



UNIVERSIDADE FEDERAL DE PELOTAS

ALGORITMO E ESTRUTURA DE DADOS III

---

# RSA AES

---

*Autores:*

Guilherme Hepp da Fonseca  
ghfonseca@inf.ufpel.edu.br

Nicolas Cipriano  
ncsdoliveira@inf.ufpel.edu.br

20 de fevereiro de 2025

## Sumário

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Arquivos Analisados</b>	<b>2</b>
<b>3</b>	<b>Key_Public e Hash</b>	<b>3</b>
3.0.1	Hashcat e Descriptografia da key_public.en . . . . .	3
<b>4</b>	<b>Fatoração Cado-NFS</b>	<b>6</b>
<b>5</b>	<b>Encontrar a Mensagem Final</b>	<b>8</b>
<b>6</b>	<b>Conclusão</b>	<b>11</b>
<b>7</b>	<b>Referências</b>	<b>12</b>

# 1 Introdução

A criptografia desempenha um papel fundamental na proteção da confidencialidade e integridade das informações transmitidas, sendo essencial em diversos contextos, como comunicação segura e armazenamento de dados sensíveis. No entanto, a segurança de sistemas criptográficos pode ser comprometida quando as chaves de criptografia ou os mecanismos de proteção são mal implementados ou descobertos por atacantes. Este trabalho aborda um cenário de criptoanálise, onde uma mensagem cifrada foi interceptada, e a chave utilizada para cifrá-la está criptografada com a técnica de RSA, sendo adicionalmente protegida por AES-256.

Neste contexto, o desafio é recuperar a chave que foi usada para criptografar a mensagem, levando em consideração que a chave RSA utilizada para protegê-la também está cifrada com AES-256, e que a senha usada para cifrar essa chave foi parcialmente descoberta. Utilizando técnicas de criptoanálise e ferramentas como o OpenSSL, exploramos a possível quebra das camadas de criptografia para acessar a chave original, que permitirá a decifração da mensagem interceptada. A resolução deste problema exige o uso de conceitos de criptografia simétrica e assimétrica, hash, e uma análise cuidadosa das informações disponíveis, como a hash da senha e as partes da chave pública RSA.

# 2 Arquivos Analisados

Primeiramente visualizamos os arquivos que nos foi entregue para realizar a decodificação.

1. `message.en` está encriptada com `key_for_message.en` (usando AES 256 bits).
2. `key_for_message.en` foi encriptado usando uma chave pública `key_public.en` (usando RSA).
3. A `key_public.en` foi encriptada usando AES 256 bits com uma chave que sabemos ter 12 letras e tem o seguinte formato: `m i - - - - -`.
4. Temos o hash da chave para quebrar a chave pública RSA no arquivo `key_for_rsa_public.hash` (codificado em base 64).

Com essas informações seguimos um roteiro para realizar cada etapa para conseguir revelar as chaves e a mensagem final.

### 3 Key\_Public e Hash

Para decriptar a chave pública (`key_public.en`), precisávamos primeiro quebrar o hash da senha que havia sido utilizado para cifrar essa chave com AES 256. Sabíamos que a senha estava criptografada com AES 256 bits, e que a chave para a decriptação tinha 12 caracteres. No entanto, apenas os dois primeiros caracteres da senha, `mi`, foram descobertos por um de nossos agentes antes de ser capturado.

O próximo passo foi utilizar o hash fornecido no arquivo `key_for_rsa_public.hash`, que está codificado em base64. Sabíamos disso porque o hash terminava com o caractere `=`, um indicativo comum de codificação em base64. Para facilitar a análise, utilizamos uma calculadora para converter a string codificada em base64 para seu formato hexadecimal.

Base64: `t8TT9lwdQ4sYw0ibxFQN6dXxjnOGuusGHbEcRVNX8BM=`

Hexadecimal: `b7c4d3f65c1d438b18c3489bc4540de9d5f18e7386baeb061db11c455357f013`

#### 3.0.1 Hashcat e Descriptografia da `key_public.en`

Com o valor da hash convertido para hexadecimal, o próximo passo foi tentar descobrir a senha de 12 caracteres. Para isso, utilizamos o **Hashcat**, uma das ferramentas mais poderosas e amplamente utilizadas para quebra de senhas a partir de hashes. O Hashcat emprega diversas técnicas avançadas, incluindo ataques de força bruta, dicionário e baseados em regras, para identificar a senha correspondente a um determinado hash.

O processo foi executado em um dos nossos computadores, equipado com uma **GTX-1650**. No entanto, como mostrado na imagem abaixo, o tempo estimado para a conclusão era bastante elevado. Após aproximadamente **18 horas de execução**, apenas **43% das combinações possíveis** haviam sido testadas, evidenciando a complexidade e o tempo necessário para a quebra da senha.

```
Checkpoint enabled. Will quit at next restore-point update.

Session.....: hashcat
Status.....: Aborted (Checkpoint)
Hash.Mode.....: 1400 (SHA2-256)
Hash.Target.....: b7c4d3f65c1d438b18c3489bc4540de9d5f18e7386baeb061db...57f013
Time.Started.....: Tue Feb 18 04:00:23 2025 (18 hours, 52 mins)
Time.Estimated...: Thu Feb 20 00:22:34 2025 (1 day, 1 hour)
Kernel.Feature...: Optimized Kernel
Guess.Mask.....: 'mi?l?l?l?l?l?l?l?l?l' [14]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 865.3 MH/s (7.60ms) @ Accel:128 Loops:26 Thr:256 Vec:1
Speed.#2.....: 16784.5 kH/s (60.55ms) @ Accel:512 Loops:26 Thr:8 Vec:4
Speed.#*.....: 882.0 MH/s
Recovered.....: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.....: 60228848902144/141167095653376 (42.66%)
Rejected.....: 0/60228848902144 (0.00%)
Restore.Point....: 2316489601024/5429503678976 (42.66%)
Restore.Sub.#1...: Salt:0 Amplifier:0-26 Iteration:0-26
Restore.Sub.#2...: Salt:0 Amplifier:0-26 Iteration:0-26
Candidate.Engine.: Device Generator
Candidates.#1....: 'mintzmsyhmoy' -> 'miwkaggbkngt'
Candidates.#2....: 'minnkeshwysd' -> 'miwbpcunfysd'
Hardware.Mon.#1...: Temp: 80c Util: 97% Core:1785MHz Mem:6000MHz Bus:8
Hardware.Mon.#2...: N/A

Started: Tue Feb 18 03:58:24 2025
Stopped: Tue Feb 18 22:53:15 2025
```

Figura 1: Execução do Hashcat exibindo o tempo estimado para quebra do hash utilizando força bruta.

Como o tempo estimado para a quebra da chave neste computador era muito alto, optamos por utilizar uma máquina mais potente. Com esse novo equipamento, conseguimos recuperar a chave em aproximadamente 3 horas de operação.

```
b7c4d3f65c1d438b18c3489bc4540de9d5f18e7386baeb061db11c455357f013:mijumperjohn
Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 1400 (SHA2-256)
Hash.Target.....: b7c4d3f65c1d438b18c3489bc4540de9d5f18e7386baeb061db...57f013
Time.Started.....: Sun Feb 16 21:38:03 2025 (3 hours, 33 mins)
Time.Estimated...: Mon Feb 17 01:11:57 2025 (0 secs)
Kernel.Feature...: Optimized Kernel
Guess.Mask.....: mi?l?l?l?l?l?l?l?l?l [12]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 3481.4 MH/s (75.30ms) @ Accel:64 Loops:676 Thr:256 Vec:1
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 41989311037440/141167095653376 (29.74%)
Rejected.....: 0/41989311037440 (0.00%)
Restore.Point....: 62113972224/208827064576 (29.74%)
Restore.Sub.#1...: Salt:0 Amplifier:0-676 Iteration:0-676
Candidate.Engine.: Device Generator
Candidates.#1....: minipreujohn -> miqfsqrhvaay
Hardware.Mon.#1...: Temp: 67c Fan: 76% Util:100% Core:2670MHz Mem:8250MHz Bus:8
Started: Sun Feb 16 21:37:59 2025
Stopped: Mon Feb 17 01:11:57 2025
```

Figura 2: Execução do Hashcat exibindo o tempo estimado para quebra do hash utilizando força bruta.

```
hashcat -a 3 -m 1400 key_for_rsa_public.hex 'mi?!?!?!?!?!?!?!?!' -O -w 3
```

Onde:

- **-a 3**: Ataque por força bruta.
- **-m 1400**: Tipo do hash (SHA2-256).
- **-w 3 e -O**: Otimizações para melhorar a performance.

A partir deste comando conseguimos recuperar a senha chamada '**mijumperjohn**', que referencia John Michael Jumper, um cientista e investigador estadunidense da DeepMind Technologies. Com esta chave, conseguimos decriptar a chave pública **key\_public.en** com o OpenSSL a partir do comando:

```
openssl enc -d -nosalt -aes256 -a -in key_public.en -out public.key -k mijumperjohn
```

Quando executado, gerou a seguinte chave pública:

```
-----BEGIN PUBLIC KEY-----
MEwwDQYJKoZIhvcNAQEBBQADAwAwOAIxAK7SDSQVzmxUt9NWTASwSTFEgABuDcWq
8JlUx4nvi6dCkoXc2/mzGUHj1T/heywzNwIDAQAB
-----END PUBLIC KEY-----
```

Figura 3: Chave pública

Em seguida utilizamos o OpenSSL para extrair o módulo da chave publica, como visto abaixo:

```
C:\Program Files\OpenSSL-Win64\bin>openssl rsa -in "C:\Users\ghepp\OneDrive\Área de Trabalho\Computação AE03\public.key" -pubin -text -noout
Public-Key: (384 bit)
Modulus:
 00:ae:d2:0d:24:15:ce:6c:54:b7:d3:56:4c:04:b0:
 49:31:44:80:00:6e:0d:c5:aa:f0:9b:94:c7:89:ef:
 8b:a7:42:92:85:dc:db:f9:b3:19:41:e3:95:3f:e1:
 7b:2c:33:37
Exponent: 65537 (0x10001)
```

Figura 4: Módulo da Chave Pública

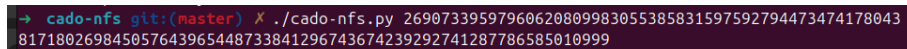
## 4 Fatoração Cado-NFS

O Cado-NFS (ou Cado-NFS) é uma implementação de um sistema de fatoração de números grandes utilizando o método de fatoração por corpos numéricos (Number Field Sieve, NFS). É uma ferramenta avançada usada principalmente em criptografia e segurança de dados para encontrar fatores primos de números muito grandes, o que é um problema fundamental na quebra de criptografia de chaves públicas.

Com o módulo, utilizamos um programa em Python para fazer a conversão do número em hexadecimal para decimal, chegando no número:

$N = 26907339597960620809983055385831597592794473474178043817180269845057643965448733841$

Com essa ferramenta e o nosso valor de  $N$ , podemos dar início ao próximo passo, que seria a fatoração de  $N$ . O software levou aproximadamente 5 horas para concluir a tarefa.



```
→ cado-nfs glt:(master) X ./cado-nfs.py 26907339597960620809983055385831597592794473474178043817180269845057643965448733841296743674239292741287786585010999
```

Figura 5: Comando no Cado para fazer a fatoração

Com esse resultado em mãos, o próximo passo é reconstruir a chave privada. Para isso, desenvolvemos um script em Python inspirado por outro código que encontramos:

```

1  import gmpy2
2  from Crypto.PublicKey import RSA
3
4  def rsa_keyfromprimes(p, q, e):
5      n = gmpy2.mul(p, q)
6      phi = gmpy2.mul((p-1), (q-1))
7      d = gmpy2.invert(e, phi)
8      key = RSA.construct((int(n), int(e), int(d), int(p), int(q)))
9      return key
10
11 def main():
12     p = 4775857644794676250173788374034235557066651390842989010331
13     q = 5634033005001224569208144003165564895248210119299568609429
14     e = 65537
15
16     try:
17         key = rsa_keyfromprimes(p, q, e)
18         private_key_pem = key.export_key().decode('utf-8')
19
20         # Salvar a chave privada no arquivo "private.key"
21         with open("private.key", "w") as f:
22             f.write(private_key_pem)
23
24         print("✅ Chave privada salva como 'private.key'")
25     except Exception as e:
26         print(f"Erro ao gerar a chave: {e}")
27
28 if __name__ == "__main__":
29     main()
30

```

Figura 6: Programa para recuperação da chave privada

Resultando na chave:

```

|-----BEGIN RSA PRIVATE KEY-----
MIHZAEEAAjEArtINJBXObFS301ZMBLBJMUSAAG4Nxarwm5THie+Lp0KShdzb+bMZ
QeOVP+F7LDM3AgMBAAECMAZ6CbqYhDlH0jg+LjrOPSfyHbvmV+RG30tsGLhCCg8r
5GsMZ3hml2gvNKOZ7WYX4QIZAMLGR8X/lvXkRntwSL+20eDsDqJm4zVdmwIZA0XG
DJdc8ZzrNBzk7dEtZMeY98TaOX2olQIZAINbxDYVrRMctEI1tOq0lDRMH4aJizyH
dwIZAMM1NPFLV0GXWVjK7Xhu9lXUKxs+YuVs2QIYAQt4rTy/AH8hfG18y2LNWE/D
y/j6BQ+A
-----END RSA PRIVATE KEY-----

```

Figura 7: Chave Recuperada



## 5 Encontrar a Mensagem Final

Com a chave privada em mãos, seguimos para o último passo, que seria descriptografar a mensagem. A partir do comando:

```
C:\Program Files\OpenSSL-Win64\bin>openssl pkeyutil -decrypt -inkey "C:\Users\ghepp\OneDrive\Documents\Cripto\private.key" -in "C:\Users\ghepp\OneDrive\Documents\Cripto\key_for_message.em" -out "C:\Users\ghepp\OneDrive\Documents\Cripto\password_message"
C:\Program Files\OpenSSL-Win64\bin>type "C:\Users\ghepp\OneDrive\Documents\Cripto\password_message"
johnhopfield
```

Figura 8: Comando para Descriptografar

Obtivemos a senha chamada 'johnhopfield'. John Joseph Hopfield é um físico, biólogo e neurologista estadunidense. Foi distinguido em 2024 com o Prémio Nobel de Física, conjuntamente com Geoffrey Hinton, por "descobertas fundamentais e invenções que permitem aprendizagem automática com redes neurais artificiais". Assim, utilizamos essa senha no comando:

```
C:\Program Files\OpenSSL-Win64\bin>openssl enc -aes-256-ctr -d -a -nosalt -in "C:\Users\ghepp\OneDrive\Documents\Cripto\message.em" -out "C:\Users\ghepp\OneDrive\Documents\Cripto\message.de" -pass pass:johnhopfield
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
C:\Program Files\OpenSSL-Win64\bin>type "C:\Users\ghepp\OneDrive\Documents\Cripto\message.de"
```

Figura 9: Senha no Comando

Com a mensagem destrancada, podemos finalmente visualizar o texto que existia ali:

```
Quote https://techcrunch.com/2024/12/10/google-says-its-new-quantum-chip-indicates-that-multiple-universes-exist/
Google says its new quantum chip indicates that multiple universes exist | TechCrunch
Julie Dart
00:04 minutos

Google on Monday announced Willow, its latest, greatest quantum computing chip. The speed and reliability performance claims GoogleQCs made about this chip were newsworthy in themselves, but what really caught the tech industryQCs attention was an even wilder claim tucked into the blog post about the chip.

Google Quantum AI founder Hartmut Neven wrote in his blog post that this chip was so mind-boggling fast that it must have borrowed computational power from other universes.

Ergo the chipQCs performance indicates that parallel universes exist and QCs live in a multiverse.QCs

HereQCs the passage:

WillowQCs performance on this benchmark is astonishing: It performed a computation in under five minutes that would take one of todayQCs fastest supercomputers 1025 or 10 septillion years. If you want to write it out, itQCs 10,000,000,000,000,000,000,000,000 years. This mind-boggling number exceeds known timescales in physics and vastly exceeds the age of the universe. It lends credence to the notion that quantum computation occurs in many parallel universes, in line with the idea that we live in a multiverse, a prediction first made by David Deutsch.

This drop-the-mic moment on the nature of reality was met with skepticism by some, but, surprisingly, others on the internet who profess to understand these things argued that NevenQCs conclusions were more than plausible. The multiverse, while stuff of science fiction, is also an area of serious study by the founders of quantum physics.

The skeptics, however, point out that the performance claims are based on the benchmark that Google itself created some years ago to measure quantum performance. That alone doesnQCs prove that parallel versions of you arenQCs running around in other universes QCs just where the underlying measuring sticks came from.

While classic digital computers that calculate based on whether a bit is a 0 or 1 (on or off), quantum computers rely on incredibly tiny qubits. These can be on/off or both (somewhere in between) and they can also tap into quantum entanglement QCs a mysterious connection at the tiniest levels of the universe between two or more particles where their states are linked, no matter the distance that separates them.

Quantum computers use such quantum mechanics to calculate highly complex problems that cannot currently be addressed with classic computers.

The problem is that the more qubits used in the computer, the more prone to errors they are. So itQCs not clear yet if quantum computers will ever be reliable enough and powerful enough to live up to their hype. GoogleQCs mission with Willow was to reduce those errors, and Neven says it accomplishes that.

C:\Program Files\OpenSSL-Win64\bin>type "C:\Users\ghepp\OneDrive\Documents\Cripto\message.de"
```

Figura 10: Mensagem Gerada

Texto formatado gerado no arquivo:

Fonte: <https://techcrunch.com/2024/12/10/google-says-its-new-quantum-chip-indicates-that-multiple-universes-exist/>  
**Google says its new quantum chip indicates that multiple universes exist — TechCrunch**

Julie Bort  
3–4 minutos

Google on Monday announced Willow, its latest, greatest quantum computing chip. The speed and reliability performance claims Google’s made about this chip were newsworthy in themselves, but what really caught the tech industry’s attention was an even wilder claim tucked into the blog post about the chip.

Google Quantum AI founder Hartmut Neven wrote in his blog post that this chip was so mind-bogglingly fast that it must have borrowed computational power from other universes.

Therefore, the chip’s performance indicates that parallel universes exist and “we live in a multiverse.”

Here’s the passage:

Willow’s performance on this benchmark is astonishing: It performed a computation in under five minutes that would take one of today’s fastest supercomputers 10<sup>25</sup> or 10 septillion years. If you want to write it out, it’s 10,000,000,000,000,000,000,000,000 years. This mind-boggling number exceeds known timescales in physics and vastly exceeds the age of the universe. It lends credence to the notion that quantum computation occurs in many parallel universes, in line with the idea that we live in a multiverse, a prediction first made by David Deutsch.

This drop-the-mic moment on the nature of reality was met with skepticism by some, but, surprisingly, others on the internet who profess to understand these things argued that Neven’s conclusions were more than plausible. The multiverse, while stuff of science fiction, is also an area of serious study by the founders of quantum physics.

The skeptics, however, point out that the performance claims are based on the benchmark that Google itself created some years ago to measure quantum performance. That alone doesn’t prove that parallel versions of you aren’t running around in other universes — just where the underlying measuring stick came from.

Unlike classic digital computers that calculate based on whether a bit is 0 or 1 (on or off), quantum computers rely on incredibly tiny qubits. These can be on/off or both (somewhere in between), and they can also tap into quantum entanglement — a mysterious connection at the tiniest levels of the universe between two or more particles where their states

are linked, no matter the distance that separates them.

Quantum computers use such quantum mechanics to calculate highly complex problems that cannot currently be addressed with classic computers.

The problem is that the more qubits used in the computer, the more prone to errors they are. So it's not clear yet if quantum computers will ever be reliable enough and powerful enough to live up to their hype. Google's mission with Willow was to reduce those errors, and Neven says it accomplishes that.

## 6 Conclusão

Neste trabalho, exploramos a criptoanálise de uma mensagem cifrada utilizando técnicas avançadas de criptografia, como AES-256 e RSA. Nosso objetivo foi demonstrar como as chaves e mensagens criptografadas podem ser recuperadas através de um processo cuidadoso e metódico, mesmo diante de desafios como a cifragem de chaves com algoritmos robustos e a presença de hashes.

Ao longo do processo, fomos capazes de quebrar a cifra AES-256 utilizada para proteger a chave pública RSA, utilizando a informação parcial da senha fornecida por nosso agente. Com a chave privada RSA em mãos, foi possível recuperar a chave de criptografia da mensagem e, finalmente, descriptografar o conteúdo.

O uso de ferramentas como Cado-NFS para a fatoração de números grandes foi fundamental para a obtenção dos fatores primos necessários para quebrar a chave pública RSA. A partir dessa fatoração, conseguimos avançar na recuperação da chave privada e, conseqüentemente, na descriptografia da mensagem.

A descoberta da mensagem, que envolvia uma referência ao chip quântico da Google, ilustra a relevância da criptografia no mundo moderno e a complexidade envolvida na proteção de informações sensíveis. A realização de tarefas como essa demonstra a importância de compreender os fundamentos da criptografia, da matemática e das técnicas de criptoanálise, além de ressaltar o papel crucial da segurança digital na proteção de dados.

Por fim, este trabalho destacou não apenas a teoria por trás dos algoritmos de criptografia, mas também a aplicação prática de técnicas avançadas para quebrar sistemas de segurança e recuperar informações criptografadas. O estudo da criptoanálise continua sendo uma área de pesquisa vital para a evolução da segurança digital e para o avanço das tecnologias de computação.

## 7 Referências

### Referências

- [1] OpenSSL. *Open Source Toolkit for SSL/TLS*. Disponível em: <https://www.openssl.org/>. Acesso em: 18 fev. 2025.
- [2] CADO-NFS. *An Implementation of the Number Field Sieve (NFS) Algorithm*. Disponível em: <https://cado-nfs.gitlabpages.inria.fr/>. Acesso em: 18 fev. 2025.
- [3] Hashcat. *Advanced Password Recovery*. Disponível em: <https://hashcat.net/hashcat/>. Acesso em: 18 fev. 2025.