FACULDADE VANGUARDA

LUCAS BRITO CORDEIRO

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Desenvolvimento de aplicação utilizando técnicas criptográficas

TÉCNICA RIVEST-SHAMIR-ADLEMAN (RSA)

São José dos Campos

2025

LUCAS BRITO CORDEIRO

ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Desenvolvimento de aplicação utilizando técnicas criptográficas

TÉCNICA RIVEST-SHAMIR-ADLEMAN (RSA)

Atividades Práticas Supervisionadas do curso de ENGENHARIA DA COMPUTAÇÃO da FACULDADE VANGUARDA, sob orientação de:

Prof. Responsável: Profa. Dra. Ivana Yoshie Sumida

Coordenador: Prof. MSc. André Yoshimi Kusumoto

São José dos Campos

2025

SUMÁRIO

1 INTRODUÇÃO

Este trabalho acadêmico, do curso de Engenharia da Computação da Faculdade Vanguarda, tem como objetivo principal a aplicação de técnicas criptográficas, com foco específico no algoritmo Rivest-Shamir-Adleman (RSA), contendo os conceitos fundamentais da criptografia, abordando seus tipos, princípios e aplicações gerais. Em seguida, aprofundará na técnica RSA, descrevendo seu funcionamento matemático, os processos de criptografia e descriptografia, suas vantagens, desvantagens, principais aplicações e a complexidade computacional envolvida.

A escolha do tema “Desenvolvimento de aplicação utilizando técnicas criptográficas” para a Atividade Prática Supervisionada (APS) visa promover um contato dos fundamentos da segurança da computação e à prática da programação aplicada a problemas reais aos estudantes.

2 FUNDAMENTOS DE CRIPTOGRAFIA

Maziero e Lima (2013) conceituam a criptografia como uma disciplina antiga, intrinsecamente ligada à necessidade humana de comunicação segura. Em sua essência, pode ser definida como o conjunto de técnicas utilizadas para transformar uma informação original, denominada texto claro, em uma forma ininteligível, chamada texto cifrado, de modo que apenas o destinatário, de posse de uma chave secreta, consiga reverter o processo e recuperar a informação original. Este processo de transformação é conhecido como cifragem ou encriptação, enquanto o processo inverso é a decifragem ou decriptação.

De acordo com Pelzl (2010), o principal objetivo da criptografia é garantir a confidencialidade da informação, assegurando que apenas as partes autorizadas tenham acesso ao seu conteúdo. No entanto, para Stallings (2017) as técnicas criptográficas modernas vão além, oferecendo mecanismos para garantir outros pilares da segurança da informação, como a integridade (garantia de que a mensagem não foi alterada durante a transmissão), a autenticidade (verificação da identidade do remetente) e o não repúdio (impossibilidade de o remetente negar a autoria da mensagem).

Historicamente, como confirma Maziero, os métodos criptográficos eram predominantemente simétricos, também conhecidos como criptografia de chave secreta. Nesses sistemas, a mesma chave é utilizada tanto para cifrar quanto para decifrar a mensagem. Um exemplo clássico é a Cifra de César. A grande vantagem dos algoritmos simétricos reside na sua eficiência computacional, sendo geralmente muito mais rápidos que os algoritmos assimétricos. Contudo, seu principal desafio é a necessidade de um canal seguro para a distribuição e o gerenciamento das chaves secretas entre as partes comunicantes. Se a chave for interceptada por um adversário durante a distribuição, toda a segurança do sistema é comprometida.

A revolução na criptografia ocorreu na década de 1970 com a introdução do conceito de criptografia assimétrica, ou de chave pública. Proposta por Diffie e Hellman (1976), essa abordagem utiliza um par de chaves relacionadas por um algoritmo: uma

chave pública, que pode ser livremente distribuída, e uma chave privada, que deve ser mantida em segredo pelo seu proprietário. Uma mensagem cifrada com a chave pública só pode ser decifrada com a chave privada correspondente, e vice-versa. Essa situação resolve o problema da distribuição de chaves, pois a chave pública pode ser enviada por canais inseguros sem comprometer a segurança.

O algoritmo RSA, criado por Rivest, Shamir e Adleman (1978) é o exemplo mais conhecido e amplamente utilizado de criptografia assimétrica, como indica Castro (2014). Para Sousa (2013), sua segurança baseia-se na dificuldade computacional de fatorar números inteiros grandes, um problema matemático considerado extremamente complexo para os computadores atuais quando se utilizam números suficientemente grandes.

Além dos algoritmos de cifragem, Maziero confirma que as funções de *hash* criptográfico são ferramentas essenciais na criptografia moderna. Uma função de *hash* gera uma saída de tamanho fixo a partir de uma entrada de tamanho variável. Essas funções são projetadas para serem computacionalmente inviáveis de encontrar a entrada original a partir do *hash* e é muito difícil encontrar duas entradas diferentes que produzam o mesmo *hash*.

Para Stallings (2017) aplicações da criptografia são vastas e podem ser encontradas em quase todos os aspectos da vida digital contemporânea. Desde a proteção de comunicações na internet, transações financeiras, armazenamento seguro de senhas e até na urna eletrônica. A criptografia é a tecnologia fundamental que sustenta a confiança e a segurança no mundo digital.

3 TÉCNICA CRIPTOGRÁFICA ESCOLHIDA

A técnica criptográfica selecionada para aplicação neste trabalho é o algoritmo Rivest-Shamir-Adleman, amplamente conhecido pela sigla RSA. Desenvolvido em 1977 por Ron Rivest, Adi Shamir e Leonard Adleman no Instituto de Tecnologia de Massachusetts (MIT), o RSA representou um marco na história da criptografia, sendo de acordo com Sousa (2013), o primeiro sistema de chave pública funcional e seguro a ser publicamente descrito, adequado tanto para cifragem quanto para assinaturas digitais.

Para Castro (2014) a genialidade e a segurança do RSA residem em sua fundamentação matemática, especificamente na dificuldade computacional de fatorar números inteiros grandes. Enquanto a multiplicação de dois números primos grandes é uma operação relativamente simples e rápida para um computador, o processo inverso de encontrar os fatores primos originais de um número grande que é produto de dois primos, é considerado um problema intratável com os algoritmos e o poder computacional atualmente disponíveis, especialmente quando os primos envolvidos possuem centenas de dígitos. Isso é a base da segurança do RSA.

O funcionamento do RSA envolve três etapas principais: geração de chaves, cifragem e decifragem.

1. **Geração de Chaves:** O processo inicia com a escolha de dois números primos distintos e grandes, denominados p e q. Calcula-se então o módulo n, que é o produto de p e q ($n = p*q$). O valor de n fará parte tanto da chave pública quanto da privada. Em seguida, calcula-se a função totiente de Euler de n, representada por $\varphi(n)$, que para o produto de dois primos é dada por $\varphi(n) = (p-1)*(q-1)$. O próximo passo é escolher um número inteiro e (expoente público) tal que $1 < e < \varphi(n)$ e que o máximo divisor comum entre e e $\varphi(n)$ seja 1. A chave pública é então formada pelo par (n, e). Finalmente, calcula-se o expoente privado d, que é o inverso modular de e em relação a $\varphi(n)$. Isso significa que d deve satisfazer a congruência $e*d = 1 \pmod{\varphi(n)}$. A chave privada é o par (n, d). Os números primos p e q, bem como $\varphi(n)$, devem ser mantidos em

segredo e descartados após a geração das chaves (LIMA, 2013, p. 20-22; ASSIS, 2024, p. 15-17).

2. **Cifragem:** Para cifrar uma mensagem *M* (representada como um número inteiro tal que $0 \leq M < n$), o remetente utiliza a chave pública (*n, e*) do destinatário. O texto cifrado *C* é obtido pela potenciação modular: $C = M^e \pmod{n}$. Qualquer pessoa pode realizar essa operação, pois a chave pública é de conhecimento geral (CASTRO, 2014, p. 26).

3. **Decifragem:** Para decifrar o texto cifrado *C*, o destinatário utiliza sua chave privada (*n, d*). A mensagem original *M* é recuperada através da operação: $M = C^d \pmod{n}$. A correção matemática do processo é garantida pelo Teorema de Euler e pelas propriedades da aritmética modular, assegurando que $(Me)d \equiv M \pmod{n}$ (LIMA, 2013, p. 23; SOUSA, 2013, p. 20).

Segundo Maziero, as vantagens do RSA são significativas. Primeiramente, ele resolve o problema da distribuição segura de chaves, em comparação aos sistemas simétricos, ao utilizar chaves públicas. Além disso, permite a implementação de assinaturas digitais, onde o remetente cifra um *hash* da mensagem com sua chave privada, e qualquer pessoa pode verificar a autenticidade usando a chave pública do remetente, garantindo autenticidade. Sua ampla adoção e estudo ao longo de décadas conferem-lhe muita confiança pela comunidade de segurança.

Contudo, o RSA também apresenta desvantagens. Segundo Stallings (2017), a principal é sua lentidão computacional em comparação com algoritmos simétricos. As operações de exponenciação modular são significativamente mais custosas do que as operações tipicamente usadas em cifras simétricas. Por essa razão, o RSA é frequentemente utilizado em sistemas híbridos: um algoritmo simétrico rápido é usado para cifrar a maior parte dos dados com uma chave de sessão aleatória, e o RSA é usado apenas para cifrar e transmitir essa chave de sessão de forma segura. Outra desvantagem para Assis (2024) é a necessidade de chaves relativamente longas (atualmente, 2048 bits ou mais são recomendados) para garantir a segurança contra os

ataques baseados na fatoração, o que aumenta o custo computacional e o tamanho das mensagens.

Assis (2024) garante que a complexidade e a segurança do RSA estão diretamente ligadas ao tamanho da chave (o número de bits do módulo *n*). Aumentar o tamanho da chave torna a fatoração de *n* exponencialmente mais difícil, elevando a segurança, mas também aumenta o custo computacional das operações de cifragem e decifragem. A escolha do tamanho da chave envolve, portanto, um equilíbrio entre o nível de segurança desejado e o desempenho aceitável para a aplicação específica.

4 SOLUÇÃO DESENVOLVIDA

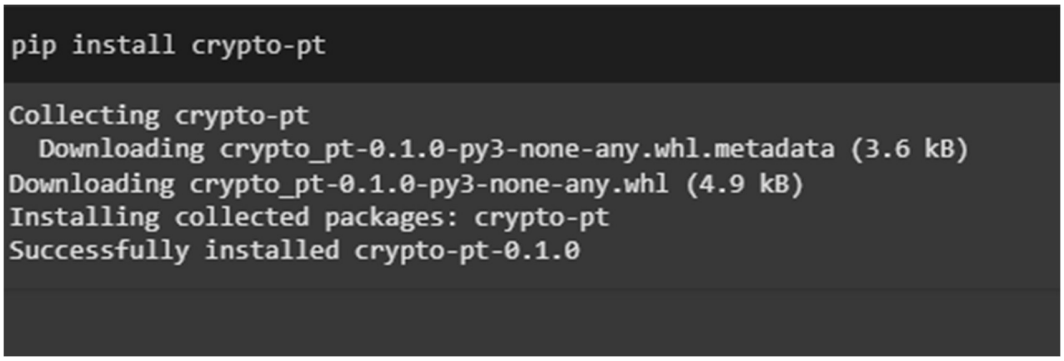
Foi desenvolvida uma biblioteca na liguagem Python que foi publicada no PyPi, com o nome de *crypto-pt*. Essa biblioteca faz a criptografia e descriptografia de *strings* (textos na linguagem Python) baseada na técnica RSA.

Ela possui funções para gerar uma chave pública e outra chave privada com tamanho de bits personalizável, faz a criptografia de strings e descriptografa mensagens criptografadas. A biblioteca pode ser usada em qualquer computador que tenha o Python 3 instalado, usando o comando *pip install crypto-pt.*

Figura 01 – Biblioteca crypto-pt no Pypi

Fonte: PyPi, 2025

Figura 02 – Exemplo de instalação da biblioteca



Fonte: Autoria própria, 2025

Para desenvolver as funções que realizam a criptografia RSA foi seguida a descrição do modelo matematicamente com todas as fórmulas e testes exigidos. Além disso foi criada uma função para gerar dois números primos com a possibilidade de o usuário da biblioteca escolher a quantidade de bits a ser usada nesses números, caso o usuário não escolha, por padrão é usado 512 bits.

Para que o sistema verifique se o número aleatório que ele gerou para os cálculos é um número primo, foi usado o algoritmo de Miller-Rabin, que segundo descreve Conrad (2011), consiste em iterar uma sequência de cálculos em um número *n* por *k* vezes, quanto mais vezes fizer essa sequência de cálculos mais provável do número ser primo. No caso do algoritmo desenvolvido para a *crypto-pt*, ele faz essa sequência 5 vezes, isso não pode ser modificado pelo usuário da biblioteca. Todos os testes têm que retornar um resultado igual à 1 ou à *n-1* para que o número ímpar seja considerado primo.

A escolha do algoritmo de Miller-Rabin se dá pois com algoritmos simples não é possível verificar a primalidade de um número com 512 bits, por exemplo, mesmo que não seja 100% assertivo é a maneira mais prática e confiável que existe para tal.

Após isso, os números primos gerados passam pela sequência de cálculos descritas no item anterior para gerar a criptografia (chave pública) da mensagem (chave privada) passada. A mensagem é decodificada em uma sequência de bytes que depois é transformada em um número inteiro de tamanho fixo, que passará pela fórmula da criptografia, para que a chave pública seja um único número. Na descriptografia o processo inverso é feito para devolver a mensagem original como uma string.

Na sequência foi desenvolvida uma pequena aplicação que solicita ao usuário digitar uma mensagem a ser criptografada, a mensagem deve ter no máximo 256 caracteres. Após isso o programa gera a chave pública da mensagem e pede para que o usuário insira essa chave para que ele retorne a chave privada, ou seja, a mensagem original. Tudo isso importando a biblioteca que foi criada anteriormente.

Figura 03 - Sistema em funcionamento

Fonte: autoria própria

Essa aplicação resolve o problema proposto, criptografando uma mensagem de até 256 caracteres e gerando a descriptografia da mesma baseada na técnica de criptografia assimétrica RSA. Após amplos estudos sobre a técnica foi desenvolvido um código simples que consegue aplicar todas as teorias propostas e retornar uma chave criptografada corretamente, que é segura para ser usada no mundo real e pode ser aproveitada em diversos projetos futuros, já que a biblioteca foi publicada no PyPi e com isso pode ser instalada por qualquer computador com o Python 3 instalado no mundo.

5 CÓDIGO-FONTE

CÓDIGO FONTE DO ARQUIVO *crypto-pt.py*, QUE CONTEM AS FUNÇÕES DA BIBLIOTECA:

```
import random

# criação de funções para realizar a criptografia e descriptografia


def gerar_numero_aleatorio(bits): # função gera um número aleatório com um número de bits que passar
    return random.getrandbits(bits) | (1 << bits - 1) | 1 # garante que o número é ímpar e tem o tamanho desejado


def miller_rabin(n): # usar Miller-Rabin para testar primalidade
  if n <= 1 or n % 2 == 0: # excluir de cara números pares e negativos, e 1 ou 0
    return False
  k = 5 # atribuir um valor fixo para k
  s = 0 # atribuir o contador
  d = n-1 # atribuir o valor de d como n-1
  while d % 2 == 0: # dividir o número s vezes até que o resultado seja ímpar
    d = d // 2
    s += 1 # contar quantas vezes foi feita a divisão


  for repeticao in range(k): # repetir a operação k vezes para garantir a confiabilidade
    a = random.randrange(2, n-1) # atribuir um número aleatorio entre 2 e n-1 para a
    if pow(a, d, n) == 1 or pow(a, d, n) == n-1: # fazer a potenciação modular, se der 1 ou n-1 ele continua para a próxima repetição
      continue
    for x in range(s-1): # se der errado ele entra nesse loop que repete a operação s - 1 vezes
```

```
    r = pow(a, 2, n) # potenciação modular de a ao quadrado dividida pelo módulo de n
    if r == n-1:
      break # se der n-1 n provavelmente é primo e seguimos para a próxima repetição
  else:
    return False

 return True


def inverso_modular(e, phi): # função para encontrar o inverso modular usando o algoritmo extendido de Euclides
  d, x1, x2, y1 = 0, 0, 1, 1
  temp_phi = phi

  while e > 0:
    q = temp_phi // e
    temp1 = temp_phi - q * e
    temp_phi, e = e, temp1

    x = x2 - q * x1
    y = d - q * y1

    x2, x1 = x1, x
    d, y1 = y1, y

  return d + phi if d < 0 else d


def gerar_primo(bits): # função que gera um número primo, usando as duas funções atribuidas anteriormente
```

```
    while True:
        candidato = gerar_numero_aleatorio(bits) # chama a função que gera um número
        if miller_rabin(candidato): # chama a função que testa se o número é primo
            return candidato


def gerar_chaves_grandes(bits=512): # função que realiza todos os cálculos para a criptografia passando o número de bits por padrão definido como 512
    p = gerar_primo(bits) # gerar número primo 1
    q = gerar_primo(bits) # gerar número primo 2
    while q == p: # garantindo que são números diferentes
        q = gerar_primo(bits)


    n = p * q # cálculo do módulo
    phi = (p - 1) * (q - 1) # cálculo do Totiente de Euller
    e = 65537  # atribuindo fixamente um valor comum e seguro para a chave


    from math import gcd
    if gcd(e, phi) != 1: # garantindo que e e phi são primos entre sim
        raise Exception("e e phi não são primos entre si") # erro caso não seja


    d = inverso_modular(e, phi)
    return ((e, n), (d, n)) # retornando valores que serão usados na fórmula da criptografia


def criptografar(mensagem, chave_publica): # função para criptografar a string
    e, n = chave_publica
    mensagem_bytes = mensagem.encode('utf-8') # transformar a string em uma sequência de bytes
    mensagem_numerica = int.from_bytes(mensagem_bytes, byteorder='big') # transformar a sequência de bytes em um número inteiro
```

```
    if mensagem_numerica >= n:
        raise Exception("Mensagem muito longa para essa chave pública com essa quantidade de bits. Tente diminuir a mensagem ou aumentar a quantidade de bits") # erro, pois o número não pode ser maior ou igual a n
    criptografada = pow(mensagem_numerica, e, n) # calcula c = m^e % n
    return criptografada


def descriptografar(mensagem_criptografada, chave_privada): # função para descriptografar a mensagem
    d, n = chave_privada


    mensagem_numerica = pow(mensagem_criptografada, d, n) # calcula m = c^d % n
    mensagem_bytes = mensagem_numerica.to_bytes((mensagem_numerica.bit_length() + 7) // 8, byteorder='big') # retorna o número para uma sequência de bytes


    return mensagem_bytes.decode('utf-8') # retorna a sequência de bytes para a string
```

CÓDIGO FONTE DO ARQUIVO *__init__.py*, QUE FOI USADO PARA PUBLICAR A BIBLIOTECA, INDICANDO QUAIS FUNÇÕES PODEM SER CHAMADAS POR QUEM FIZER A INSTALAÇÃO DA MESMA:

```
from .core import gerar_chaves_grandes, criptografar, descriptografar


__version__ = "0.1.0"  # versão da biblioteca crypto_pt
__all__ = ['gerar_chaves_grandes', 'criptografar', 'descriptografar'] # armazenando as funções que podem ser chamadas por quem instalar a biblioteca
```

CÓDIGO FONTE DO ARQUIVO *programinhaCriptografia.py*, QUE PROPICIA UMA INTERFACE PARA O USUÁRIO REALIZAR A CRIPTOGRAFIA E DESCRIPTOGRAFIA:

```
import crypto_pt as cpt # importando a biblioteca que criei
```

```
while True:
  mensagem = input("Digite uma mensagem secreta para ser criptografada: ") # recebe a mensagem a ser criptografada
  # garantindo que a mensagem não tenha mais que 256 caracteres
  if len(mensagem) > 256:
    print("ERRO: Mensagem muito longa! Use até 256 caracteres. Para prosseguir.")
    continue

  print("Entrada válida, prosseguindo com a criptografia...")
  break

publica, privada = cpt.gerar_chaves_grandes(512) # chamando a função que inicia a criptografia gerando os números, nesse caso passando 512 bits como parâmetro

cript = cpt.criptografar(mensagem, publica)
print(f"\nSua chave pública é: {publica}")
print(f"\nSua mensagem criptografada é: {cript}")

cript = int(input("\nForneça sua mensagem criptografada para ser descriptografada: "))
decript = cpt.descriptografar(cript, privada)
print(f"\nSua chave privada é: {privada}")
print(f"\nSua mensagem original é: {decript}")
```

REFERÊNCIAS

Anthropic. **Sugestões de artigos acadêmicos sobre RSA.** Claude Sonnet 3.7 versão de 24 fev. 2025. Inteligência Artificial. Disponível em: https://claude.ai. Acesso em: maio 2025.

ASSIS, T. S. **Criptografia RSA: Teoria e Prática com Python**. 2024. Instituto de Matemática, Estatística e Física, Universidade Federal do Rio Grande, São Lourenço do Sul. Disponível em: https://imef.furg.br/images/documentos/matematica-aplicada/monografias/2024_Tatiellen_Souza_Assis.pdf. Acesso em: maio 2025.

CASTRO, F. L. **Criptografia RSA: uma abordagem para professores do ensino básico**. 2014. Instituto de Matemática, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: https://lume.ufrgs.br/handle/10183/110014. Acesso em: maio 2025.

CONRAD, K. **Encyclopedia of Cryptography and Network Security: Miller-Rabin Test**. 1 ed. Boston: Springer, 2011.

DIFFIE, W.; HELLMAN, M. E. New directions in cryptography. **IEEE Transactions on Information Theory**, nov. 1976.

LIMA, R. C. **Criptografia RSA e a Teoria dos Números**. 2013. Centro de Ciências Exatas e da Natureza, Universidade Federal da Paraíba, João Pessoa. Disponível em: https://repositorio.ufpb.br/jspui/handle/tede/7507. Acesso em: maio 2025.

MAZIERO, C. A. **Fundamentos de criptografia**. [s.d.]. Departamento de Informática, Universidade Federal do Paraná. Disponível em: https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-27.pdf. Acesso em: maio 2025.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, fev. 1978.

SOUSA, A. N. L. **Criptografia de Chave Pública, Criptografia RSA**. 2013. Faculdade de Engenharia, Universidade Estadual Paulista "Júlio de Mesquita Filho", Ilha Solteira. Disponível em: https://repositorio.unesp.br/handle/11449/111044. Acesso em: maio 2025.

STALLINGS, W. **Cryptography and Network Security: Principles and Practice**. 7. ed. Boston: Pearson, 2017.