Métodos Clássicos de Fatoração para Tentativa de Quebra o RSA

Prof. Me. Bryan Kano
Prof. Dr. Routo Terada

Roteiro

Objetivo da aula

Tema central:

- Como **métodos clássicos de fatoração** podem ser usados para **quebrar o RSA**.
- **Importante:** apenas computação clássica (sem algoritmos quânticos).

Pergunta: O que significa "quebrar" o RSA?

- Descobrir a chave privada d a partir da chave pública (n, e);
- O caminho "natural" (melhor conhecido hoje): **fatorar n em p e q**.

RSA e fatoração

Lembrando a estrutura:

- $n = p \cdot q$ (produto de dois primos grandes);
- $\varphi(n) = (p-1)(q-1)$;
- e e d satisfazem:

$$e \cdot d \equiv 1 \pmod{\varphi(n)}.$$

Leitura da congruência:

"e vezes d é congruente a 1 módulo $\varphi(n)$".

Consequência:

- Se eu descubro p e q, calculo $\varphi(n)$;
- Com $\varphi(n)$, encontro d via algoritmo de Euclides estendido;
- Logo, **fatorar** $n \Rightarrow$ **quebrar o RSA**.

Modelo de ataque (computação clássica)

Suposição do atacante:

- Conhece a chave pública (n, e);
- Tem acesso a computadores clássicos (CPU, GPU, clusters etc.);
- **Não** tem computador quântico.

Objetivo:

- Fatorar n em p e q de forma mais rápida possível;
- Usar o melhor algoritmo clássico disponível para o tamanho de n.

Complexidade de algoritmos de fatoração

Queremos comparar algoritmos por sua **complexidade**, isto é, o número de operações em função de n.

Notações comuns:

- Tempo **polinomial**: $O((\log n)^k)$;
- Tempo **exponencial**: $\exp(c(\log n))$ ou pior;
- Tempo **subexponencial**: entre polinomial e exponencial.

Para fatoração, usamos com frequência a notação:

$$L_n[\alpha, c] = \exp\big((c + o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}\big)$$

onde $0 \leq \alpha \leq 1$, $c > 0$.

Lendo a notação $L_n[\alpha, c]$

$$L_n[\alpha, c] = \exp\big((c + o(1))(\ln n)^{\alpha}(\ln \ln n)^{1-\alpha}\big)$$

Leitura em voz alta:

> *"L de n colchete alfa vírgula c é igual a exponencial de* $(c + o(1))$ *vezes log natural de n elevado a alfa vezes log log de n elevado a* $1 - \alpha$*".*

Significado:

- $o(1)$: termo que tende a 0 quando n cresce;
- α controla o "grau de subexponencialidade";
- Quanto menor α, melhor (mais próximo de polinomial).

Divisão por tentativa (trial division)

Ideia básica: testar fatores possíveis de n.

- Verificar se 2 divide n, depois 3, depois 5...
- Em geral, testar todos os inteiros até $\sqrt{n}$:

$$\text{Se } n = ab \text{ então } a \leq \sqrt{n} \text{ ou } b \leq \sqrt{n}.$$

Complexidade:

- Aproximadamente $O(\sqrt{n})$ divisões;
- Em termos de bits, isso é **exponencial** em $\log n$.

Uso prático:

- Descarta pequenos fatores;
- Não é viável para chaves RSA reais.

Fatoração de Fermat

Ideia: se $n = pq$ e p e q estão próximos, podemos escrever:

$$n = x^2 - y^2 = (x - y)(x + y).$$

Procedimento:

1. Começar com $x = \lceil \sqrt{n} \rceil$;
2. Calcular $x^2 - n$;
3. Verificar se $x^2 - n$ é um quadrado perfeito y^2;
4. Se sim, então:

$$p = x - y, \quad q = x + y.$$

Bom quando: p e q estão muito próximos.
Má notícia: RSA seguro escolhe primos de forma que isso não seja eficiente.

Algoritmo ρ de Pollard (Pollard's rho)

Ideia: usar uma sequência pseudoaleatória módulo n e explorar colisões.
Definimos uma função, por exemplo:

$$f(x) = x^2 + 1 \pmod{n}$$

Leitura:

"f de x é igual a x ao quadrado mais 1 módulo n".

Geramos a sequência:

$$x_{k+1} = f(x_k) \pmod{n}$$

E consideramos duas sequências:

$$\begin{cases} x_{k+1} = f(x_k) \pmod{n} \\ y_{k+1} = f(f(y_k)) \pmod{n} \end{cases}$$

Em cada passo, calculamos:

$$d_k = \gcd(|x_k - y_k|, n)$$

Intuição do ρ de Pollard

Por que funciona?

- Trabalhamos com a mesma sequência, mas um índice “lento” (x_k) e um “rápido” (y_k);
- Em um dos fatores p, as sequências entram em ciclo mais cedo;
- A diferença $x_k - y_k$ é múltiplo de p em algum momento;
- O $\gcd(|x_k - y_k|, n)$ “captura” esse divisor.

Complexidade esperada:

- Aproximadamente $O(n^{1/4})$ para um fator;
- Melhor que trial division, mas ainda longe de quebrar RSA moderno.

Método $p-1$ de Pollard

Ideia: explorar propriedades de $p-1$ quando ele é "liso" (smooth).
Escolhemos:

$$\text{um inteiro } a \in \mathbb{Z}_n^*, \quad \text{e um grande } B.$$

Calculamos:

$$M = \operatorname{lcm}(1, 2, 3, \ldots, B)$$

Depois calculamos:

$$g = \gcd(a^M - 1, n)$$

Leitura:

"g é o máximo divisor comum entre $a^M - 1$ e n".

Se $1 < g < n$, então g é um fator de n.

Por que o $p - 1$ funciona?

Suponha que p é um fator de n e que $p - 1$ só tem **fatores primos pequenos** (isto é, $p - 1$ é B-liso).

No grupo multiplicativo $\mathbb{Z}_p^*$:

- A ordem de qualquer elemento divide $p - 1$;
- Escolhendo M múltiplo de $p - 1$, temos:

$$a^M \equiv 1 \pmod{p}.$$

Então:

$$a^M - 1 \equiv 0 \pmod{p} \quad \Rightarrow \quad p \mid (a^M - 1).$$

Logo:

$$\gcd(a^M - 1, n) \text{ tende a revelar o fator } p.$$

Limitação: só é eficaz se **pelo menos um** dos fatores tiver $p - 1$ suficientemente liso.

Método de Curvas Elípticas (ECM)

Ideia: Generalizar o método $p - 1$ usando o grupo de pontos de uma **curva elíptica** sobre $\mathbb{Z}_p$.
Uma curva elíptica típica:

$$E : y^2 \equiv x^3 + ax + b \pmod{n}$$

Leitura:

"E é o conjunto de pontos (x, y) *tais que* y^2 *é congruente a* $x^3 + ax + b$ *módulo n".*

Idéia do algoritmo:

- Escolher aleatoriamente uma curva e um ponto P nela;
- Computar múltiplos kP no grupo;
- Em algum momento, o cálculo de uma operação de grupo exige dividir por um número que não possui inverso módulo n;
- O gcd desse denominador com n revela um fator.

Por que o ECM é importante?

Complexidade (em função do menor fator p):

$$\exp\left(\sqrt{(c+o(1))\ln p \ln\ln p}\right),$$

para uma constante c.

Leitura:

"Exponencial da raiz de log de p vezes log log de p vezes uma constante".

Interpretação:

- Muito eficiente para encontrar **fatores médios/pequenos** de n;
- Frequentemente usado como **pré-processamento** antes de métodos mais pesados;
- No contexto de RSA real, ajuda a eliminar fatores "acidentalmente pequenos" (o que não deveria acontecer numa boa geração de chaves).

Ideia geral dos métodos quadráticos

Família de algoritmos baseada em encontrar:

$$x^2 \equiv y^2 \pmod{n}, \quad x \not\equiv \pm y \pmod{n}.$$

Então:

$$n \mid (x^2 - y^2) = (x - y)(x + y),$$

e muitas vezes:

$$\gcd(x - y, n) \text{ ou } \gcd(x + y, n)$$

dá um fator não-trivial.

Leitura:

"x^2 é congruente a y^2 módulo n, mas x não é congruente a mais ou menos y".

Método de Dixon

Ideia:

- Escolher vários x aleatórios com $x^2 \bmod n$;
- Procurar valores de $x^2 \bmod n$ que sejam **fatoráveis em primos pequenos** ("liso");
- As fatorações produzem relações lineares sobre expoentes;
- Combinando relações, obtemos um produto que é um quadrado perfeito modulo n.

Pontos-chave:

- Primeiro método prático baseado em **smoothness** e sistemas lineares mod 2;
- Conceitualmente próximo do Quadratic Sieve e do Number Field Sieve.

Quadratic Sieve (QS)

Quadratic Sieve é uma versão mais eficiente do Dixon.

Ideia central:

- Escolher uma função quadrática, por exemplo

$$Q(x) = (x + \lceil \sqrt{n} \rceil)^2 - n$$

- Avaliar $Q(x)$ para vários x inteiros consecutivos;
- Procurar valores de $Q(x)$ que sejam produtos de primos de uma **base de fatores** (primos pequenos escolhidos antes);
- Construir um sistema linear sobre os expoentes (mod 2);
- Combinar relações para obter um quadrado perfeito.

Complexidade do Quadratic Sieve

A complexidade esperada do QS é subexponencial, do tipo:

$$L_n\left[\tfrac{1}{2}, 1\right] = \exp\Big((1 + o(1))\sqrt{\ln n \ln \ln n}\Big)$$

Leitura:

> *"L de n colchete um meio, um, é exponencial de (1 mais um termo pequeno) vezes a raiz quadrada de log de n vezes log log de n."*

Significado:

- Muito melhor que métodos exponenciais;
- Foi o melhor algoritmo prático para números de tamanho intermediário por muito tempo;
- Ainda é competitivo para números na faixa de até 100–120 dígitos (dependendo da implementação).

MPQS (Multiple Polynomial Quadratic Sieve):

- Usa **vários polinômios quadráticos** em vez de um só;
- Melhora a taxa de geração de valores "lisos";
- Na prática, é mais rápido que o QS original.

Há muitas variantes e otimizações:

- Block Lanczos / Wiedemann para resolver o sistema linear;
- Técnicas sofisticadas de "sieving".

Moral: Antes do Number Field Sieve, o QS/MPQS era **o** algoritmo de eleição para fatorações grandes.

Number Field Sieve (NFS) – visão geral

Hoje, o **Number Field Sieve (NFS)** é o algoritmo clássico mais rápido conhecido para fatorar números grandes "sem estrutura especial", como os módulos RSA.

Ideia de alto nível:

- Em vez de trabalhar só em $\mathbb{Z}$, trabalha-se em **corpos de números** (number fields);
- Escolhe-se um polinômio $f(x)$ com uma raiz aproximada de n;
- Consideram-se duas "visões" do mesmo número:
  - Uma no corpo numérico associado a f (lado algébrico);
  - Outra em $\mathbb{Z}$ (lado racional).
- Coletam-se relações "lisas" nos dois lados;
- Monta-se um sistema linear gigantesco e obtém-se uma congruência de quadrados.

Complexidade do NFS

A complexidade heurística do **General Number Field Sieve (GNFS)** é:

$$L_n\left[\frac{1}{3}, \left(\frac{64}{9}\right)^{1/3}\right]$$

Leitura:

"L de n colchete um terço, (64 sobre 9) elevado a um terço".

Traduzindo:

$$L_n\left[\frac{1}{3}, c\right] = \exp\Big((c + o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}\Big).$$

Significado:

- Muito mais rápido (assimptoticamente) que $L_n[1/2, c]$ do Quadratic Sieve;
- É o melhor algoritmo conhecido para fatorar módulos RSA grandes;
- Utilizado nos maiores recordes de fatoração clássica.

Variantes do NFS

Existem versões especializadas:

- **Special Number Field Sieve (SNFS):**
  - Para números com forma especial (por exemplo, $a^b \pm c$);
  - Mais rápido nesses casos.
- **General Number Field Sieve (GNFS):**
  - Para números gerais, como módulos RSA típicos.

Na prática:

- Implementações altamente otimizadas;
- Uso massivo de paralelismo e memória;
- Fatorar um RSA-1024 comum com NFS ainda é extremamente caro em recursos.

Resumo dos principais métodos clássicos

- **Tentativa/Trial division, Fermat:** só para números pequenos ou mal escolhidos.
- **Pollard ρ:** boa opção para achar fatores "médios".
- **Pollard $p-1$, ECM:** ótimos para achar fatores quando há alguma estrutura (ordens lisas, grupos pequenos).
- **Dixon, QS, MPQS:** subexponenciais, bons para tamanhos intermediários.
- **Number Field Sieve (NFS):** melhor algoritmo clássico conhecido para módulos RSA grandes.

Regime de uso típico (heurístico)

Muito grosseiro, mas didático:

- Até ~ 60 dígitos: métodos simples (Pollard ρ etc.) podem funcionar bem;
- ~ 60–110 dígitos: Quadratic Sieve / MPQS geralmente preferidos;
- Acima de ~ 110–120 dígitos: Number Field Sieve passa a ser a melhor escolha;
- Em qualquer faixa: ECM como pré-processamento para achar fatores médios.

Importante: Módulos RSA reais (1024 bits, 2048 bits, 3072 bits, etc.) estão bem além do que se pode fatorar rotineiramente com recursos normais hoje.

Mensagem final (computação clássica)

Resumo conceitual:

- Todos esses algoritmos são de **computação clássica**;
- Melhor algoritmo conhecido para **módulos RSA gerais** é o NFS;
- A segurança do RSA "cru" é baseada na suposição de que:

Nenhum algoritmo clássico consegue fatorar n grande em tempo viável.

Na prática:

- Usamos tamanhos de chave (ex: 2048 bits) para os quais NFS ainda é considerado caro demais;
- Se amanhã surgir um algoritmo clássico significativamente mais rápido, os parâmetros de RSA precisariam ser revisados.

Encerramento

Métodos de fatoração clássicos são a
"linha de frente" dos ataques matemáticos ao RSA.

Na próxima etapa, podemos estudar:
como implementar (ou simular) alguns desses ataques em laboratório.