifrado:	C
Texto claro:	$M = C^d \bmod n$

Fonte: Adaptado de Stallings (2014)

Figura 15 – Exemplo de algoritmo RSA



Fonte: Adaptado de Stallings (2014)

Explorando as propriedades da aritmética modular, podemos fazer isso da seguinte forma:

$88^7 \bmod 187 = [(88^4 \bmod 187) \times s88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$

$88^1 \bmod 187 = 88$

$88^2 \bmod 187 = 7744 \bmod 187 = 77$

$88^4 \bmod 187 = 59.969.536 \bmod 187 = 132$

$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894.432 \bmod 187 = 11$

Para a decriptação, calculamos $M = 11^{23} \bmod 187$:

$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times$
$\times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$

$11^1 \bmod 187 = 11$

$11^2 \bmod 187 = 121$

$11^4 \bmod 187 = 14.641 \bmod 187 = 55$

$11^8 \bmod 187 = 214.358.881 \bmod 187 = 33$

$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187$
$= 79.720.245 \bmod 187 = 88$

Agora, vejamos um exemplo de Hellman (HELLMAN, 1970), que mostra o uso do RSA para processar vários blocos de dados. Neste exemplo simples, o texto claro é uma sequência alfanumérica. Cada símbolo do texto claro recebe um código exclusivo de dois dígitos decimais (por exemplo, a = 00, A = 26). Um bloco de texto claro consiste em quatro dígitos decimais, ou dois caracteres alfanuméricos. A Figura 16a ilustra a sequência de eventos para a encriptação de vários blocos, e a Figura 16b é um exemplo específico. Os números circulados indicam a ordem em que as operações são realizadas (STALLINGS, 2014).

2.3.6.1 Segurança do RSA

Segundo Stallings (STALLINGS, 2014), cinco técnicas possíveis para atacar o algoritmo RSA são as seguintes:

Força bruta: isso envolve tentar todas as chaves privadas possíveis.

Ataques matemáticos: existem várias técnicas, todas equivalentes em esforço a fatorar o produto de dois primos.

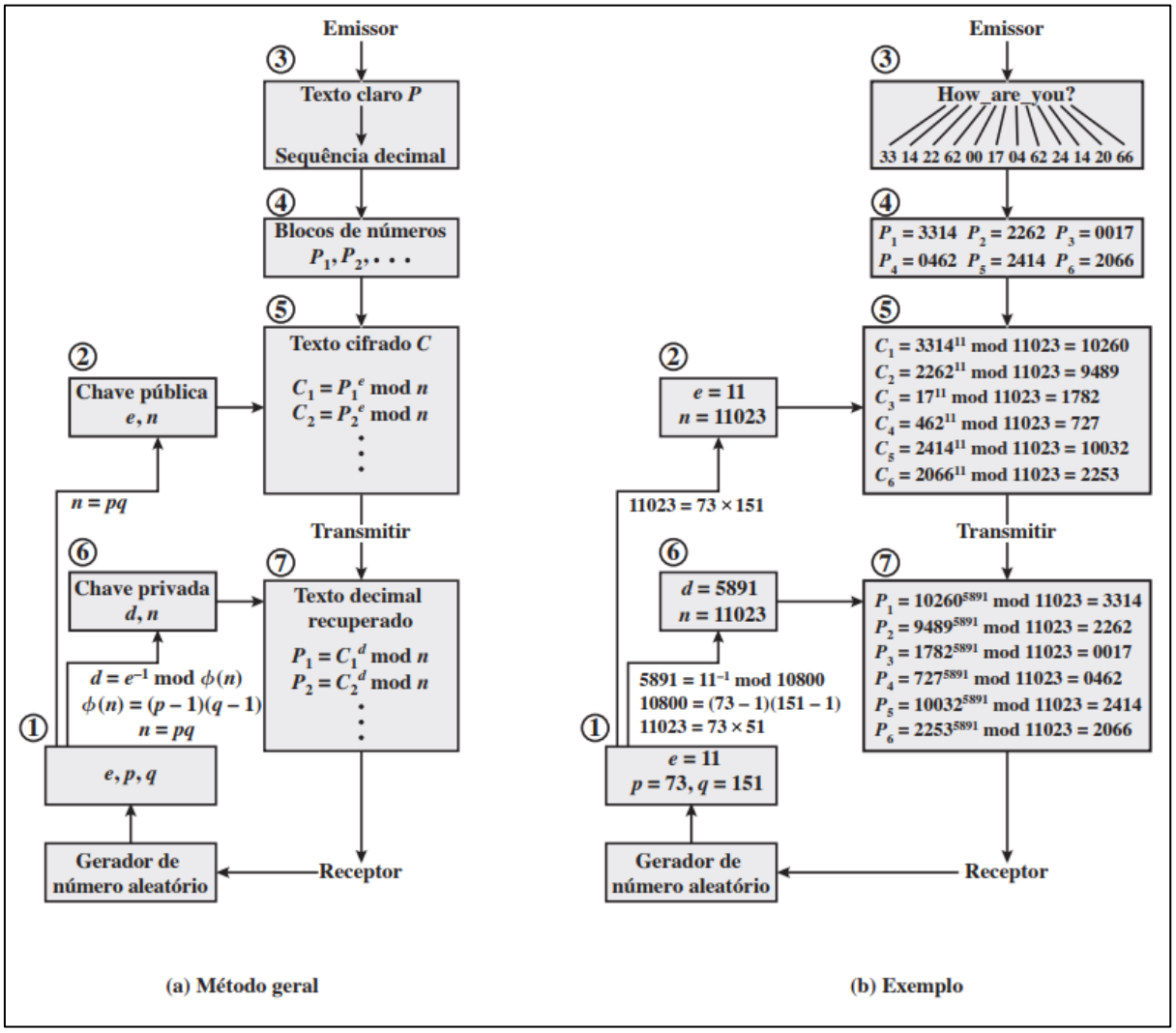
Ataques de temporização: estes dependem do tempo de execução do algoritmo de decriptação.

Ataques baseados em falha de hardware: estes envolvem a indução de falhas de hardware no processador que está gerando as assinaturas digitais.

Ataques de texto cifrado escolhido: esse tipo de ataque explora as propriedades do algoritmo RSA.

O autor explica que (STALLINGS, 2014) a defesa contra a técnica de força bruta é a mesma para o RSA e para outros criptossistemas, ou seja, usar um espaço de chave grande. Assim, quanto maior o número de bits em *d*, melhor. Porém, por conta da complexidade dos cálculos envolvidos, tanto na geração da chave como na encriptação/decriptação, quanto maior o tamanho da chave, mais lento o sistema será.

Figura 16 – Processamento de múltiplos blocos com RSA



Fonte: Adaptado de Stallings (2014)

2.3.6.2 O problema da fatoração

Stallings (STALLINGS, 2014) diz que podemos identificar três técnicas para atacar o RSA matematicamente:

1 - Fatorar n em seus dois fatores primos. Isso permite o cálculo de $\Phi(n) = (p - 1) \times (q - 1)$, que, por sua vez, admite determinar $d \equiv e^{-1} \pmod{\Phi(n)}$.

2. Definir $\Phi(n)$ diretamente, sem estabelecer p e q primeiro. De novo, isso permite a determinação de $d \equiv e^{-1} \pmod{\Phi(n)}$.

3. Determinar d diretamente, sem delimitar $\Phi(n)$ primeiro.

O autor diz que para um n grande, com fatores primos grandes, fatorar é um problema difícil, mas não tanto difícil quanto antes. Em 1977, os três inventores do RSA desafiaram os leitores da Scientific American a decodificarem uma cifra que eles divulgaram na coluna “Mathematical Games” (jogos matemáticos), de Martin Gardner (GARDNER, 1977). Eles ofereceram uma recompensa de US$ 100 para o retorno de uma sentença em texto claro, um evento que eles previam que não poderia acontecer por cerca de 40 quadrilhões de anos. Em abril de 1994, um grupo trabalhando pela Internet reivindicou o prêmio depois de oito meses de esforço (LEUTWYLER, 1994). Esse desafio usava uma chave pública com tamanho (de n) de 129 dígitos decimais, ou cerca de 428 bits. Nesse meio-tempo, assim como haviam feito para o DES, a RSA Laboratories lançou desafios para a cifra RSA com tamanhos de chave de 100, 110, 120 dígitos, e assim por diante. O desafio mais recente (na data de 2014) a ser atendido é o RSA-768, com um tamanho de chave de 232 dígitos decimais, ou cerca de 768 bits. A Tabela 7 mostra os resultados até agora (2014). O nível de esforço é medido em MIPS-anos: um processador de um milhão de instruções por segundo rodando por um ano, que corresponde a cerca de 3 × 1013 instruções executadas. Um Pentium de 1 GHz é uma máquina com cerca de 250 MIPS.

Tabela 7 – Progresso na fatoração RSA

NÚMERO DE DÍGITOS DECIMAIS	NÚMERO DE BITS	DATA EM QUE FOI ALCANÇADO
100	332	abril de 1991
110	365	abril de 1992
120	398	junho de 1993
129	428	abril de 1994

130	431	abril de 1996
140	465	fevereiro de 1999
155	512	agosto de 1999
160	530	abril de 2003
174	576	dezembro de 2003
200	663	maio de 2005
193	640	novembro de 2005
232	768	dezembro de 2009

Fonte: Adaptado de Stallings (2014)

3. METODOLOGIA DA PESQUISA

3.1 Definição e Tipo da Pesquisa

Este trabalho de conclusão de curso tem como objetivo principal verificar a resistência dos algoritmos RSA e AES ante ataques de força bruta. Para alcançar esse objetivo, será conduzida uma pesquisa empírica e exploratória, utilizando métodos quantitativos para avaliar o desempenho e a segurança dos algoritmos mencionados. A pesquisa empírica envolve a coleta de dados através de experimentos práticos, enquanto a abordagem exploratória visa entender o comportamento dos algoritmos RSA e AES quando submetidos a ataques de força bruta. A pesquisa quantitativa permitirá a medição objetiva do tempo necessário para que o algoritmo teste diferentes quantidades de chaves.

3.2 Delineamento da pesquisa

A metodologia adotada para esta pesquisa pode ser detalhada da seguinte forma:
Teste com um algoritmo em JavaScript:

- Será utilizado um algoritmo em JavaScript para gerar e testar uma certa quantidade de chaves nos algoritmos RSA e AES.
- Este algoritmo gerará um número especificado de chaves aleatórias e medirá o tempo necessário para testá-las.
- A análise será realizada para diferentes tamanhos de chaves, tanto para RSA (1024 e 2048 bits) quanto para AES (128, 192 e 256 bits).

Configuração dos Experimentos:

- Serão definidos cenários de teste variando o número de chaves a serem testadas e o tamanho das chaves.
- O tempo de processamento será registrado em milissegundos para cada tentativa.
- O computador utilizado possui a configuração ... Ryzen 5 5600x 3.7GHz, 64gb ram ddr 4 dual channel, placa de vídeo Rx 6600 xt, placa mãe Steel legends.

Execução dos Experimentos:

- O algoritmo será executado múltiplas vezes para cada configuração de chave, a fim de garantir a precisão e a consistência dos resultados.

- Serão coletados dados suficientes para realizar uma análise estatística.

Análise dos Dados:

- Os tempos de execução serão analisados para identificar padrões e tendências na resistência dos algoritmos RSA e AES quanto ao tamanho das chaves.
- Comparações serão feitas entre os diferentes tamanhos de chave para avaliar a escalabilidade e a segurança dos algoritmos.

Interpretação dos Resultados:

- Os resultados serão interpretados para determinar a viabilidade de ataques de força bruta contra RSA e AES.
- Discussões sobre as implicações dos tempos para geração de chaves observados serão realizadas, destacando a segurança prática dos algoritmos nos contextos atuais de poder computacional.

A escolha de JavaScript como linguagem de implementação se deve à sua ampla utilização e facilidade de execução em diferentes plataformas, permitindo uma análise flexível e acessível. Os resultados obtidos fornecerão dados valiosos sobre a eficácia dos algoritmos RSA e AES contra ataques de força bruta e contribuirão para o entendimento da segurança desses métodos de criptografia em cenários de uso real.

4. DESCRIÇÃO E ANÁLISE DOS RESULTADOS

Abaixo os resultados dos testes realizados para verificar o comportamento dos algoritmos RSA e AES em relação ao aumento do tamanho de chaves, levando em consideração o tempo para gerar e testar chaves de acordo com o tamanho. Esses testes proporcionam uma visão da segurança dos algoritmos com relação ao tamanho das chaves.

4.1 Testes no RSA

Os testes com o algoritmo RSA foram realizados utilizando chaves de 1024 e 2048 bits. As tabelas a seguir apresentam os tempos obtidos através dos testes, a média dos resultados e a proporção do aumento de tempo com o aumento do tamanho das chaves.

Geração e teste de chaves no RSA

N° Teste	Tamanho da chave	Tempo (ms)	Quantidade de chaves
1	1024	29763	10000
2	1024	30073	10000
3	1024	29882	10000
4	1024	29621	10000
5	1024	2860	1000
6	1024	2778	1000
7	1024	3012	1000
8	1024	2947	1000
9	1024	373	100
10	1024	362	100
11	1024	375	100
12	1024	359	100
13	1024	94	10
14	1024	73	10
15	1024	78	10

16	1024	80	10
17	1024	82	10
18	2048	152736	10000
19	2048	153752	10000
20	2048	155832	10000
21	2048	152481	10000
22	2048	15187	1000
23	2048	15480	1000
24	2048	16114	1000
25	2048	15020	1000
26	2048	7785	500
27	2048	6882	500
28	2048	7350	500
29	2048	6943	500
30	2048	4182	250
31	2048	4230	250
32	2048	3975	250
33	2048	4070	250
34	2048	1707	100
35	2048	2246	100
36	2048	1865	100
37	2048	1783	100
38	2048	221	10
39	2048	227	10
40	2048	222	10
41	2048	234	10

Média de tempo para geração e teste de chaves no RSA

Tamanho da chave	**Tempo médio (ms)**	**Quantidade de chaves**
1024	29834,75	10000
1024	2899,25	1000

1024	367,25	100
1024	81,25	10
2048	153700,25	10000
2048	15450,25	1000
2048	7240	500
2048	4114,25	250
2048	1900,25	100
2048	226	10

Aumento de tempo em relação ao tamanho da chave no RSA – 1024 para 2048

Qtd. Chaves	Proporção
10000	5,151719052:1
1000	5,329050617:1
100	5,17426821:1
Média	5,21834596:1

Tempo estimado para gerar e testar todas as chaves possíveis no algoritmo

1024	segundos	$\approx 5{,}662 \times 10^{305}$
	dias	$\approx 6{,}554 \times 10^{300}$
	anos	$\approx 1{,}795 \times 10^{298}$
2048	segundos	$\approx 5{,}175 \times 10^{614}$
	dias	$\approx 5{,}989 \times 10^{609}$
	anos	$\approx 1{,}641 \times 10^{607}$

4.2 Testes no AES

Os testes com o algoritmo AES foram realizados utilizando chaves de 128, 192 e 256 bits. As tabelas a seguir apresentam os tempos obtidos, o tempo médio necessário para gerar e testar diferentes quantidades de chaves e a proporção do aumento do tempo com relação ao aumento do tamanho da chave.

Geração e teste de chaves no AES

N° Teste	Tamanho da chave	Tempo milisegundos	Quantidade de chaves
1	256	2017,9	1.000.000

2	256	1944,7	1.000.000
3	256	1928	1.000.000
4	256	1981,95	1.000.000
5	256	1018,75	500.000
6	256	1001,2	500.000
7	256	1085,85	500.000
8	256	978,3	500.000
9	256	514	250.000
10	256	494,9	250.000
11	256	501,4	250.000
12	256	494,3	250.000
13	256	201,1	100.000
14	256	201,65	100.000
15	256	201,15	100.000
16	256	198,3	100.000
17	192	1458,5	1.000.000
18	192	1460,6	1.000.000
19	192	1450,9	1.000.000
20	192	1458,85	1.000.000
21	192	789,3	500.000
22	192	744,85	500.000
23	192	749,7	500.000
24	192	842,4	500.000
25	192	381,95	250.000
26	192	375,8	250.000
27	192	376,4	250.000
28	192	378,1	250.000
29	192	160,85	100.000
30	192	152,15	100.000
31	192	153,9	100.000
32	192	167,75	100.000

33	128	1033,95	1.000.000
34	128	1013,85	1.000.000
35	128	1015,1	1.000.000
36	128	1017,65	1.000.000
37	128	526,5	500.000
38	128	532,3	500.000
39	128	523,6	500.000
40	128	509,95	500.000
41	128	265,25	250.000
42	128	257,35	250.000
43	128	257,55	250.000
44	128	259,05	250.000
45	128	103,05	100.000
46	128	102,9	100.000
47	128	104,8	100.000
48	128	104,55	100.000

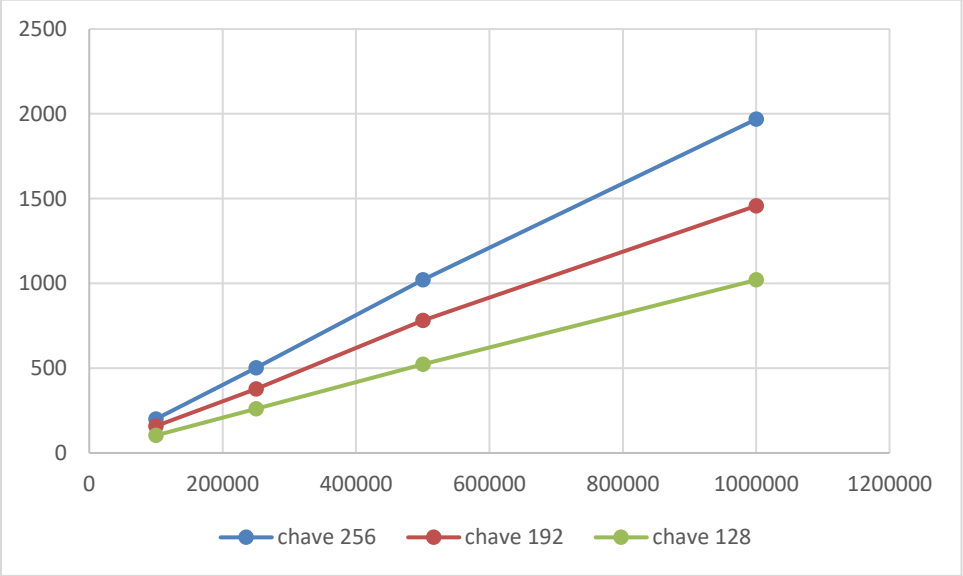
Média de tempo para geração e teste de chaves no AES

Tamanho da chave	Tempo médio (ms)	Quantidade de chaves
128	1020,137	1.000.000
128	523,0875	500.000
128	259,8	250.000
128	103,825	100.000
192	1457,2125	1.000.000
192	781,5625	500.000
192	378,0625	250.000
192	158,6625	100.000
256	1968,1375	1.000.000
256	1021,025	500.000
256	501,15	250.000
256	200,55	100.000

Aumento de tempo em relação ao tamanho da chave no AES – 128 para 256

Qtd. Chaves	Proporção
1000000	1,929286493
500000	1,95192009
250000	1,928983834
100000	1,931615699
Média	1,935451529

Comparação da média de tempo(ms) de geração e teste de chaves no AES



Tempo para gerar todas as chaves possíveis com o algoritmo

128	segundos	≈3507391520882110000000000000000000	≈ 351 x 10^{30}
	dias	≈40594809269468900000000000000	≈ 406 x 10^{25}
	anos	≈111218655532791000000000000	≈ 11 x 10^{24}
192	segundos	≈941734911706256000000000000000000000000000000000000000000000	≈ 942 x 10^{49}
	dias	≈10899709626229800000000000000000000000000000000000000000	≈ 109 x 10^{45}
	anos	≈29862218154054300000000000000000000000000000000000	≈ 299 x 10^{42}
256	segundos	≈2310122594392770000000000000000000000000000000000000000000000000000000000000000	≈ 231 x 10^{69}
	dias	≈2673753002769420000000000000000000000000000000000000000000000000000000000000	≈ 267 x 10^{64}
	anos	≈7325350692518950000000000000000000000000000000000000000000000000000000000000	≈ 732 x 10^{61}

4.3 Comparação dos resultados entre RSA e AES

A análise dos resultados revela diferenças significativas entre os algoritmos RSA e AES, tanto em termos de tempo de geração e teste de chaves quanto em termos de impacto do tamanho da chave. Os tempos médios para a geração de chaves RSA são significativamente mais altos em comparação com AES, especialmente à medida que o tamanho da chave aumenta. Por exemplo, para 10.000 chaves, uma chave RSA de 2048 bits leva em média cerca de 153700,25 ms, enquanto a chave de 1024 bits leva em média 29.834,75 ms para a mesma quantidade, já para o AES-256 é possível gerar e testar 1.000.000 de chaves em 1968,13 ms e para o AES-128, é possível gerar e testar 1.000.000 de chaves em 1020,13 ms de acordo com as tabelas de média dos resultados.

No RSA, o aumento do tamanho da chave de 1024 bits para 2048 bits resulta em um aumento substancial no tempo necessário para gerar e testar as chaves, aproximadamente 5,2 vezes maior. Isso se deve à complexidade matemática subjacente ao algoritmo RSA, que envolve operações de grande complexidade como a fatoração de grandes números primos.

No AES, embora o aumento no tamanho da chave de 128 para 256 bits também aumente o tempo de geração de chaves, esse aumento é proporcionalmente menor quando comparado ao RSA, aproximadamente 1,935 vezes. Isso ocorre porque o AES é um algoritmo de criptografia simétrica que realiza operações de substituição e permutação, que são menos complexas computacionalmente do que as operações do RSA.

5. CONSIDERAÇÕES FINAIS

5.1 Conclusão e recomendações

A análise realizada neste trabalho revela insights significativos sobre a resistência dos algoritmos RSA e AES a ataques de força bruta. Durante os testes, observou-se que o tempo necessário para gerar e testar uma quantidade específica de chaves aumenta exponencialmente com o aumento do tamanho das chaves. No caso do RSA, os tempos médios para o teste de 10.000 chaves de 1024 bits foram em torno de 29.834,75 ms, enquanto para 10.000 chaves de 2048 bits, o tempo médio subiu para 153.700,25 ms. Já o AES demonstrou tempos médios mais curtos em comparação com o RSA, embora o aumento do tamanho das chaves também tenha influenciado significativamente. Por exemplo, para o AES com chave de 256 bits, o tempo médio para o teste de 1.000.000 de chaves foi de 1968,13 ms.

Os resultados claramente indicam que a geração e teste de chaves para algoritmos de criptografia robusta, como RSA e AES, é um processo computacionalmente intensivo, especialmente à medida que o tamanho das chaves aumenta. A comparação entre os dois algoritmos destaca que, enquanto ambos são seguros contra ataques de força bruta, o RSA exige mais tempo de processamento para tamanhos de chave maiores, devido à complexidade matemática subjacente à sua estrutura. Em contrapartida, o AES, apesar de mais eficiente em termos de tempo, também apresenta um crescimento significativo no tempo de geração com o aumento do tamanho da chave, embora de forma menos pronunciada que o RSA.

Este estudo contribui significativamente para a compreensão da segurança e eficiência dos algoritmos RSA e AES, fornecendo dados empíricos que demonstram a inviabilidade de ataques por força bruta, especialmente em cenários onde chaves de tamanho adequado são utilizadas. Os benefícios deste trabalho incluem a consolidação do entendimento sobre a importância de escolher tamanhos de chave apropriados para garantir a segurança dos dados e a eficiência dos processos criptográficos.

No entanto, este estudo apresenta algumas limitações. A principal delas reside nas configurações do computador utilizado para rodar os algoritmos de teste, que não representava a máquina mais potente disponível nos dias de hoje. Isso pode ter influenciado os tempos de processamento observados. Além disso, o próprio

algoritmo utilizado para os testes pode não ter sido otimizado para desempenho máximo, o que também pode ter impactado os resultados.

As possibilidades para novos estudos são vastas. Pesquisas futuras podem explorar o desempenho dos algoritmos em máquinas mais potentes e com diferentes configurações de hardware. Além disso, a otimização dos algoritmos de teste pode fornecer dados ainda mais precisos e relevantes. Outro caminho interessante seria a análise de outros algoritmos de criptografia em comparação com o RSA e o AES, para fornecer um panorama ainda mais abrangente sobre a segurança e eficiência das técnicas criptográficas modernas.

Minha impressão final sobre este processo de pesquisa é extremamente positiva. Através desta investigação, foi possível aprofundar o entendimento sobre a segurança de algoritmos criptográficos essenciais na proteção de dados na era digital. O desafio de analisar e comparar o desempenho dos algoritmos RSA e AES foi recompensador, fornecendo resultados que não apenas validam teorias matemáticas, mas também oferecem dados concretos que podem ser aplicados em práticas de segurança cibernética. Este estudo reafirma a importância da escolha adequada do tamanho das chaves para garantir a segurança dos sistemas de informação e destaca a contínua relevância de pesquisas na área de criptografia. O processo de pesquisa foi enriquecedor e instigante, proporcionando uma visão clara sobre a robustez dos algoritmos estudados e abrindo portas para futuras investigações que possam contribuir ainda mais para a evolução da segurança digital.

REFERÊNCIA BIBLIOGRÁFICA

ASSUNÇÃO, MARCOS FLÁVIO ARAÚJO. **Guia do Hacker Brasileiro**. Florianópolis: Visual Books, 2002.

DAEMEN, JOAN; RIJMEN, VINCENT, **The Design of Rijndael:** The Advanced Encryption Standard (AES) 2°Ed. Germany**:** Springer, 2020

DAEMEN, JOAN; RIJMEN, VINCENT. **AES Proposal:** Rijndael, Version 2. Submission to NIST, mar 1999. Disponível em https://www.researchgate.net/publication/2237728_AES_proposal_rijndael. Acesso em: 10 abr.2024

DIFFIE, W; HELLMAN, M. **"Multiuser Cryptographic Techniques"**. IEEE Transactions on Information Theory, nov 1976

FIGUEIREDO, CANDIDO DE, **Novo Dicionário da Língua Portuguesa.** Portugal: Livraria Clássica, 1913.

GARDNER, M., **A New Kind of Cipher That Would Take Millions of Years to Break**. Scientific American, ago 1977.

HELLMAN, M., **The Mathematics of Public-Key Cryptography**. Scientific American, ago 1970.

LEUTWYLER, K., **Superhack**. Scientific American, jul 1994.

LEVY, STEVEN, **Crypto:** How the Code Rebels Beat the Government. New York: Penguin Publishing Group, 2001.

PRIETO, MANUEL JESUS, **História de la Criptografia.** Madrid: La Esfera de Los Libros, 2020.

RIVEST, R.; SHAMIR, A. e Adleman, L. **"A Method for Obtaining Digital Signatures and Public Key Cryptosystems"**. Communications of the ACM, fev. 1978.

SINGH, SIMON, **The Code Book**. New York: Delacorte Press, 2001.

SCHNEIER, Bruce. **Applied Cryptography**. New York: Wiley, 1996.

STALLINGS, WILLIAM, **Criptografia e segurança de redes:** Princípios e Práticas. São Paulo: Pearson Universidades, 2014.

TKOTZ, VIKTORIA, **Criptografia: Segredos Embalados para Viagem**. São Paulo: Novatec, 2005.

APÊNDICE A

Algoritmo de teste usado para o AES

```
// Importando a biblioteca CryptoJS
const CryptoJS = require("crypto-js");

// Função para criptografar usando AES
function encryptAES(message, key) {
  // Criando o objeto de configuração com a chave
  const config = {
    mode: CryptoJS.mode.ECB, // Modo de operação ECB
    padding: CryptoJS.pad.Pkcs7 // Preenchimento PKCS7
  };

  // Criptografando
  const encrypted = CryptoJS.AES.encrypt(message, key, config);

  // Retornando a mensagem cifrada como string codificada em Base64
  return encrypted.toString();
}

// Função para descriptografar usando AES
function decryptAES(ciphertext, keys) {
  // Criando o objeto de configuração
  const config = {
    mode: CryptoJS.mode.ECB, // Modo de operação ECB
    padding: CryptoJS.pad.Pkcs7 // Preenchimento PKCS7
  };

  // Iterando sobre as chaves
  for (let i = 0; i < keys.length; i++) {
    const key = keys[i];
    // Decifrando com a chave atual
    const decrypted = CryptoJS.AES.decrypt(ciphertext, key, config);
    // Verificando se a decifragem foi bem-sucedida
    try {
      const decryptedMessage = decrypted.toString(CryptoJS.enc.Utf8);
      // Se a decifragem for bem-sucedida, retorna a mensagem decifrada
      return decryptedMessage;
    } catch (error) {
      // Se a decifragem falhar, continua para a próxima chave
      continue;
    }
  }
  // Se nenhuma chave funcionar, lança um erro
  throw new Error("Nenhuma chave válida encontrada.");
}

// Função para gerar chaves aleatórias
```

```
function generateRandomKeys(numKeys, keyLength) {
  const keys = [];
  for (let i = 0; i < numKeys; i++) {
    const key =
CryptoJS.lib.WordArray.random(keyLength/8).toString(CryptoJS.enc.Hex);
    keys.push(key);
  }
  return keys;
}

// Função para gerar uma chave correta
function generateCorrectKey(keyLength) {
  return
CryptoJS.lib.WordArray.random(keyLength/8).toString(CryptoJS.enc.Hex);
}

// Exemplo de uso
const message = "Mensagem secreta";
const keyLength = 256; // Tamanho da chave em bits

// Gerar a chave correta
const correctKey = generateCorrectKey(keyLength);
console.log("Chave correta:", correctKey);

// Iniciar a medição do tempo
const startTime = Date.now();

// Gerar chaves aleatórias para teste
const numKeysToTest = 1000000;
const randomKeys = generateRandomKeys(numKeysToTest, keyLength);

// Adicionar a chave correta à lista de chaves aleatórias
randomKeys.push(correctKey);


// Criptografar a mensagem com a chave correta
const encryptedMessage = encryptAES(message, correctKey);
console.log("Mensagem criptografada:", encryptedMessage);

// Testar várias chaves para descriptografar a mensagem
try {
  const decryptedMessage = decryptAES(encryptedMessage, randomKeys);
  if (decryptedMessage) {
    console.log("Mensagem descriptografada:", decryptedMessage);
  } else {
    throw new Error("Erro ao descriptografar a mensagem: Nenhuma mensagem
decifrada encontrada.");
  }
} catch (error) {
```

```
  console.error(error.message);
}

// Finalizar a medição do tempo
const endTime = Date.now();

// Calcular o tempo decorrido em milissegundos
const elapsedTime = endTime - startTime;
console.log("Tempo decorrido:", elapsedTime, "milissegundos");
```

Apêndice B
Algoritmo utilizado no teste do RSA

```
// biblioteca
const forge = require("node-forge");

// Função para gerar chaves RSA
function generateRSAKeyPair(keySize) {
    const rsa = forge.pki.rsa;
    const keyPair = rsa.generateKeyPair({ bits: keySize });
    return {
      publicKey: forge.pki.publicKeyToPem(keyPair.publicKey),
      privateKey: forge.pki.privateKeyToPem(keyPair.privateKey)
    };
  }

  // Função para criptografar usando RSA
  function encryptRSA(message, publicKey) {
    const rsa = forge.pki.publicKeyFromPem(publicKey);
    const encrypted = rsa.encrypt(message, "RSA-OAEP");
    return Buffer.from(encrypted, "binary").toString("base64");
  }

  // Função para descriptografar usando RSA
  function decryptRSA(ciphertext, privateKey) {
    const rsa = forge.pki.privateKeyFromPem(privateKey);
    const decrypted = rsa.decrypt(Buffer.from(ciphertext,
"base64").toString("binary"), "RSA-OAEP");
    return decrypted;
  }

  // Função para testar várias chaves RSA
  function testRSAKeys(ciphertext, keys) {
    const startTime = Date.now();
    for (let i = 0; i < keys.length; i++) {
      const key = keys[i];
      try {
        const decryptedMessage = decryptRSA(ciphertext, key.privateKey);
```

```
        console.log("Mensagem descriptografada:", decryptedMessage);
        return; // Encerra a função se a mensagem for descriptografada com
sucesso
      } catch (error) {
        // Se a decifragem falhar, continua para a próxima chave
        continue;
      }
    }
    // Se nenhuma chave funcionar, lança um erro
    console.error("Nenhuma chave válida encontrada.");
  }

  // Função para gerar chaves aleatórias RSA
  function generateRandomRSAKeys(numKeys, keySize) {
    const keys = [];
    for (let i = 0; i < numKeys; i++) {
      keys.push(generateRSAKeyPair(keySize));
    }
    return keys;
  }

  // Exemplo de uso
  const message = "Mensagem secreta";
  const numKeysToTest = 500;
  const keySize = 2048; // Tamanho da chave RSA em bits

  // Iniciar a medição do tempo
  const startTime = Date.now();

  // Gerar chaves aleatórias para teste
  const randomKeys = generateRandomRSAKeys(numKeysToTest, keySize);

  // Criar uma chave pública e privada RSA para o exemplo
  const { publicKey, privateKey } = generateRSAKeyPair(keySize);

  // Criptografar a mensagem com a chave pública
  const encryptedMessage = encryptRSA(message, publicKey);
  console.log("Mensagem criptografada:", encryptedMessage);

  // Testar várias chaves para descriptografar a mensagem
  testRSAKeys(encryptedMessage, randomKeys);

  // Calcular o tempo decorrido em milissegundos
  const endTime = Date.now();
  const elapsedTime = endTime - startTime;
  // Exibir o tempo total de teste
  console.log("Tempo total de teste:", elapsedTime, "milissegundos");
```