Logotipo

Descrição gerada automaticamente com confiança baixa

Relatório de CTF

miniRSA – PicoCTF

|  |  |
| --- | --- |
| **Informações do documento** | |
| **Referência** | miniRSA – Guilherme Gonsales de Sá |
| **N° Revisão** | 1 |
| **Data de publicação** | 08/09/2025 |
| **Link** | [picoCTF - picoGym Challenges](https://play.picoctf.org/practice/challenge/73?category=2&page=6&solved=0) |

|  |  |  |
| --- | --- | --- |
| **Redação** | Guilherme Gonsales de Sá | Estudante |
| **Revisão** | Nome do revisor | Orientador |
| **Aprovação** | Nome do aprovador | Diretor |

|  |  |  |
| --- | --- | --- |
| **Histórico de revisões** | | |
| **N°** | **Entregas** | **Descrição** |
| **0** | 07/08/2025 | Produção |
| **1** | DD/MM/AAAA | Revisão |
| **2** | DD/MM/AAAA | Aprovação |

|  |  |
| --- | --- |
| **Informações do CTF** | |
| **Nível de Dificuldade** | Difícil |
| **Tipo de acesso** | Gratuito |
| **Conceitos envolvidos** | Criptografia |
| **Plataforma** | PicoCTF |
| **Área** | Red |

**Sumário**

Sumário

[Contextualização 2](#_Toc208263063)

[Desenvolvimento 2](#_Toc208263064)

[Conclusão 6](#_Toc208263065)

[Referências 6](#_Toc208263066)

[Scripts 6](#_Toc208263067)

Contextualização

O CTF miniRSA pratica um dos mais fundamentais conceitos da criptografia moderna: a criptografia RSA. Nele, somos expostos a uma vulnerabilidade específica desse tipo de encriptação, além de aprender alguns ‘rabbit holes’ que podem ser encontrados quando tentando quebrar essa cifra.

Desenvolvimento

**Let's decrypt this:** [**ciphertext**](https://jupiter.challenges.picoctf.org/static/eb5e6df8e14c52873cf88c582a1a4008/ciphertext)**? Something seems a bit small.**

De início, somos mostrados um arquivo texto, onde tem-se as seguintes informações:

N: 29331922499794985782735976045591164936683059380558950386560160105740343201513369939006307531165922708949619162698623675349030430859547825708994708321803705309459438099340427770580064400911431856656901982789948285309956111848686906152664473350940486507451771223435835260168971210087470894448460745593956840586530527915802541450092946574694809584880896601317519794442862977471129319781313161842056501715040555964011899589002863730868679527184420789010551475067862907739054966183120621407246398518098981106431219207697870293412176440482900183550467375190239898455201170831410460483829448603477361305838743852756938687673

e: 3

ciphertext (c): 2205316413931134031074603746928247799030155221252519872650080519263755075355825243327515211479747536697517688468095325517209911688684309894900992899707504087647575997847717180766377832435022794675332132906451858990782325436498952049751141

Compreende-se imediatamente – com o auxílio do título - que estamos tratando de uma cifra RSA. Vejamos brevemente um esquemático de como funciona a criptografia RSA.

Com certeza. Aqui está a tabela resumindo o processo matemático da criptografia RSA:

|  |  |  |
| --- | --- | --- |
| Etapa | Operação Matemática | Descrição |
| **Geração de Chaves** |  |  |
|  | n=p×q | Cria o módulo, que é público. |
|  | ϕ(n)=(p−1)×(q−1) | Calcula a função totiente, mantida em segredo. |
|  | mdc(e, ϕ(n) )=1 | Escolhe um expoente público e. |
|  | d\*e≡1(mod ϕ(n) ) | Calcula o expoente privado d. |
| **Chave Pública** | (e,n) | Usada para encriptar. |
| **Chave Privada** | (d,n) | Usada para decriptar. |
| **Encriptação** | C=M^e(mod n) | Transforma a mensagem M no texto cifrado C. |
| **Decriptação** | M=C^d(mod n) | Recupera a mensagem original M a partir de C. |

Dessa forma, já temos algumas informações: a chave pública, a mensagem cifrada e o módulo.

Comecemos com uma abordagem padrão: descobrir os fatores, *p e q,* para calcular a função totiente de euler.

Observa-se imediatamente que o módulo n é extremamente grande, sendo de fatoração muito complexa. Por sorte, após uma breve pesquisa, descobre-se que esse número já é conhecido, com nome de ***RSA-250.*** Tal número já foi fatorado, e seus fatores são:

P = 6413528947707158027879019017057738908482501474294344720811685963202453234463023862359875266834770873766192558569463979456518169520706660338379439831725461991821289469135013744951525049969894975003310925148008828986380633652876333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333

Q = 45750555163573684123935124254035984266882104482153361811332467451314902621458997324316686362845344216867393223305433144181363715891163136254413342416392471688838392576214433383515413305225919100373076119425887763639943171363333333333333333333333333333333333333333333333333333333333333333333333333333333

Com isso, pode-se calcular o resultado da função totiente, sendo esse:

29331922499794985782735976045591164936683059380558950386560160105740343201513369939006307531165922708949619162698623675349030430859547825708994708321803705309459438099340427770580064400911431856656901982789948285309956111848686906152664473350940486507451771223435835260168971210087470894448460745593956840586419174152069199349931839211312124535356911634803732328424874452150532289233019344436893431835133573867623353533923521293233628441113130129219967113122104184310328222225343411617477626101416178339803064430489246335134640103632345221332851151323142211599321453969826379968032542456108424169382211326227221061328

Com isso, estamos preparados para calcular a chave privada *d*, dado que possuímos a chave privada *e*. No entanto, aqui destaca-se um detalhe crucial para a solução do CTF. Para expoentes públicos grandes, é natural assumir que a operação *M^e* seja maior que *n*, de forma com que a operação modular seja composta de um *k* inteiro qualquer que multiplica *n* somado de um resto *r,* sendo, portanto:

Dessa forma, ao tentarmos descobrir *d*, em parte considerável dos scripts disponíveis na internet, a divisão modular por *n* retornará um erro matemático, sendo este ‘impossível dividir por 0’. Isso foi testado tanto com o método padrão para decriptação de chaves RSA (exposto acima, apenas aplicando o inverso modular) tanto com o método do *Chinese Remainder Theorem (CRT),* método mais complexo que é aplicável para a decriptação de chaves RSA.

Após esses ‘*becos sem saída’*, o CTF aparenta ser impossível de ser resolvido.

No entanto, existe um ângulo de ataque possível: o *Ataque de Expoente Baixo.* Tal ataque consiste na premissa que é menor que *n*. Isso implica que a divisão modular pelo módulo *n* torna-se irrelevante, dado que o resto será o próprio resultado de , pois .

Dessa forma, pode-se converter a equação para . Nessa forma, conhecemos duas das três incógnitas necessárias, podendo apenas isolar *m.* De maneira geral, possuindo tanto *c* quanto *e*, deve-se calcular a raiz *e-ésima* de *c,* ou seja:

Com isso, apenas é necessário calcular a raiz cúbica de *c* e converter o resultado numérico para bytes, de forma que forme um texto legível.

Após a confecção de um script para tal, encontra-se a flag, concluindo o desafio.



Conclusão

Embora a solução seja simples, o CTF demonstra que a solução padrão, ou até mesmo uma mais complexa, não necessariamente é a correta. O desafio nos obriga a de fato compreender de forma mais profunda como a matemática de chaves RSA funciona e como ela normalmente é realizada nos computadores, além de apresentar uma vulnerabilidade extrema desse tipo de cifra e como podemos explora-lo.

Referências

[RSA (sistema criptográfico) – Wikipédia, a enciclopédia livre](https://pt.wikipedia.org/wiki/RSA_(sistema_criptogr%C3%A1fico))

[Chinese remainder theorem - Wikipedia](https://en.wikipedia.org/wiki/Chinese_remainder_theorem)

[Google Gemini](https://gemini.google.com/app) - tabela

Scripts

