A Regra de Simpson para o cálculo da entropia de geradores pseudo-aleatórios no sistema criptográfico RSA

Felipe Gurgel Araujo
Pedro Francisco Staino Santayana

19 de Julho, 2024

Abstract

A criptografia RSA é um dos métodos mais populares para a segurança de dados, utilizando chaves públicas e privadas para criptografar e descriptografar informações. A geração de chaves RSA envolve a criação de dois grandes números primos, que são usados para formar a chave pública e a chave privada. A entropia das chaves geradas é crucial para garantir a segurança, pois ela mede a imprevisibilidade das chaves. Neste artigo, exploramos a geração de chaves RSA usando a biblioteca `cryptography` do Python, calculamos a entropia das chaves geradas utilizando a Regra de Simpson 3/8.

1 Introdução

A criptografia RSA é amplamente utilizada para a segurança de dados, empregando chaves públicas e privadas. A segurança dessas chaves é avaliada pela entropia, uma medida da imprevisibilidade das chaves geradas. Neste artigo, detalhamos o processo de geração de chaves RSA usando a biblioteca `cryptography` do Python e realizamos a análise de entropia das chaves geradas.

2 Metodologia

2.1 Geração de Chaves RSA

Utilizamos a biblioteca `cryptography` para gerar um conjunto de chaves RSA de 2048 bits. A função de geração de chaves foi implementada da seguinte forma:

```
import math
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
from collections import Counter
```

```
def generate_rsa_key():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )
    public_key = private_key.public_key()
    return public_key

# Gerar um conjunto de chaves
num_keys = 1000
keys = [generate_rsa_key() for _ in range(num_keys)]
```

2.2 Serialização e Contagem de Frequências

As chaves públicas geradas foram serializadas e suas frequências foram contadas para analisar a distribuição das chaves:

```
# Serializar as chaves públicas para comparar e contar frequências
serialized_keys = [key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
) for key in keys]

# Calcular a frequência de cada chave pública
key_frequencies = Counter(serialized_keys)
```

2.3 Cálculo da Entropia

A entropia foi calculada diretamente a partir das frequências das chaves geradas, usando a fórmula de Shannon:

$$H(X) = -\sum p(x) \log_2(p(x))$$

Onde $p(x)$ é a probabilidade de ocorrência de cada chave.

2.4 Regra de Simpson 3/8

Para calcular a entropia, utilizamos a segunda regra de Simpson (Simpson 3/8), que é uma fórmula de integração numérica que usa polinômios cúbicos para aproximar a integral de uma função.

```
def simpsons_3_8_rule(f, a, b, n):
    if n % 3 != 0:
        n += 3 - (n % 3)  # n deve ser múltiplo de 3 para a regra de Simpson 3/8
```

```
    h = (b - a) / n
    integral = f(a) + f(b)

    for i in range(1, n):
        xi = a + i * h
        if i % 3 == 0:
            integral += 2 * f(xi)
        else:
            integral += 3 * f(xi)

    return (3 * h / 8) * integral

# Função de entropia
def entropy(p):
    return -p * math.log2(p) if p > 0 else 0

# Função de entropia ajustada para índices
def entropy_index(i, unique_keys, total_keys):
    key_index = int(i)
    if key_index >= len(unique_keys):
        return 0
    key = unique_keys[key_index]
    p = key_frequencies[key] / total_keys
    return entropy(p)

# Como temos uma distribuição discreta de chaves, vamos usar os índices dos elementos como x
a = 0  # índice inicial
b = len(key_frequencies) - 1  # índice final
n = 1000  # número de subdivisões

# Para facilitar a integração, mapeamos os índices aos itens únicos na lista de chaves
unique_keys = list(key_frequencies.keys())

# Calculando a entropia diretamente a partir das frequências usando a regra de Simpson 3/8
entropy_value = simpsons_3_8_rule(lambda i: entropy_index(i, unique_keys, num_keys), a, b, n
```

2.5 Valor Recomendado de Entropia

O valor recomendado de entropia para n chaves únicas é $\log_2(n)$. Isso representa a quantidade máxima de informação que pode ser obtida de um conjunto de n itens únicos, assumindo que todos os itens são igualmente prováveis. Para 1000 chaves, o valor recomendado é:

```
recommended_entropy = math.log2(num_keys)
```

3 Resultados

Após a execução do script, a entropia calculada das chaves RSA geradas foi revelada:

```
print(f"A entropia calculada das chaves RSA é: {entropy_value} bits")
```

Os resultados mostraram que a entropia calculada estava próxima do valor recomendado, indicando uma boa distribuição e imprevisibilidade das chaves geradas.

4 Conclusão

A geração de chaves RSA usando a biblioteca `cryptography` do Python provou ser eficiente e segura. A análise da entropia das chaves geradas indicou que as chaves são suficientemente aleatórias e seguras. Comparar a entropia calculada com o valor recomendado é uma prática importante para garantir a robustez das chaves criptográficas.

5 Referências

- Shannon, C. E. (1948). "A Mathematical Theory of Communication". Bell System Technical Journal, 27, 379-423, 623-656.
- Cover, T. M., & Thomas, J. A. (2006). "Elements of Information Theory". John Wiley & Sons.
- Katz, J., & Lindell, Y. (2020). "Introduction to Modern Cryptography". CRC Press.
- RSA (Rivest–Shamir–Adleman)
- Cryptography Library Documentation
- Python Documentation