Universidade Federal de Santa Catarina
Departamento de Informática e Estatística (INE)
Ciências da Computação
Segurança da Computação (INE5429)
Professores: Jean Everson Martina e Thaís Bardini Idalino

Gian Ferrari (22100619)
Guilherme Adenilson de Jesus (22100620)
Gustavo Konescki Fuhr (22203675)

Relatório - Curvas Elípticas

Florianópolis
2025

Sumário

1 Introdução

A segurança digital se tornou um aspecto fundamental no mundo atual, no qual garante a privacidade de conversas e proteção de dados sensíveis. Neste contexto, a criptografia de chave pública possui uma função fundamental, possibilitando a troca de mensagens de forma segura. Os algoritmos mais populares são RSA e criptografia com curvas elípticas.

Assim, o presente trabalho busca fazer um estudo da eficiência da criptografia de chave pública baseada em curvas elípticas. Este estudo analisará tanto a eficiência de diferentes curvas elípticas usadas, quanto o uso de curvas elípticas versus RSA.

Sua principal característica é baseada na criptografia de mensagens por meio da **operação de grupo de curvas elípticas** de corpos finitos. Essa terminologia será esclarecida no desenvolvimento. Em contraste, criptografia de chave pública aparece se baseando em operações feitas no grupo multiplicativo $\mathbb{Z}_p$ onde p é primo (HANKERSON; MENEZES; VANSTONE, 2004). A dificuldade envolvida com certas equações, em ambos os casos, é o que confere a força criptográfica dos protocolos.

A motivação da escolha de curvas elípticas se justifica por seu uso em criptografia de chave pública que aparece em diversos protocolos modernos, como o protocolo Diffie-Hellman com curvas elípticas (ECDH). Além disso, entender e observar algumas de suas vantagens e desvantagens cumpre um papel claro na formação de profissionais da área de segurança e ciência da computação.

2 Desenvolvimento

2.1 Criptografia de Chave Pública

Protocolos criptográficos que utilizam de chave pública se baseiam na dificuldade de se encontrar a solução para certas equações. O exemplo mais claro é a troca de chaves Diffie-Hellman, que se baseia na dificuldade de encontrar o **logaritmo discreto**, cuja computação resolveria, para K, uma equação da forma

$$K = a^{X_a \cdot X_b} \mod p \tag{2.1}$$

onde p e a são conhecidos e a^{X_a} e a^{X_b} não estão necessariamente criptografados (MARTINA; IDALINO, 2025).

2.1.1 Curvas Elípticas

O desenvolvimento a seguir se baseia na apresentação em (KOBLITZ, 1987). Definimos uma curva elíptica $E_k(a, b)$ sobre um corpo [1] K como o conjunto

$$E_K(a, b) = \{(x, y) \in K \times K : y^2 = x^3 + ax + b\} \cup \{\infty\}$$

em outras palavras, $E_K(a, b)$ são os pontos que fornecem soluções em K para a equação disposta acima (e mais um ponto "no infinito"). Consideramos que não há raízes múltiplas para x^3. Nos reais, por exemplo, poderíamos ter

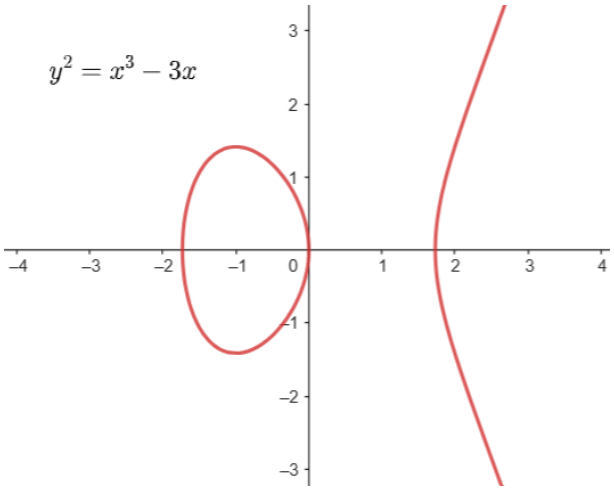

Figura 1 – $E_K(-3, 0)$ em $\mathbb{R}$

[1] Um corpo é um conjunto munido de duas operações, um exemplo seria o conjunto dos números racionais com soma e multiplicação usuais.

onde $E_K(-3, 0)$ está em vermelho. Sobre estas curvas elípticas, podemos definir um grupo. A operação do grupo será descrita geometricamente: Para pontos $P, Q \in K \times K$, o produto PQ será o ponto conforme segue:

1. Trace a reta $\bar{PQ}$,
2. Encontre um terceiro ponto $R = (r_x, r_y)$ onde $\bar{PQ}$ intersecta com a curva,
3. Calcular $(r_x, -r_y) = PQ$.

Se $P = Q$, então a reta é a tangente naquele ponto. Iremos definir ∞ como o elemento neutro tal que $P\infty = P$, e observaremos que o inverso de um ponto $P = (p_x, p_y)$ é $-P = (p_x, -p_y)$. Ilustramos essa operação na figura abaixo:

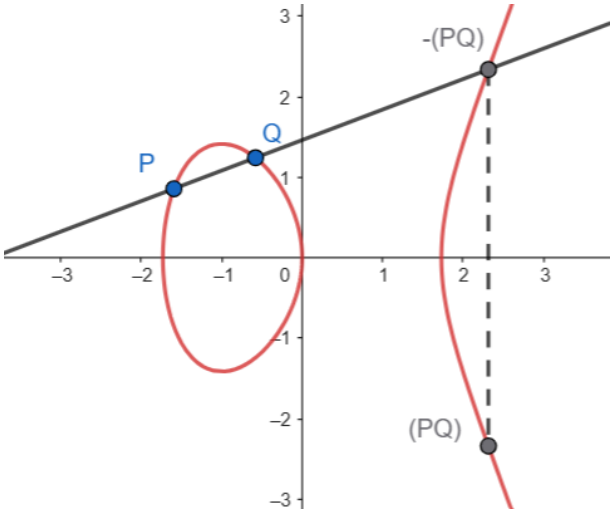

Figura 2 – Exemplo de produto PQ

Existem fórmulas algébricas que nos permitem calcular estas novas coordenadas. Com estas fórmulas, podemos computar uma operação $mP = PP...P$ no mesmo tempo computacional de uma exponenciação a^m, por meio de procedimentos análogos (KOBLITZ, 1987).

Formas de transformar mensagens em pontos de $E_K(a, b)$ dependem do tipo de corpo K envolvido. Iremos assumir que há um método eficiente de codificar e decodificar mensagens em $E_K(a, b)$. A segurança da criptografia com curvas elípticas vem do fato de que é tão difícil resolver a equação 2.1 quanto a seguinte equação: Dados pontos $P, Q \in E_K(a, b)$, encontre x inteiro tal que

$$Q = xP \tag{2.2}$$

se tal x de fato existir. Esse problema é análogo ao logaritmo discreto, e por vezes essa analogia dá a este problema o nome de **logaritmo discreto para curvas elípticas.**

2.1.2 Uso de Curvas Elípticas

Esta seção se baseia na apresentação de (HANKERSON; MENEZES; VANSTONE, 2004).

Um dos principais usos de curvas elípticas na criptografia prática é na construção de esquemas de troca de chaves seguras. O **Elliptic Curve Diffie-Hellman** (ECDH) é uma versão do protocolo clássico de Diffie-Hellman, adaptada ao contexto de curvas elípticas, e permite que duas partes estabeleçam uma chave secreta compartilhada sobre um canal inseguro.

O protocolo assume que as duas partes, Alice e Bob, concordam previamente sobre uma curva elíptica $E_K(a, b)$ sobre um corpo finito K e um ponto gerador público $P \in E_K(a, b)$. Cada parte então escolhe uma chave privada aleatória: Alice escolhe $n \in \mathbb{Z}$ e Bob escolhe $u \in \mathbb{Z}$. A partir disso, calculam suas respectivas chaves públicas como

$$A = nP \quad \text{e} \quad B = uP.$$

Essas chaves públicas são trocadas entre as partes. Alice, ao receber B, calcula a chave compartilhada como $K = nB = nuP$, enquanto Bob, ao receber A, calcula $K = uA = unP$. Ambos obtêm a mesma chave, sem nunca transmitirem suas chaves privadas.

Esse protocolo é amplamente utilizado em sistemas modernos como TLS, Signal e criptomoedas, devido à sua eficiência em oferecer segurança equivalente com tamanhos de chave menores em comparação a métodos clássicos como RSA e DH sobre $\mathbb{Z}_p$.

2.2 Padrões de Curvas Elípticas

Diversos órgãos e comunidades propuseram curvas elípticas padronizadas para uso em criptografia, com diferentes objetivos de segurança, desempenho e interoperabilidade. Os principais padrões adotados internacionalmente são apresentados a seguir.

Diferenças entre os padrões incluem:

- Qual o corpo subjacente ($\mathbb{F}_p$ vs. $\mathbb{F}_{2^m}$),
- Os parâmetros a, b da curva $E_K(a, b)$, e
- O ponto gerador $G \in E_K(a, b)$ e sua ordem [2] n .

2.2.1 NIST (National Institute of Standards and Technology)

O *National Institute of Standards and Technology* (NIST), órgão dos Estados Unidos, publicou curvas como **secp256r1 (P-256)**, **secp384r1 (P-384)** e **secp521r1 (P-521)**

[2] A ordem de um ponto é o menor número n tal que $nG = \infty$.

no padrão FIPS 186-5 (NIST, 2013). Essas curvas são amplamente utilizadas em protocolos como TLS, certificados digitais e autenticação. No entanto, após as revelações de Edward Snowden em 2013, surgiram preocupações sobre possíveis *backdoors*, o que motivou a busca por alternativas mais transparentes.

2.2.2 SECG (Standards for Efficient Cryptography Group)

O *Standards for Efficient Cryptography Group* (SECG) propôs curvas como **secp256k1**, amplamente utilizada em criptomoedas como Bitcoin e Ethereum. Essas curvas seguem a especificação SEC 2 (SECG, 2010) e são otimizadas para eficiência em ambientes com recursos computacionais limitados.

2.2.3 Brainpool

O projeto Brainpool, promovido por instituições europeias, definiu curvas como **brainpoolP256r1** e **brainpoolP512r1** no RFC 5639 (LOCHTER; MERKLE, 2010). Essas curvas utilizam primos aleatórios em vez de primos especiais, com o objetivo de evitar possíveis vulnerabilidades estruturais. Embora apresentem desempenho inferior às curvas NIST, são consideradas mais transparentes em sua construção.

2.2.4 Curvas aceitas no Brasil

No Brasil, a ICP-Brasil (Infraestrutura de Chaves Públicas Brasileira) regulamenta os algoritmos permitidos para certificados digitais por meio do documento DOC-ICP-01.01 (ITI, 2022). Historicamente, o Brasil adotou curvas NIST (como **P-256** e **P-521**), mas desde 2014 passou a priorizar curvas Brainpool (COELHO, 2022), especialmente:

- **brainpoolP256r1** para certificados A1, A3, S3, T3;
- **brainpoolP512r1** para certificados A4, S4 e T4.

Além disso, outras curvas como **Curve25519**, **Ed25519**, **Ed448** e **E-521** também foram incorporadas às versões mais recentes da regulamentação, ampliando a diversidade de algoritmos aceitos em território nacional (COELHO, 2022).

3 Experimento

Neste capítulo, serão apresentados experimentos comparativos no desempenho de operações utilizando criptografia de curvas elípticas. Os experimentos serão realizados utilizando a linguagem `Python` em ambiente `Jupyter Notebook` no Google Colab. As bibliotecas principais usadas incluem `cryptography` para algoritmos criptográficos, `pandas` e `numpy` para processamento, e `matplotlib` para visualização.

As análises realizadas serão:

- Medição e comparação do tempo de execução na geração de chave, assinatura e verificação entre RSA (de 1024, 2048 e 4096 bits) e as curvas NIST, SECG e Brainpool disponíveis na biblioteca `cryptography` (CRYPTOGRAPHY, 2025).
- Avaliação do tempo para gerar um segredo compartilhado via protocolo Diffie-Hellman utilizando diferentes curvas elípticas.

As curvas utilizadas para os experimentos estão descritas na Tabela 1. Foram escolhidas com base na recomendação por órgãos padronizadores como NIST, SEGC e Brainpool. Ademais, a fim de garantir uma medição mais confiável e minimizar variações temporais decorrentes do *hardware*, cada operação será executada 100 vezes. O código dos experimentos pode ser visto a partir do link: <https://colab.research.google.com/drive/1NEr_HIRZLBO2w6sLryl73tSbZHJH0SPE?usp=sharing>

Tabela 1 – Curvas utilizadas

Nome da Curva	Tamanho da Chave	Origem/Padrão
SECP256K1	256 bits	SECG (SEC 2)
SECP256R1	256 bits	NIST P-256
SECP384R1	384 bits	NIST P-384
SECP521R1	521 bits	NIST P-521
BrainpoolP256r1	256 bits	Brainpool (RFC 5639)
SECT163K1	163 bits	NIST K-163
SECT163R2	163 bits	NIST B-163
SECP192R1	192 bits	NIST P-192
SECP224R1	224 bits	NIST P-224
SECT233K1	233 bits	NIST K-233
SECT233R1	233 bits	NIST B-233
SECT283K1	283 bits	NIST K-283
SECT283R1	283 bits	NIST B-283
BrainpoolP384R1	384 bits	Brainpool (RFC 5639)
SECT409K1	409 bits	NIST K-409
SECT409R1	409 bits	NIST B-409
BrainpoolP512R1	512 bits	Brainpool (RFC 5639)
SECT571K1	571 bits	NIST K-571
SECT571R1	571 bits	NIST B-571

Fonte: (CRYPTOGRAPHY, 2025)

3.1 Função de medição

As medições foram realizadas a partir da função abaixo, com auxílio da biblioteca `time` para temporização. A variável `op` é o método que será medido o tempo de execução, com `args_op` sendo seus parâmetros, por `n` iterações.

```
    def realizar_medicao(op, args_op, n):
        tempos = []
        for _ in range(n):
          start = time.process_time_ns()
          op(*args_op)
          end = time.process_time_ns()
          tempos.append(end - start)

        return np.array(tempos)
```

3.2 Geração de chaves

Abaixo está o código que realiza as medições para geração de chaves de cada curva citada na Tabela 1 e RSA de 1024, 2048 e 4096 bits. Ao fim, os tempos (em ms) são

agrupados em um `DataFrame`.

```
n = 100
tempos = dict()

for curva in curvas:
  tempo_curva = realizar_medicao(ec.generate_private_key, [curva], n) / 1e6
  tempos[curva.name] = tempo_curva

for rsa_config in rsa_configs:
  tempo_curva = realizar_medicao(rsa.generate_private_key, rsa_config, n) / 1e6
  tempos[f'RSA_{rsa_config[-1]}'] = tempo_curva

resultados = pd.DataFrame(tempos)
resultados.describe()
```

A Tabela 2 mostra as medidas descritivas dos resultados na geração de chaves de cada curva e RSA ordenadas pelo tamanho da chave, enquanto a Figura 3 exibe o tempo médio em milissegundos na geração de chaves entre as curvas e o RSA.

Nota-se que os algoritmos de RSA obtiveram os maiores tempos de execução. A curva secp256r1 (NIST P-256) apresentou o menor tempo médio de execução (0,03 ms), o que pode ser justificado pelo número primo utilizado, $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, ser um pseudo-Mersenne, que permite reduções modulares rápidas, acelerando a multiplicação escalar (HANKERSON; MENEZES; VANSTONE, 2004).

Já o RSA, em especial o RSA_4096, que registrou um tempo médio 20.617 vezes maior que a secp256r1, atingindo 618,51 ms. O motivo pode estar relacionado à necessidade da geração de dois números primos aleatórios a partir de testes de primalidade, como Miller-Rabin, que possui um custo computacional (MARTINA; IDALINO, 2025).

Tabela 2 – Estatísticas descritivas na geração de chaves (ms).

Curva/RSA	**Média**	**Desvio Padrão**	**Mínimo**	**25%**	**Mediana**	**75%**	**Máximo**
sect163k1	0,23	0,09	0,18	0,18	0,19	0,22	0,74
sect163r2	0,23	0,07	0,19	0,19	0,20	0,22	0,64
secp192r1	0,24	0,02	0,23	0,23	0,23	0,24	0,35
secp224r1	0,03	0,01	0,03	0,03	0,03	0,03	0,14
sect233k1	0,26	0,03	0,24	0,25	0,25	0,26	0,40
sect233r1	0,26	0,01	0,25	0,25	0,25	0,26	0,30
secp256r1	0,03	0,01	0,02	0,02	0,02	0,02	0,15
secp256k1	0,41	0,02	0,40	0,40	0,40	0,41	0,60
brainpoolP256r1	0,39	0,06	0,36	0,36	0,37	0,38	0,81
sect283k1	0,44	0,06	0,42	0,42	0,43	0,43	0,94
sect283r1	0,63	0,20	0,44	0,47	0,63	0,74	1,99
secp384r1	0,18	0,04	0,15	0,15	0,16	0,20	0,26
brainpoolP384r1	0,92	0,12	0,85	0,86	0,88	0,91	1,60
sect409k1	0,77	0,11	0,70	0,72	0,73	0,77	1,21
sect409r1	0,83	0,11	0,75	0,76	0,78	0,87	1,28
secp521r1	0,16	0,02	0,15	0,15	0,15	0,16	0,31
brainpoolP512r1	1,35	0,20	1,23	1,25	1,27	1,31	2,24
sect571k1	2,31	0,52	1,55	1,66	2,58	2,70	3,12
sect571r1	3,07	0,35	1,75	2,92	3,09	3,23	4,06
RSA_1024	11,93	4,37	6,57	9,33	11,19	13,66	31,23
RSA_2048	67,95	34,44	18,24	41,08	67,17	85,90	176,42
RSA_4096	618,51	418,77	45,67	317,40	520,76	828,76	2329,86

Fonte: Autores.

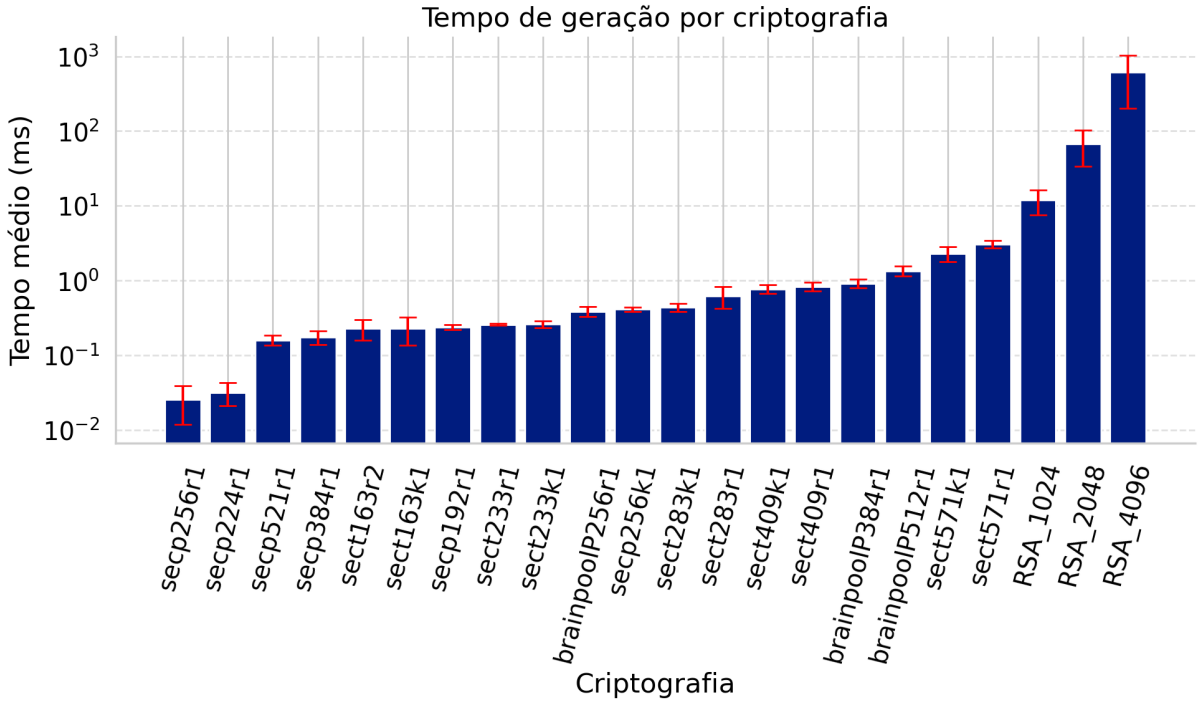

Figura 3

Fonte: Autores.

3.3 Assinatura

O código a seguir executa as medições para assinatura de uma mensagem de cada curva citada na Tabela 1 e RSA de 1024, 2048 e 4096 bits. Além disso, as funções auxiliares `aux_curve_sign` e `aux_rsa_sign` geram as chaves privadas para poder realizar a assinatura.

```
n = 100
tempos = dict()
data = b"Uma mensagem a ser assinada"
for curva in curvas:
  private_key = aux_curve_sign(curva)
  tempo_curva = realizar_medicao(private_key.sign, [data, ec.
   ECDSA(hashes.SHA256())], n) / 1e6
  tempos[curva.name] = tempo_curva

for rsa_config in rsa_configs:
  private_key = aux_rsa_sign(rsa_config)
  tempo_curva = realizar_medicao(private_key.sign, [data, padding
   .PSS(mgf=padding.MGF1(hashes.SHA256()), salt_length=padding.PSS
   .MAX_LENGTH), hashes.SHA256()], n) / 1e6
  tempos[f'RSA_{rsa_config[-1]}'] = tempo_curva

resultados = pd.DataFrame(tempos)
resultados.describe()
```

A Tabela 3 exibe as medidas descritivas dos resultados na assinatura de cada curva e RSA ordenadas pelo tamanho da chave, já a Figura 4 apresenta o tempo médio em milissegundos na assinatura entre as curvas e o RSA. A curva secp256r1 registrou o menor tempo para assinatura pelo mesmo motivo apresentado na etapa de geração de chaves. Observa-se que o algoritmo de RSA_1024 se destacou em relação a várias curvas, obtendo o terceiro menor tempo de execução. Isto ocorre devido ao tamanho da chave usada para exponenciação modular ser menor (1024), enquanto curvas elípticas realizam operações mais complexas (HANKERSON; MENEZES; VANSTONE, 2004).

Tabela 3 – Estatísticas descritivas da assinatura (ms).

Curva/RSA	Média	Desvio Padrão	Mínimo	25%	Mediana	75%	Máximo
sect163k1	0,37	0,05	0,30	0,35	0,37	0,38	0,81
sect163r2	0,37	0,05	0,29	0,34	0,37	0,39	0,60
secp192r1	0,43	0,03	0,37	0,41	0,42	0,44	0,55
secp224r1	0,10	0,02	0,07	0,09	0,09	0,10	0,25
sect233k1	0,50	0,16	0,39	0,46	0,49	0,51	1,98
sect233r1	0,53	0,07	0,45	0,50	0,51	0,53	0,99
secp256r1	0,04	0,02	0,04	0,04	0,04	0,04	0,21
secp256k1	0,69	0,05	0,61	0,65	0,69	0,73	0,83
brainpoolP256r1	0,60	0,06	0,55	0,57	0,58	0,61	0,96
sect283k1	0,82	0,09	0,70	0,77	0,80	0,85	1,28
sect283r1	0,84	0,08	0,70	0,79	0,84	0,87	1,29
secp384r1	0,35	0,02	0,28	0,34	0,35	0,36	0,44
brainpoolP384r1	1,45	0,19	0,97	1,39	1,43	1,48	2,83
sect409k1	1,42	0,08	1,13	1,39	1,42	1,44	1,93
sect409r1	1,48	0,15	1,23	1,40	1,48	1,51	2,33
secp521r1	0,44	0,05	0,37	0,42	0,43	0,44	0,70
brainpoolP512r1	2,13	0,16	1,91	2,07	2,11	2,15	3,05
sect571k1	2,04	0,49	1,66	1,69	1,75	2,57	3,15
sect571r1	1,84	0,12	1,76	1,77	1,78	1,82	2,39
RSA_1024	0,11	0,04	0,10	0,10	0,10	0,11	0,46
RSA_2048	0,69	0,08	0,65	0,66	0,67	0,68	1,17
RSA_4096	4,76	0,26	4,54	4,61	4,66	4,78	5,83

Fonte: Autores.

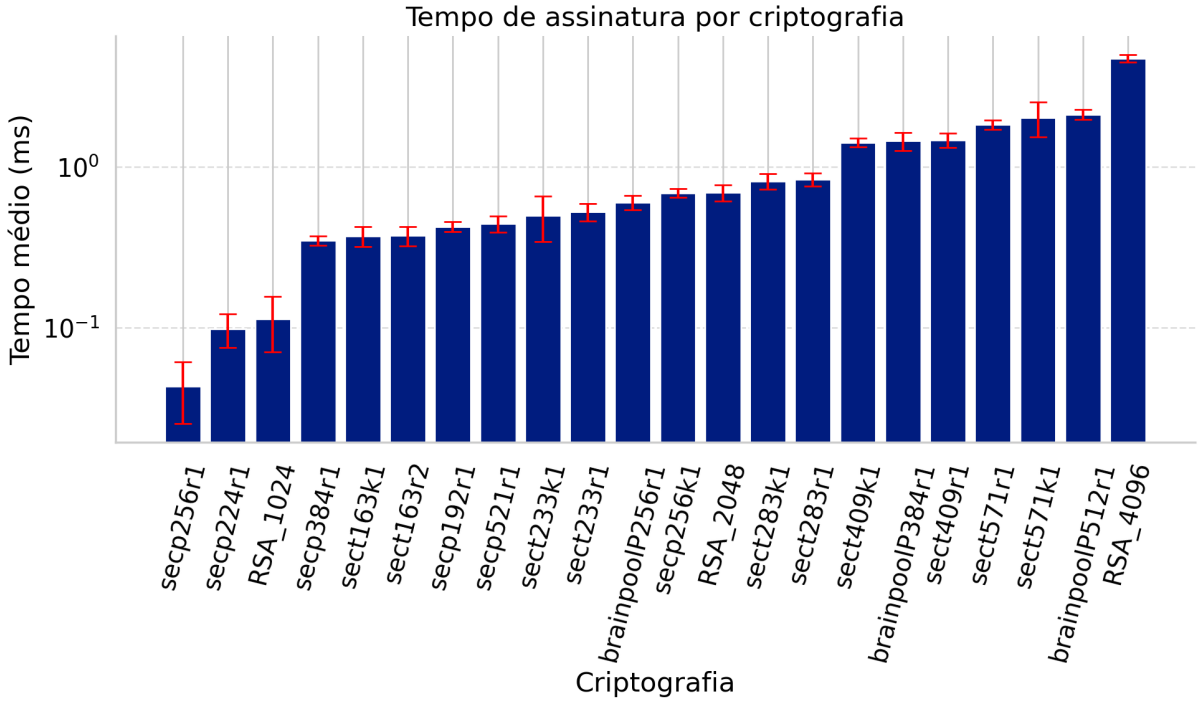

Figura 4

Fonte: Autores.

3.4 Verificação de assinatura

O código abaixo executa as medições para verificação de assinatura de uma mensagem de cada curva citada na Tabela 1 e RSA de 1024, 2048 e 4096 bits. Além disso, as funções auxiliares `aux_curve_verify` e `aux_rsa_verify` assinam uma mensagem com uma chave privada, retornando a chave pública e a assinatura.

```
n = 100
tempos = dict()
data = b"Uma mensagem a ser assinada"
for curva in curvas:
  public_key, sig = aux_curve_verify(curva, data)
  tempo_curva = realizar_medicao(public_key.verify, [sig, data,
   ec.ECDSA(hashes.SHA256())], n) / 1e6

  tempos[curva.name] = tempo_curva

for rsa_config in rsa_configs:

  public_key, sig = aux_rsa_verify(rsa_config, data)
  tempo_curva = realizar_medicao(public_key.verify, [sig, data,
   padding.PSS(mgf=padding.MGF1(hashes.SHA256()), salt_length=
   padding.PSS.MAX_LENGTH),hashes.SHA256()], n) / 1e6

  tempos[f'RSA_{rsa_config[-1]}'] = tempo_curva

resultados = pd.DataFrame(tempos)
resultados.describe()
```

A Tabela 3 mostra as medidas descritivas dos resultados na verificação de assinatura de cada curva e RSA ordenadas pelo tamanho da chave. Por sua vez, Figura 5 ilustra o tempo médio em milissegundos na verificação de assinatura entre as curvas e o RSA. Destaca-se a eficiência do RSA nessa operação, que pode ser justificado pela forma que sua verificação é feita, $m = s^e \mod n$, sendo s a assinatura e e a chave pública. Além disso, o expoente público utilizado foi de 65537, um número pequeno em comparação à magnitude das curvas elípticas, permitindo uma exponenciação modular mais rápida. Por fim, assim como na assinatura, as curvas elípticas demandam operações mais complexas, como inversão modular, que pode ter influenciado nas medições (HANKERSON; MENEZES; VANSTONE, 2004).

Tabela 4 – Estatísticas descritivas da verificação de assinatura (ms).

Curva/RSA	Média	Desvio Padrão	Mínimo	25%	Mediana	75%	Máximo
sect163k1	0,47	0,15	0,36	0,37	0,38	0,61	1,02
sect163r2	0,44	0,10	0,38	0,39	0,40	0,42	0,94
secp192r1	0,24	0,04	0,21	0,22	0,22	0,23	0,41
secp224r1	0,12	0,07	0,09	0,09	0,10	0,13	0,71
sect233k1	0,67	0,19	0,50	0,52	0,55	0,90	1,09
sect233r1	0,56	0,06	0,51	0,52	0,53	0,55	0,93
secp256r1	0,08	0,01	0,07	0,07	0,07	0,08	0,12
secp256k1	0,38	0,05	0,36	0,37	0,37	0,38	0,68
brainpoolP256r1	0,40	0,04	0,36	0,37	0,38	0,43	0,57
sect283k1	0,92	0,10	0,86	0,87	0,88	0,92	1,41
sect283r1	0,93	0,05	0,90	0,91	0,92	0,93	1,33
secp384r1	0,48	0,03	0,46	0,47	0,47	0,48	0,72
brainpoolP384r1	0,78	0,03	0,76	0,77	0,78	0,78	0,93
sect409k1	1,52	0,10	1,47	1,48	1,49	1,52	2,30
sect409r1	1,58	0,04	1,55	1,56	1,56	1,58	1,80
secp521r1	0,51	0,02	0,50	0,51	0,51	0,51	0,65
brainpoolP512r1	1,51	0,36	1,06	1,12	1,66	1,79	2,46
sect571k1	5,86	0,48	3,68	5,77	5,91	6,00	6,82
sect571r1	5,86	0,65	3,45	5,74	5,95	6,29	7,48
RSA_1024	0,02	0,01	0,01	0,01	0,01	0,01	0,13
RSA_2048	0,03	0,01	0,03	0,03	0,03	0,03	0,07
RSA_4096	0,08	0,02	0,08	0,08	0,08	0,08	0,22

Fonte: Autores.

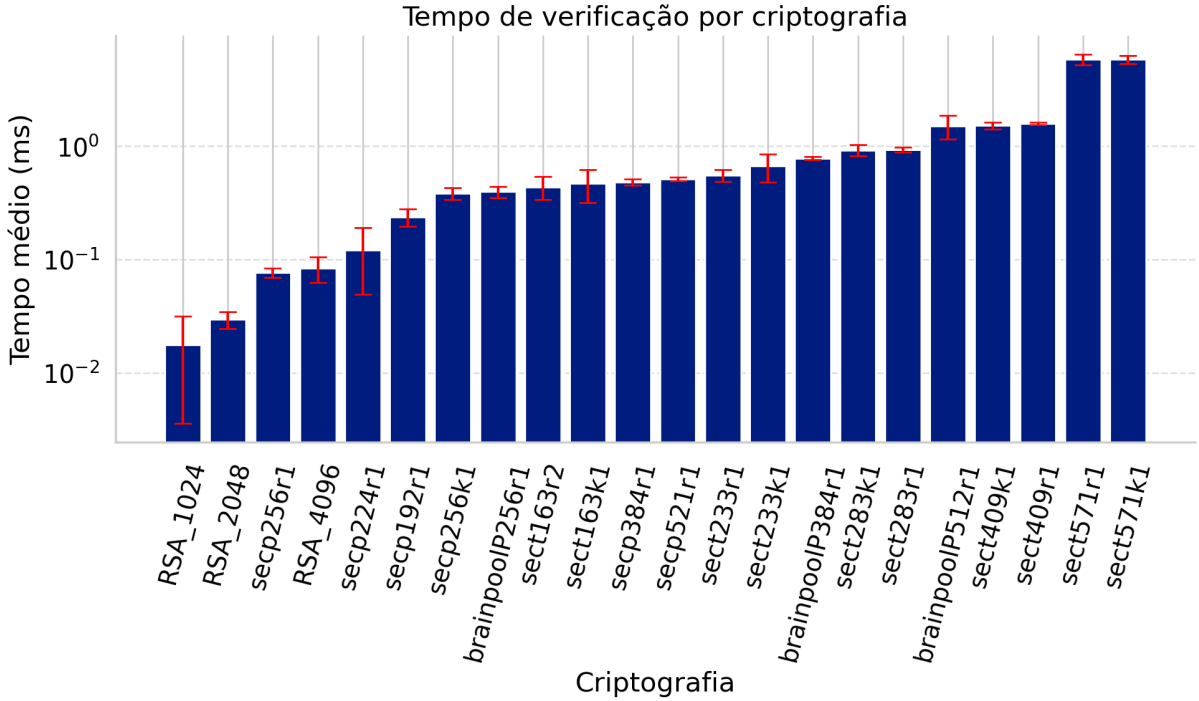

Figura 5

Fonte: Autores.

3.5 Protocolo Diffie-Hellman

O código a seguir que executa as medições para troca de chaves pelo protocolo Diffie-Hellman de cada curva citada na Tabela 1. A função auxiliar `aux_curve_DH` realiza a geração da chave privada e pública necessárias para troca.

```
n = 100
tempos = dict()
for curva in curvas:
  server_private_key, peer_public_key = aux_curve_DH(curva)
  tempo_curva = realizar_medicao(server_private_key.exchange, [ec
   .ECDH(), peer_public_key], n) / 1e6

  tempos[curva.name] = tempo_curva

resultados = pd.DataFrame(tempos)
resultados.describe()
```

A Tabela 3 mostra as medidas descritivas dos resultados no protocolo Diffie-Hellman para cada curva ordenadas pelo tamanho da chave. Por sua vez, Figura 5 ilustra o tempo médio em milissegundos entre as curvas. De modo geral, os resultados entre as curvas foram semelhantes às operações anteriores, onde as curvas secp224r1 e secp256r1 se sobressaíram entre as demais em razão de utilizarem primos pseudo-Mersenne (NIST, 2013), o que permite operações modulares mais rápidas. Ademais, constata-se que o tamanho da chave impacta significativamente no tempo de execução, no qual curvas com maiores chaves tendem a possuir um maior tempo (HANKERSON; MENEZES; VANSTONE, 2004).

Tabela 5 – Estatísticas descritivas da execução do protocolo Diffie-Hellman (ms).

Curva	Média	Desvio Padrão	Mínimo	25%	Mediana	75%	Máximo
sect163k1	0,27	0,08	0,17	0,18	0,31	0,33	0,49
sect163r2	0,20	0,04	0,18	0,18	0,18	0,19	0,34
secp192r1	0,22	0,01	0,21	0,22	0,22	0,22	0,30
secp224r1	0,06	0,01	0,06	0,06	0,06	0,06	0,13
sect233k1	0,26	0,05	0,24	0,24	0,24	0,26	0,44
sect233r1	0,27	0,05	0,24	0,25	0,25	0,26	0,63
secp256r1	0,06	0,05	0,05	0,05	0,05	0,05	0,50
secp256k1	0,42	0,06	0,38	0,39	0,40	0,42	0,91
brainpoolP256r1	0,39	0,09	0,35	0,36	0,36	0,37	0,91
sect283k1	0,44	0,07	0,41	0,41	0,42	0,42	0,85
sect283r1	0,52	0,13	0,43	0,44	0,47	0,52	1,02
secp384r1	0,40	0,09	0,33	0,34	0,34	0,41	0,76
brainpoolP384r1	0,87	0,06	0,83	0,84	0,85	0,87	1,25
sect409k1	0,76	0,06	0,69	0,71	0,77	0,78	1,08
sect409r1	0,76	0,04	0,74	0,74	0,75	0,76	1,08
secp521r1	0,31	0,02	0,30	0,30	0,31	0,31	0,45
brainpoolP512r1	1,27	0,10	1,21	1,22	1,23	1,26	1,78
sect571k1	1,57	0,07	1,52	1,54	1,55	1,57	2,06
sect571r1	1,70	0,14	1,64	1,64	1,65	1,69	2,67

Fonte: Autores.

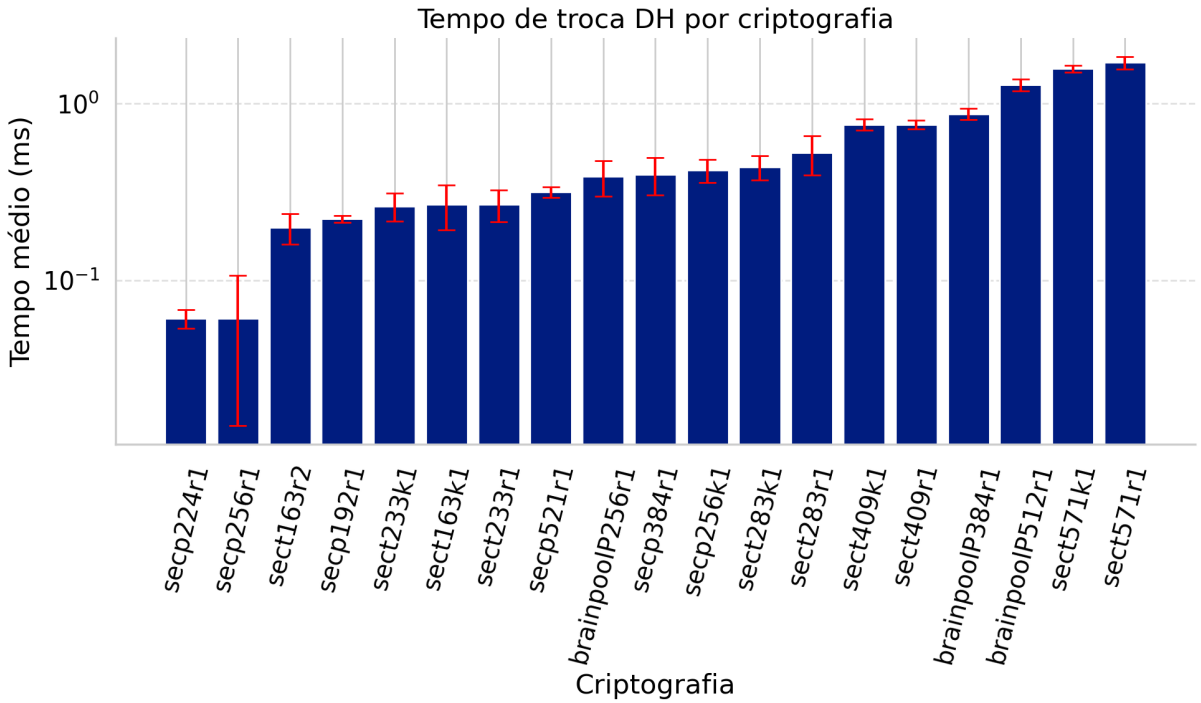

Figura 6

Fonte: Autores.

4 Conclusão

Este trabalho teve como objetivo analisar o desempenho entre curvas elípticas e RSA para diferentes operações criptográficas. Para isto, foram realizados experimentos comparativos entre o tempo de geração de chaves, assinatura, verificação de assinatura e protocolo Diffie-Hellman entre as curvas recomendadas por institutos como NIST, SEGC e Brainpool.

Dessa forma, a partir dos resultados obtidos durante o experimento, conclui-se que embora curvas elípticas sejam mais eficientes para geração de chaves, operações de verificação de assinatura podem ser computacionalmente complexas quando comparadas com o RSA. Além disso, o primo utilizado na curva, influencia significativamente na velocidade das operações, sendo os de tipo pseudo-Mersenne mais otimizados para essas funções.

Referências Bibliográficas

COELHO, M. A. **Workshop Padrões e Tecnologias da ICP-Brasil**. 2022. Acesso em: 01 jul. 2025. Disponível em: <https://workshopicpmercosul.paginas.ufsc.br/files/2022/04/Workshop_MERCOSUL_Padr%C3%B5es_e_Tecnologias_da_ICP_Brasil_Curvas_El%C3%ADpticas.pdf>. Citado na página 6.

CRYPTOGRAPHY. **Elliptic curve cryptography**. 2025. Acesso em: 28 jun. 2025. Disponível em: <https://cryptography.io/en/latest/hazmat/primitives/asymmetric/ec/#elliptic-curves>. Citado nas páginas 7 e 8.

HANKERSON, D.; MENEZES, A.; VANSTONE, S. **Guide to Elliptic Curve Cryptography**. New York: Springer, 2004. Livro de referência sobre curvas elípticas em criptografia. ISBN 978-0387952734. Citado nas páginas 2, 5, 9, 11, 13 e 15.

ITI. **PADRÕES E ALGORITMOS CRIPTOGRÁFICOS DA ICP-BRASIL DOC ICP-01.01**. 2022. Acesso em: 01 jul. 2025. Disponível em: <https://repositorio.iti.gov.br/instrucoes-normativas/IN2022_22_DOC-ICP-01.01.htm>. Citado na página 6.

KOBLITZ, N. Elliptic curve cryptosystems. **Mathematics of Computation**, American Mathematical Society, v. 48, n. 177, p. 203–209, 1987. Citado nas páginas 3 e 4.

LOCHTER, M.; MERKLE, J. **Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation**. 2010. Acesso em: 28 jun. 2025. Disponível em: <https://datatracker.ietf.org/doc/html/rfc5639>. Citado na página 6.

MARTINA, J. E.; IDALINO, T. B. **Criptografia Assimétrica e Integridade**. 2025. Slides de Aula da disciplina INE5429, Segurança em Computação, disponível via Moodle. Acessado em 29 jun. 2025. Citado nas páginas 3 e 9.

NIST. **FIPS 186-5: Digital Signature Standard (DSS)**. 2013. Acesso em: 28 jun. 2025. Disponível em: <https://csrc.nist.gov/pubs/fips/186-5/final>. Citado nas páginas 6 e 15.

SECG. **SEC 2: Recommended Elliptic Curve Domain Parameters**. 2010. Acesso em: 28 jun. 2025. Disponível em: <https://www.secg.org/sec2-v2.pdf>. Citado na página 6.