Universidade de Aveiro Departamento de Matemática
2009

Helena Ingildo de Sá Queirós Leite

Ataques ao Sistema Criptográfico RSA

Universidade de Aveiro Departamento de Matemática
2009

Helena Ingildo de Sá Queirós Leite

Ataques ao Sistema Criptográfico RSA

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática e Aplicações, realizada sob a orientação científica do Professor Doutor Paulo José Fernandes Almeida, Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro

Dedico este trabalho ao meu marido, pelo incansável apoio.

o júri

presidente

Prof. Doutor Helmuth Robert Malonek
Professor Catedrático da Universidade de Aveiro

Prof. Doutor António José de Oliveira Machiavelo
Professor Auxiliar da Universidade do Porto

Prof. Doutor Paulo José Fernandes Almeida
Professor Auxiliar da Universidade de Aveiro (orientador)

agradecimentos

Agradeço:

- ao meu orientador (Paulo José Fernandes Almeida) pelas sugestões, acompanhamento e disponibilidade durante todo o processo de elaboração desta dissertação;

- a John Pollard e Carl Pomerance, por terem sido tão prestáveis e por me terem facultado os seus artigos;

- ao meu marido, pelo carinho e paciência.

palavras-chave

RSA, primalidade, factorização, criptografia.

resumo

O RSA é um sistema criptográfico de chave pública, inventado, em 1978, por Rivest, Shamir e Adleman. Neste trabalho, serão abordadas várias técnicas desenvolvidas desde então para quebrar este sistema. Iremos descrever vários métodos de factorização, de onde destacamos o crivo quadrático, o crivo geral dos corpos de números e o método das curvas elípticas. Iremos também estudar vários ataques ao RSA que, de forma a serem evitados, vieram a permitir uma implementação mais adequada do RSA. Destes ataques destacamos aqueles que quebram o RSA quando o expoente público ou o expoente privado são demasiado pequenos, o ataque dos tempos de Kocher e o ataque com parte da chave privada exposta. Muitos dos métodos descritos são acompanhados, em apêndice, com um algoritmo, construído no software Maple 9.5.

keywords RSA, primality, factorization, cryptography.

abstract The RSA is a cryptographic system invented in 1978 by Rivest, Shamir and Adleman. In this work, we will study several methods developed since then to break this system. We will describe some factorization methods, of which we highlight the quadratic sieve, the general number field sieve and the elliptic curve method. We will also study several attacks to RSA that, in order to avoid them, a better implementation of the RSA was achieved. In particular, we will describe those that break the RSA when a small public exponent or a small private exponent is used. We will also see the Kocher's timing attack and partial private key exposure attack. Many of the methods are accompanied, in the appendix, by an algorithm constructed using the software Maple 9.5.

Conteúdo

1 Preliminares **4**

1.1 Introdução 4

1.2 Conceitos de Teoria dos Números 6

2 Criptografia **10**

2.1 Criptografia de chave simétrica 10

2.2 Criptografia de chave pública 12

2.2.1 O sistema criptográfico RSA 15

3 Complexidade **22**

3.1 Introdução 22

3.2 Estimativas de tempo 27

3.2.1 RSA 32

4 Primalidade **34**

4.1 Teste de Fermat 35

4.2 Teste de Miller-Rabin 38

4.3 Teste n-1, de Lucas 42

5 Ataques mais conhecidos ao RSA **44**

5.1 Ataques de factorização 44

5.1.1 Má escolha de n (factorização de Fermat, método $p-1$ de Pollard) 48

5.1.2 Método ρ (método de Monte Carlo) 57

5.1.3 Crivo quadrático (*Quadratic Sieve*) 60

5.1.4 Crivo geral dos corpos de números (*Number Field Sieve*) 64

5.1.5 Método curvas elípticas - factorização de inteiros, segundo Lenstra 77

5.2 Ataques de implementação do RSA 81

5.2.1 Ataque da procura exaustiva
(*Forward search attack* ou *Short plaintext attack*) 81

5.2.2 Ataque do módulo comum
(*Common modulus attack*) 82
5.2.3 Ataque do ponto fixo
(*Fixed-point attack*, *Cyclic attack* ou *Superencryption attack*) . . 84
5.2.4 Expoente público pequeno 86
5.2.5 Expoente privado pequeno 92
5.2.6 Ataque dos tempos de Kocher
(*Kocher's timing attack*) 97
5.2.7 Ataque com parte da chave privada exposta
(*Partial private key exposure attack*) 100

A **105**

B Tabela ASCII **109**

C Teste de Fermat (algoritmo) **111**

D Teste de Miller-Rabin (algoritmo) **113**

E Método ρ de Pollard (algoritmo) **115**

F Factorização de Fermat (algoritmo) **116**

G Crivo quadrático (algoritmo) **117**

H Curvas elípticas - noções gerais **119**

I Método curvas elípticas (algoritmo) **133**

J Expoente privado pequeno (algoritmo) **137**

K Ataque com parte da chave privada exposta (algoritmo) **139**

Capítulo 1

Preliminares

1.1 Introdução

Desde muito cedo, o Homem sentiu necessidade de desenvolver técnicas para esconder mensagens de curiosos ou de inimigos, de forma a garantir que apenas o destinatário as lê. Essa necessidade aumentou ainda mais em tempos de guerra. Ao longo da História, a forma como o Homem impediu que os curiosos e os inimigos lessem as mensagens evoluiu: desde esconder a mensagem (o que não impedia que se o atacante soubesse onde estava escondida a mensagem, a lesse), substituir ou transpor os caracteres da mensagem original, a formas de esconder a mensagem mais elaboradas, com recurso às potencialidades dos computadores. Em qualquer das situações, os objectivos são simples: confidencialidade (apenas o destinatário autorizado poderá decifrar a mensagem), integridade (o destinatário deve ser capaz de saber se a mensagem foi alterada durante a transmissão), autenticação do rementende (o destinatário deverá ser capaz de identificar o remetente e verificar se foi mesmo ele que enviou a mensagem) e irretratabilidade (não deverá ser possível o emissor negar o envio da mensagem). Enquanto evoluíam as formas de esconder as mensagens, evoluíam também os estudos sobre as formas de descobrir as mensagens escondidas / secretas. Desenvolveram-se, assim, de forma paralela, a criptografia e a criptanálise, sendo que a primeira área estuda formas de esconder mensagens e a segunda estuda formas de decifrar mensagens. Existem dois tipos de criptografia: a *criptografia de chave simétrica* (a chave que usamos para decifrar é idêntica à chave que usamos para cifrar, sendo que o emissor e o receptor das mensagens têm que, anteriormente, combinar chaves de cifragem / decifragem) e a *criptografia assimétrica* ou *criptografia de chave pública*, descoberta por Diffie, Hellman e Merkle (o receptor cria e publica uma chave, para que o emissor cifre e envie mensagens de forma segura, e cria uma chave privada, distinta da anterior e apenas

do seu conhecimento, para decifrar as mensagens cifradas que lhe foram enviadas). Apesar de, actualmente, a criptografia de chave simétrica ser menos usada do que a criptografia de chave pública, salientamos o facto de o algoritmo AED (*Advanced Encryption Standard*), que veio substituir o algoritmo DES (*Data Encryption Standard*), ser um dos algoritmos mais populares usados em criptografia de chave simétrica.

Em 1978, Rivest, Shamir e Adleman surpreenderam a comunidade científica com a descoberta de uma nova forma de cifrar e decifrar mensagens de forma suficientemente segura, utilizando alguns conhecimentos de Teoria dos Números - tinha surgido o RSA, uma forma de criptografia de chave pública. Desde essa altura, e até aos dias de hoje, muitos são os que se dedicam a quebrar o RSA, ou seja, conseguir ler mensagens originais, sem serem os destinatários das mesmas e sem receberem a chave de decifração. É precisamente sobre este tipo de "ataques"que nos propomos a abordar.

Para melhor compreendermos o trabalho desenvolvido, precisamos de rever alguns conceitos de Teoria dos Números (capítulo 1). A maior parte dos teoremas enunciados neste capítulo são demonstrados em [14]. Seguidamente, no capítulo 2, iremos distinguir criptografia simétrica da criptografia assimétrica e, nesta, iremos descrever o sistema criptográfico RSA. Como, para trabalharmos neste sistema criptográfico de forma eficiente, precisamos de trabalhar com números compostos n, decomponíveis em apenas dois factores primos grandes, importa descrevermos alguns testes de primalidade (capítulo 4), para que possamos tirar conclusões sobre se um determinado número é primo ou não. Sendo assim, iremos abordar dois testes de primalidade probabilísticos - o Teste de Fermat e o Teste de Miller-Rabin - e um teste de primalidade determinístico - o teste $n-1$, de Lucas. Estando reunidas as condições para melhor compreendermos como funcionam os ataques ao RSA (capítulo 5), iremos, numa primeira fase, descrever alguns ataques de factorização (factorização de Fermat; método $p-1$, de Pollard; método ρ; crivo quadrátivo; crivo geral dos corpos de números (*number field sieve*); e método das curvas elípticas) e, posteriormente, alguns ataques de implementação do algoritmo do RSA (ataque da procura exaustiva, ataque do ponto fixo, expoente público pequeno, expoente privado pequeno, ataque do módulo comum, ataque dos tempos de Kocher e ataque com parte da chave privada exposta). Praticamente todos estes testes de primalidade e estes ataques ao RSA são acompanhados, em apêndice, com um algoritmo, construído no software Maple 9.5, para que o leitor os possa testar e/ou acompanhar alguns exemplos que são dados ao longo da sua descrição. Para termos uma noção do tempo que se demora a utilizar um determinado teste de primalidade para averiguar a primalidade de um número p ou para averiguarmos o tempo que se demora

a implementar um determinado ataque ao sistema criptográfico RSA, necessitamos de algumas noções de complexidade, abordadas no capítulo 3.

1.2 Conceitos de Teoria dos Números

Definição (Divisibilidade)**.** Sejam a e b dois números inteiros. Se $a \neq 0$ e existe um inteiro c tal que $b = ac$, dizemos que *a divide b* ou que a é um *divisor* de b, ou ainda, que b é um *múltiplo* de a e escrevemos $a \mid b$. Se a não divide b, escrevemos $a \nmid b$.

Definição (Número primo)**.** A qualquer número inteiro maior do que 1 com apenas dois divisores inteiros positivos, o 1 e o próprio número, chamamos *número primo*. A um número inteiro maior do que 1 que não seja número primo diz-se *número composto*.

Definição (Máximo divisor comum)**.** Sejam a e b dois inteiros tais que pelo menos um deles é não nulo. Chamamos *máximo divisor comum* ao maior inteiro do conjunto dos divisores comuns de a e de b. Denotamos este elemento por $mdc(a,\, b)$.

Ao longo deste trabalho, iremos descrever algumas formas de factorizar números muito grandes. De facto, uma das áreas da Teoria dos Números consiste em encontrar métodos cada vez mais rápidos e eficazes para factorizar números compostos grandes. O Algoritmo de Euclides é um método que, de uma forma relativamente rápida, permite-nos encontrar o máximo divisor comum (mdc) entre dois números, desconhecendo-se qualquer um dos seus factores.

Para encontrarmos o $mdc(a,\, b)$, com $a > b$, através do algoritmo da divisão, procuramos os inteiros q_1 e r_1 tais que

$$a = bq_1 + r_1,$$

com $r_1 < b$. A seguir, utilizando o mesmo algoritmo, encontramos os inteiros q_2 e r_2 tais que

$$b = r_1 q_2 + r_2,$$

com $r_2 < r_1$. Novamente, procuramos os inteiros q_3 e r_3 tais que

$$r_1 = r_2 q_3 + r_3,$$

com $r_3 < r_2$. Continuamos sempre desta forma, dividindo o último quociente obtido pelo último resto obtido. Obtemos sempre novos quocientes e novos restos. Quando

chegarmos ao momento em que o último quociente divide o último resto, o processo termina. O último resto não nulo é o $mdc(a,\ b)$.

Seguindo este processo, e uma vez que o resto é sempre inferior ao divisor, vamos obter uma sequência de inteiros não negativos $r_1, r_2, r_3, \cdots, r_k, r_{k+1}$ tais que

$$0 \leq r_{k+1} < r_k < \cdots < r_2 < r_1,$$

pelo que o algoritmo de Euclides termina ao fim de um número finito de passos.

Teorema 1.1. *Sejam a e b números naturais. Se r_k é o último resto não nulo, obtido pelo algoritmo de Euclides, então $r_k = mdc(a,\ b)$.*

Além disso, o algoritmo de Euclides permite encontrar inteiros x e y tais que

$$ax + by = mdc(a,\ b).$$

Exemplo. *Queremos encontrar $mdc(5390,\ 1014)$. Aplicando o algoritmo de Euclides:*

$5390 = 1014 \times 5 + 320$

$1014 = 320 \times 3 + 54$

$320 = 54 \times 5 + 50$

$54 = 50 \times 1 + 4$

$50 = 4 \times 12 + 2$

$4 = 2 \times 2$

O $mdc(5390,\ 1014)$ é o último resto não nulo, ou seja, 2.

Teorema 1.2. *Se $mdc(n, a) = 1$ e $n \mid ab$, então $n \mid b$. Em particular, se p é primo e $p \mid ab$, então $p \mid a$ ou $p \mid b$.*

Demonstração: Pelo teorema 1.1, existem inteiros u e v tais que $nu + av = 1$. Então $nbu + abv = b$. Como $n \mid ab$, então $n \mid abv$. Como $n \mid nbu$ e $n \mid abv$, então $n \mid b$.

Para o caso particular apresentado, se $p \mid a$, está provado. Se considerarmos que $p \nmid a$, então $mdc(a, p) = 1$ e, como acabámos de provar, $p \mid b$. □

Definição (Congruente)**.** Sejam a e b inteiros e n um inteiro positivo. Dizemos que a é *congruente* com b módulo n, e escrevemos

$$a \equiv b \mod n,$$

se $n \mid (a - b)$, i.e., se existe um k inteiro tal que $a = b + kn$.

Teorema 1.3. *Sejam a, b e n inteiros.*

A congruência

$$ax \equiv b \mod n \tag{1.1}$$

tem soluções se e só se $d \mid b$, onde $d = mdc\,(a, n)$.

Se $d \mid b$, então a solução é única $\bmod \dfrac{n}{d}$.

Se $mdc(a, n) = 1$, então (1.1) tem uma solução que é única módulo n.

Definição (Inverso modular)**.** Sejam a e n inteiros tais que $mdc\,(a,\ n) = 1$. Ao único inteiro que é solução da equação

$$ax \equiv 1 \mod n$$

chamamos *inverso* de a mod n e denotamo-lo por a^{-1} mod n. Pelo teorema 1.3, este inteiro existe e é único módulo n.

Teorema 1.4 (Teorema Chinês do Resto)**.** *Sejam $n_1,\ n_2,\ \cdots, n_k$ inteiros positivos, primos entre si dois a dois. Sejam, ainda, $N = n_1 n_2 \cdots n_k$,*

$$N_i = \frac{N}{n_i},$$

e seja y_i o inverso multiplicativo de N_i módulo n_i.

Então, o sistema

$$\begin{cases} x \equiv a_1 \mod n_1 \\ x \equiv a_2 \mod n_2 \\ \vdots \\ x \equiv a_k \mod n_k \end{cases} \tag{1.2}$$

tem uma única solução x_0, módulo N, a saber $x_0 := \sum_{i=1}^{k} N_i\, y_i\, a_i$.

Definição (Sistema completo de resíduos)**.** Dizemos que os números

$$a_1, a_2, \cdots, a_n$$

formam um *sistema completo de resíduos* módulo n se

$$\{\overline{a_1}, \overline{a_2}, \cdots, \overline{a_n}\} = \mathbb{Z}/n\mathbb{Z},$$

isto é, se os a_i's representam todas as classes de congruência módulo n.

Exemplo. *Seja* n *um número inteiro. Então* $0,\ 1,\ 2,\ \cdots,\ n-1$ *formam um sistema completo de resíduos.*

Definição (Sistema reduzido de resíduos)**.** Seja $S = 0,\ 1,\ 2,\ \cdots,\ n-1$ um sistema completo de resíduos módulo n. Ao conjunto T formado pelos elementos de S que são primos com n chamamos *sistema reduzido de resíduos* módulo n.

Definição (Função de Euler)**.** Seja $n \geq 1$ um inteiro. O cardinal do sistema reduzido de resíduos módulo n é denotado por $\phi(n)$. Por outras palavras, $\phi(n)$ é o número de inteiros positivos menores ou iguais a n que são primos com n. Esta função de n é denominada *função* ϕ *de Euler.*

Facilmente se verifica que se p for um número primo, $\phi(p) = p - 1$.

Definição (Função multiplicativa)**.** Se uma função $f(n)$ está definida para todos os inteiros positivos, dizemos que $f(n)$ é multiplicativa se, para qualquer par de inteiros positivos m e n tais que $mdc(m, n) = 1$, se tem

$$f(mn) = f(m)f(n).$$

Teorema 1.5. *A função* $\phi(n)$ *é multiplicativa.*

Teorema 1.6 (Teorema de Euler)**.** *Se* $mdc(a, n) = 1$, *então*

$$a^{\phi(n)} \equiv 1 \mod n.$$

Teorema 1.7 (Pequeno Teorema de Fermat)**.** *Se* p *é primo, então*

$$a^p \equiv a \mod p,$$

para qualquer inteiro a.

Definição (Ordem de $a \mod n$)**.** Suponhamos que $mdc(a, n) = 1$. Definimos a ordem de $a \mod n$ (e denotamos por $ord_n(a)$) como sendo o menor inteiro positivo, digamos b, para o qual,

$$a^b \equiv 1 \mod n.$$

Do Teorema de Euler (1.6), deduzimos que sempre que $mdc(a, n) = 1$, $ord_n(a)$ existe e $ord_n(a) \leq \phi(n)$.

Teorema 1.8. *Se* $mdc(a, n) = 1$ *e se* $a^b \equiv 1 \mod n$, *para algum* $b > 0$, *então* $ord_n(a) \mid b$. *Em particular,*

$$ord_n(a) \mid \phi(n).$$

Reciprocamente, se $ord_n(a) \mid b$, *então* $a^b \equiv 1 \mod n$.

Capítulo 2

Criptografia

A palavra *criptografia* provém da junção das palavras gregas *kryptós*, que significa escondido ou secreto, e *grápheín*, ou escrita. Por outras palavras, trata-se do prática e do estudo da comunicação escondida ou secreta.

Apesar de comummente dar-se o mesmo significado a codificar e a cifrar, na realidade estes dois termos têm significados distintos. No processo de codificação/descodificação existe uma conversão de partes da informação (letras, palavras ou frases) para uma outra forma de representação, que não é necessariamente do mesmo tipo (pode ser outro alfabeto). Na cifragem/decifragem, existe um algoritmo (ou uma série de procedimentos lógicos) que transforma a mensagem original numa outra, ilegível (chamada *mensagem cifrada*), mas que utiliza o mesmo alfabeto. O receptor da mensagem deverá estar munido de um algoritmo que decifra a mensagem, isto é, que transforma a mensagem ilegível recebida na mensagem original. Com a cifragem/decifragem pretende-se que toda a pessoa que captar a mensagem cifrada e não possuir o algoritmo de decifragem, não consiga ler a mensagem original. No entanto, conforme iremos ver mais à frente, existem formas de descobrir a mensagem original, a partir da mensagem cifrada.

2.1 Criptografia de chave simétrica

Na *criptografia simétrica*, a chave para cifrar e para decifrar é a mesma ou idêntica, com alterações insignificantes, e é partilhada pelo emissor e pelo receptor. Desta forma, a chave que cifra é a "mesma" que decifra. Para garantir o secretismo da mensagem, apenas o emissor e o receptor podem ter conhecimento da chave de cifragem/decifragem

(ver Figura 2.1). Apesar de a vantagem da criptografia simétrica ser a rapidez da cifragem/decifragem, a desvantagem é o facto de o emissor e o receptor terem que encontrar-se (ou utilizar um canal de comunicação seguro) para partilharem a chave. O facto de cada um dos intervenientes no envio da mensagem ter que guardar a chave previamente combinada, torna-os um alvo fácil para um adversário criptográfico. Outra das desvantagens consiste no facto de que qualquer uma das partes pode alterar o conteúdo da mensagem, não havendo garantia que a mensagem recebida seja, de facto, a mensagem original.

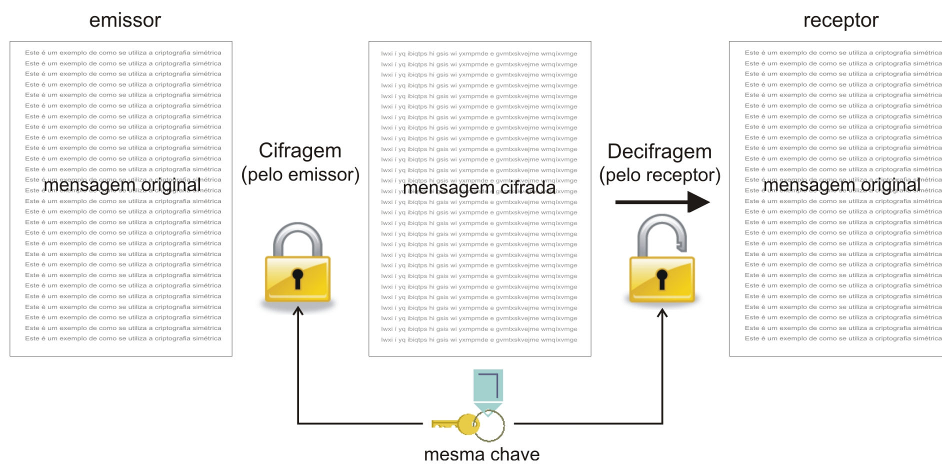


Figura 2.1: Na criptografia simétrica, a chave é partilhada pelo emissor e pelo receptor.

Um exemplo de criptografia simétrica é a *cifra de César*, também conhecida por *cifra de troca*, que consiste em trocar cada letra da mensagem original por uma outra que se encontra um determinado número fixo de letras abaixo da letra original. Por exemplo, uma troca de 3 posições implica que o "A"é substituído pelo "D", o "B"é substituído pelo "E", e assim sucessivamente. Se quisermos cifrar a mensagem "SECRETA", através da cifra de César, com deslocação de 3 posições, enviamos a mensagem "VHFUHWD". O emissor apenas tem que dizer ao receptor que a chave é 3. Recebida a mensagem cifrada, o receptor apenas tem que inverter a cifragem, ou seja, tem que trocar cada letra da mensagem cifrada por uma outra letra do mesmo alfabeto, 3 posições acima. Chega, assim, à mensagem "SECRETA". Repare-se que a chave utilizada foi praticamente a mesma.

Existem muitos tipos de criptografia simétrica, nomeadamente o algoritmo DES (em inglês *Data Encryption Standard*), mais tarde substituído pelo algoritmo AES (em inglês *Advanced Encryption Standard*), ainda usado actualmente. Estes algoritmos

são muito mais complexos que a *Cifra de César*, sendo que a principal desvantagem continua a ser o facto de o emissor e o receptor terem que partilhar a chave para cifrar/decifrar.

2.2 Criptografia de chave pública

A criptografia de chave pública foi uma descoberta de extrema importância.

Whitfield Diffie, Martin Hellman e Ralph Merkle dedicaram parte das suas vidas a tentar resolver o problema da distribuição da chave.

Singh [33] refere que:

> Em 1974, Diffie, ainda um criptógrafo itinerante, visitou o Laboratório Thomas J. Watson da IBM, onde fora convidado a dar uma conferência. Falou de várias estratégias para atacar o problema da distribuição da chave, mas todas as suas ideias eram incipientes, e a assistência mostrou-se céptica sobre as perpectivas de solução. A única resposta positiva à apresentação de Diffie partiu de Alan Konheim, um dos peritos criptográficos da IBM, que mencionou o facto de outra pessoa ter visitado o laboratório recentemente e ter dado uma conferência sobre o problema da distribuição da chave. O orador era Martin Hellman, um professor da Universidade de Stanford na Califórnia. Nessa noite, Diffie meteu-se no carro e deu início à viagem de 5000 quilómetros até à Costa Oeste, a fim de conhecer a única pessoa que parecia partilhar a sua obsessão. A aliança de Diffie e Hellman viria a constituir uma das associações mais dinâmicas no domínio da criptografia. (...)
>
> Uma vez que Hellman não dispunha de muito dinheiro, não podia dar-se ao luxo de dar trabalho como investigador à sua nova alma-gémea. Em vez disso, Diffie foi contratado como estudante de pós-graduação. Juntos, Hellman e Diffie começaram a estudar o problema da distribuição da chave (...). Algum tempo depois, Ralph Merkle foi juntar-se a eles. Merkle era um intelectual refugiado, saído de outro grupo de investigação cujo professor não nutria simpatia pelo sonho impossível de solucionar o problema da distribuição da chave.

Era frequente Diffie ter grandes momentos de meditação, até que um dia, a ideia veio-lhe à cabeça e quase se desvaneceu por completo [33]:

> «Desci as escadas para ir buscar uma Coca-Cola e quase me esqueci da ideia. Recordei-me que tinha estado a pensar em qualquer coisa interessante, mas não me conseguia lembrar do que era. Depois voltei, entusiasmado e com uma subida de adrenalina. Apercebia-me pela primeira vez, desde que trabalhava em criptografia, que tinha descoberto qualquer coisa realmente valiosa. Tudo o que havia descoberto sobre o assunto até àquele momento pareciam-me ser meros pormenores técnicos.» Estava-se a meio da tarde e ele tinha de esperar umas horas até Mary [a esposa] voltar. «Whit estava à minha espera à porta», recorda ela. «Disse que tinha uma coisa para me contar e estava com uma expressão estranha. Entrei e ele disse: "Senta-te, por favor, quero falar contigo. Acho que fiz uma grande descoberta - sei que sou a primeira pessoa a fazer isto". Naquele momento foi como se o mundo tivesse parado de girar. Senti-me como se estivesse a viver num filme de Hollywood.»

Tinha-se descoberto um novo tipo de cifra, que é suportada por uma chave assimétrica.

Na *criptografia assimétrica* ou *criptografia de chave pública*, utiliza-se um par de chaves distintas: a *chave pública*, que é a chave que permite cifrar a mensagem original, é do conhecimento público, e pode ser distribuída, por exemplo, por e-mail; e a *chave privada*, que permite decifrar a mensagem cifrada, e é apenas do conhecimento do receptor. Por outras palavras, para cifrarmos uma mensagem utilizamos a chave pública, mas apenas conseguimos decifrar a mensagem com a chave privada, que está na posse do receptor (ver Figura 2.2). Neste tipo de criptografia, conseguimos garantir não só a autenticidade do receptor da mensagem (só o "verdadeiro"destinatário da mensagem é capaz de ler a mensagem original), mas também a confidencialidade da mensagem (qualquer pessoa pode ler a mensagem cifrada, mas não entende o que está escrito, a não ser o próprio destinatário da mensagem, após utilização da chave privada), características estas cada vez mais exigidas num gigantesco mundo de trocas de informação de carácter confidencial (por exemplo, número de cartões de crédito, palavras-chave, etc).

A autenticidade do emissor da mensagem pode ser garantida através, por exemplo, da assinatura digital. Desta forma, garantimos que a mensagem provém do emissor correcto e que a mesma não foi alterada desde que foi emitida até à sua recepção.

Não existem processos de cifragem / decifragem totalmente eficazes. Em teoria, qualquer chave pode ser quebrada pela força bruta: supondo que o atacante possui uma

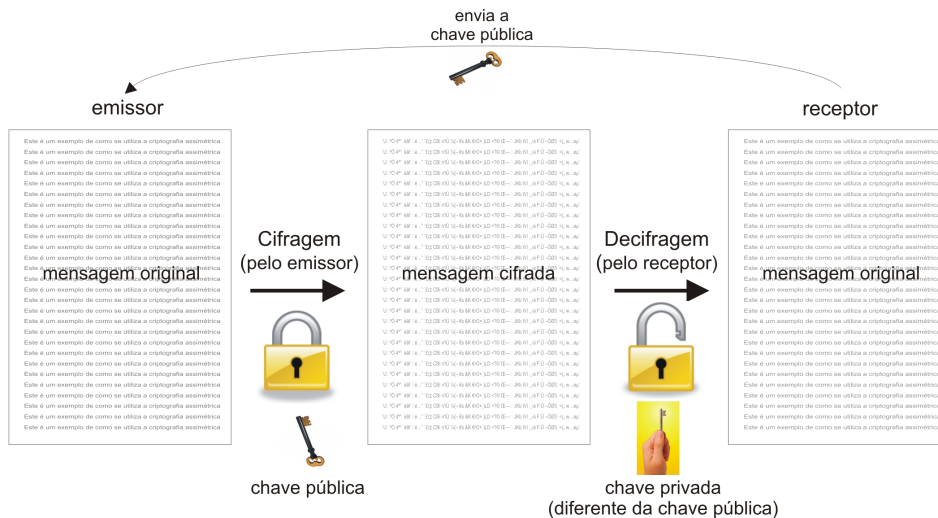


Figura 2.2: Na criptografia assimétrica, a chave para cifrar (chave pública) é distinta da chave para decifrar (chave privada).

cópia da mensagem cifrada, e conhece o tipo de criptografia que foi utilizada, basta tentar decifrar, utilizando todas as chaves possíveis, até acertar na chave correcta. Para evitar isto, os utilizadores da criptografia deverão recorrer às capacidades de processamento dos computadores actuais de modo a usar algoritmos e chaves que não possam ser descobertos em tempo útil. O tempo necessário para quebrar uma chave pela força bruta depende do número de chaves possiveis (número de bits da chave) e do tempo de execução do algoritmo.

O grande problema desta abordagem é que a capacidade de processamento dos equipamentos tem aumentado de forma impressionante, pelo que há necessidade de, de tempos a tempos, aumentar o tamanho das chaves.

Além de resolver definitivamente o problema da distribuição de chaves, a criptografia de chave pública facilita a implementação de mecanismos de autenticação de mensagens e assinatura digital. Embora tenham sido propostos outros algoritmos, actualmente o RSA é o mais sólido. O algoritmo "Merkle-Hellman Knapsack" demorou quatro anos a ser quebrado por Adi Shamir. Uma segunda versão, pensava-se mais sólida, demorou dois anos a ser quebrada. Actualmente praticamente pode-se afirmar que o RSA é o algoritmo de chave pública. Este algoritmo deve-se a Ron Rivest, Adi Shamir e Len Adleman (RSA).

2.2.1 O sistema criptográfico RSA

Em 1978, Rivest, Shamir e Adleman [29] procuraram garantir que as comunicações feitas através do correio electrónico mantinham as características do correio usual: as mensagens serem secretas e poderem ser assinadas. Estas garantias podem ser dadas através de um novo sistema de (de)cifragem, o RSA, que se pode descrever da seguinte forma:

- Escolhem-se dois números primos grandes, p e q, e calcula-se $n := p \cdot q$. Apesar de n ser público, p e q deverão ser secretos;
- Determina-se e, que deverá ser primo com $(p-1)(q-1)$. Por outras palavras, $mdc(e, (p-1) \cdot (q-1)) = 1$. O valor de e deverá ser publicado, uma vez que servirá para cifrar a mensagem.
- Determina-se d, que será o inverso multiplicativo de $e \mod \phi(n)$. Obtemos, assim, $ed \equiv 1 \mod \phi(n)$. Uma vez que d servirá para decifrar mensagens, este valor deverá ser mantido secreto, à semelhança de $\phi(n)$.

O algoritmo RSA é sustentado pelo Teorema de Euler (1.6) e pelo Pequeno Teorema de Fermat (1.7), já enunciados.

Teorema 1.6 (Teorema de Euler)**.** *Se $mdc(a, n) = 1$, então*

$$a^{\phi(n)} \equiv 1 \mod n.$$

Teorema 1.7 (Pequeno Teorema de Fermat)**.** *Se p é primo, então*

$$a^p \equiv a \mod p,$$

para qualquer inteiro a.

A mensagem a cifrar deve ser um número M, inferior a n. Se n for um número muito grande e, como, por construção, n é o produto de dois números primos inferiores a n, então n é primo com quase todos os números até n.

A mensagem cifrada, C, é tal que $C \equiv M^e \mod n$. Para descobrirmos d tal que $ed \equiv 1 \mod \phi(n)$, basta utilizarmos o algoritmo de Euclides para determinar os inteiros u e d tais que $ed - u\phi(n) = 1$.

Se $mdc(M, n) = 1$,

$$\begin{aligned} C^d &\equiv (M^e)^d \mod n \\ &\equiv M^{1+u\phi(n)} \mod n \\ &\equiv M \cdot (M^{\phi(n)})^u \mod n \\ &\equiv M \cdot 1^u \mod n, \text{ pelo Teorema de Euler (1.6)} \\ &\equiv M \mod n. \end{aligned}$$

Suponhamos, agora, que $mdc(M, n) \neq 1$. Suponhamos, sem perda de generalidade, que p é um factor próprio de n.

Se $p \mid M$, então

$$M^{ed} \equiv 0 \mod p \equiv M \mod p.$$

Se $p \not| M$, então, pelo pequeno teorema de Fermat (teorema 1.7),

$$M^{ed} = M \cdot (M^{p-1})^{u(q-1)} \equiv M \mod p.$$

Sabendo que n tem dois factores próprios e dada a generalidade de p, temos que, em qualquer dos casos,

$$M^{ed} \equiv M \mod p$$

e

$$M^{ed} \equiv M \mod q,$$

donde

$$M^{ed} \equiv M \mod n.$$

Seja M a mensagem original, a enviar por Alice. Vejamos o que acontece quando $M > n$.

Exemplo. *Consideremos um alfabeto de* 27 *símbolos, onde* $A \mapsto 01$, $B \mapsto 02$, $C \mapsto 03$, $\cdots$, $Y \mapsto 25$, $Z \mapsto 26$ *e espaço em branco* $\mapsto 00$.

Imaginemos que Alice quer enviar a Bob a mensagem SEI CIFRAR. *Em código, a mensagem* (M) *fica*

$$\begin{aligned} &(19\,05\,09\,00\,03\,09\,06\,18\,01\,18)_{27} \\ =&19\cdot 27^9+05\cdot 27^8+09\cdot 27^7+03\cdot 27^5+09\cdot 27^4+06\cdot 27^3+18\cdot 27^2+01\cdot 27+18 \\ =&146392691036940. \end{aligned}$$

Suponhamos que a Alice vai cifrar uma mensagem, com recurso ao algoritmo RSA, com $n = 24083417$, $p = 7477$, $q = 3221$ *e* $e = 168481$.

Repare-se que M é maior que n, pelo que há necessidade de dividir M em blocos, cujo tamanho seja inferior a M, de modo a garantir a unicidade da cifragem/decifragem. Confirmemos que o facto de M ser superior a n não garante a que o RSA devolva a mensagem original:

$$\begin{aligned} C &\equiv M^e \mod n \\ &\equiv 146392691036940^{168481} \mod 24083417 \\ &\equiv 11405364 \mod 24083417 \end{aligned}$$

Então, a mensagem cifrada é $C = (11405364)_{10} = (21\,12\,12\,05\,24)_{27}$. Alice envia a Bob a mensagem ULLEX.

Neste momento, propomo-nos a descobrir d (já que Bob vai precisar dele para decifrar a mensagem), de tal forma que

$$d \cdot e \equiv 1 \mod \phi(n),$$

i.e.,

$$d \cdot e - kn = 1, \text{ para algum inteiro } k,$$

ou ainda,

$$d \cdot e - kn = mdc(e, n).$$

$\phi(n) = (p-1) \cdot (q-1) = 7476 \cdot 3220 = 24072720.$

Utilizando o algoritmo de Euclides, temos que:

	i	1	2	3	4	5	6
	q_i			142	1	7	2
	r_i	24072720	168481	148418	20063	7977	4109
24072720	x_i	1	0	1	-1	8	-17
168481	y_i	0	1	-142	143	-1143	2429

(cont.)

	i	7	8	9	10
	q_i	1	1	16	20
	r_i	3868	241	12	1
24072720	x_i	25	-42	697	-13982
168481	y_i	-3572	6001	-99588	1997761

Para $i \geq 3$*,*

$$x_i = x_{i-2} - x_{i-1} \times q_i$$

e

$$y_i = y_{i-2} - y_{i-1} \times q_i.$$

Chegamos à expressão $1 = 1997761 \cdot 168481 - 13982 \cdot 24072720$*. Logo,* $d = 1997761$ *(note-se que* $mdc\,(d, \phi(n) = 1)$*).*

Bob pega na mensagem ULLEX*, transforma-a em código*

$$((21\,12\,12\,05\,24)_{27}),$$

transforma-a num número decimal

$$(21 \cdot 27^4 + 12 \cdot 27^3 + 12 \cdot 27^2 + 05 \cdot 27 + 24 = 11405364),$$

eleva este número a d, módulo n

$$(11405364^{1997761} \equiv 3130084 \mod 24083417),$$

escreve o resultado na base 27 $((3130084)_{10} = (05\,24\,00\,18\,01)_{27})$ *e obtém a mensagem* EX RA*, que não coincide com a mensagem original e não tem qualquer significado.*

Note-se que SEI CIFRAR *e* EX RA *cifrados tornam-se* ULLEX*.*

Mostrámos, com um simples exemplo, que M nunca poderá ser superior a n, sob pena de não garantirmos a unicidade do processo de cifragem/decifragem.

Como determinar o tamanho dos blocos de M?

Quando às mensagens originais correspondem números grandes, superiores a n, há necessidade, de as dividirmos em blocos, de forma que o tamanho de cada bloco seja inferior a n e que todos os blocos tenham o mesmo tamanho (em inglês, este processo denomina-se *message blocking*). Mas... que tamanho? Segundo Mollin [24], depende do tamanho do alfabeto da mensagem original (chamemos-lhe N). Se quisermos enviar uma mensagem simples, só com as letras do alfabeto (só maiúsculas ou só minúsculas) e o espaço em branco, $N = 27$; se usarmos o código *American Standard Code for Information Interchange*, vulgo ASCII[1], $N = 256$. Sabendo N, os blocos deverão ter um tamanho l (que é único) tal que $N^l < n < N^{l+1}$. No último bloco, poderá haver

[1] Em anexo encontra-se a tabela ASCII

necessidade de acrescentar cópias do símbolo neutro, à direita, por forma a ficar com o tamanho l (note-se que a divisão é feita sobre a mensagem original e não sobre a mensagem numérica). Depois, cifra-se cada um dos blocos separadamente. Posteriormente, cada um dos blocos cifrados poderá ser escrito como blocos de comprimento $l+1$ (acrescentando, se necessário, zeros à esquerda do bloco).

Exemplo. *Utilizando o exemplo anterior, em que Alice pretende enviar a Bob, com recurso ao RSA, a mensagem* SEI CIFRAR*, utilizando um alfabeto de 27 símbolos, com* $n = 24083417$*,* $e = 168481$ *e* $d = 1997761$*.*

Já vimos que $M = 146392691036940$ *e que* $M > n$*.*

Como $27^5 < n < 17^6$*,* $l = 5$*, pelo que a mensagem original irá ser dividida em duas submensagens* SEI C *e* IFRAR*, cada uma delas constituída por 5 símbolos de alfabeto (se o último bloco tivesse um tamanho inferior a 5, acrescentaríamos espaços em branco suficientes à direita, por forma a que o tamanho ficasse igual a 5).*

$M = M_1 M_2$*, onde*

$$M_1 = (19\,05\,09\,00\,03)_{27} = 10202358$$

e

$$M_2 = (09\,06\,18\,01\,18)_{27} = 4914234.$$

Repare-se que cada um dos blocos é inferior a n*.*

$$\begin{aligned} M_1^e &= 10202358^{168481} \mod 24083417 \\ &= 22090061 \mod 24083417 \\ &= (01\,14\,15\,07\,23\,11)_{27} \mod 24083417 \end{aligned}$$

$$\begin{aligned} M_2^e &= 4914234^{168481} \mod 24083417 \\ &= 20946687 \mod 24083417 \\ &= (01\,12\,11\,05\,12\,06)_{27} \mod 24083417 \end{aligned}$$

Alice envia a mensagem ANOGWKALKELF *a Bob.*

Bob transforma a mensagem recebida em blocos de comprimento 6 *(*ANOGWK *e* ALKELF*), tansforma cada um destes blocos em números na base 27 (*$(011415072311)_{27}$ *e* $(01\,12\,11\,05\,12\,06)_{27}$*, respectivamente), em números na base 10 (*22090061 *e* 20946687*, respectivamente), eleva estes números a* d*, módulo* n

$$(22090061^{1997761} \equiv 10202358 \mod 24083417$$

e

$$20946687^{1997761} \equiv 4914234 \mod 24083417,$$

respectivamente), escreve os resultados obtidos na base 27, obtendo

$$(19\,05\,09\,00\,03)_{27} \;\; e \;\; (09\,06\,18\,01\,18)_{27},$$

que corresponde a SEI C *e* IFRAR. *Bob chega, assim, à mensagem original, enviada por Alice,* SEI CIFRAR.

Vejamos um exemplo no Maple, sobre como cifrar e decifrar o título desta dissertação, com $n = 1227028877$, $e = 57157$ e $d = 138564193$, com recurso à tabela ASCII:

Exemplo. *Utilizando o algoritmo do Apêndice A, com o seguinte* input

```
p = 17911;
q = 68507;
e = 57157;
msg = "Ataques ao sistema criptográfico RSA",
```

este algoritmo produz o seguinte output [2]*:*

[2] Apesar de no *output* aparecerem várias cópias do símbolo □, o computador trabalha, em memória, com o código binário. O mesmo símbolo pode, de facto, corresponder a códigos binários / decimais distintos.

```
Introduza um número inteiro para 'p', seguido do símbolo ';': 17911;
```

$$p := 17911$$

```
Introduza um número inteiro para 'q', seguido do símbolo ';': 68507;
```

$$q := 68507$$

$$n := 1227028877$$

$$fiDeN := 1226942460$$

```
Introduza um número inteiro para 'e', seguido do símbolo ';': 57157;
```

$$e := 57157$$

$$d := 138564193$$

```
Introduza a mensagem a cifrar, entre aspas, seguido de ';': "Ataques ao
                                                            sistema
                                                            criptográfico
                                                            RSA";
```

msg := "Ataques ao sistema criptográfico RSA"

true

36

tamanhoAlf := 256

```
Cada bloco é constituído por 3 símbolos.

Mensagem cifrada: 943823949 55474197 816703296 715709051 303308710
589937602 195257930 309084719 57059009 974124635 1187028409 491389203

A Alice envia a seguinte mensagem (cifrada) a Bob:
```

*8AæM □Nx□ 0-ç@ *¨Ú(□□□¦ #)»Â □£fJ □lB/ □f¦Á :□ö[FÀ□' □J□□*

```
Mensagem decifrada: 4289633 7435621 7544929 7282803 6910836 6647137
2122610 6910068 7300978 14771817 6516512 5395265

Bob lê a seguinte mensagem (decifrada):
```

Ataques ao sistema criptográfico RSA

Capítulo 3

Complexidade

Matematicamente, quase todas as cifras são quebráveis [1]. A questão é saber quanto tempo demora a quebrar uma cifra que matematicamente sabemos ser quebrável. Quanto mais tempo demorar a quebrar a cifra, melhor ela é.

3.1 Introdução

Considera-se uma *operação-bit* como sendo cada operação elementar feita entre dois *bits*. É de extrema importância, em criptografia, estimar o número de *operações-bit* necessárias para efectuar operações aritméticas ou outros cálculos num computador. Comecemos com algumas noções necessárias.

Podemos escrever um número inteiro não negativo n numa base b, na forma

$$(d_{k-1}...d_2d_1d_0)_b,$$

onde $0 \leq d_i < b$, $\forall i \in \{0, 1, 2, \ldots, k-1\}$ e $d_{k-1} \neq 0$. Isto significa que, na base decimal,

$$n = d_{k-1}b^{k-1} + \cdots + d_2b^2 + d_1b^1 + d_0b^0.$$

Esta representação é única na base b. Dizemos que n tem um comprimento k.

Se utilizarmos o sistema decimal, em que a base é 10, temos que

$$6352 = 6 \times 10^3 + 3 \times 10^2 + 5 \times 10 + 2 \times 1,$$

sendo o comprimento deste número igual a 4. Por uma questão de comodidade, escrevemos 6352 em vez de $(6352)_{10}$. Se usarmos o mesmo número, no sistema binário (base

[1] Existem cifras que matematicamente são inquebráveis (criptografia quântica)

2), obtemos o número $(1100011010000)_2$. Neste caso, o número de bits, no sistema binário, é 13.

No caso geral, para $n > 1$,

$$\text{número de } bits = [\log_b n] + 1 = \left[\frac{\log n}{\log b}\right] + 1,$$

onde $[r]$ representa a função que devolve o maior inteiro menor ou igual a r, sendo r um número real.

De referir que, neste trabalho, $\log n$ representa $log_e n$.

Estamos agora em condições de analisar o tempo necessário para efectuar operações aritméticas.

Adição de números

Como os computadores trabalham no sistema binário, iremos, ao longo deste capítulo, focar a nossa atenção neste mesmo sistema. Considere-se os números 1100111 (7 bits) e 11101 (5 bits).

```
   1111111   --> transporte
   01100111
 + 00011101
 ----------
   10000100
```

Ao somarmos, bit a bit, pode acontecer uma das seguintes situações [2]:

- se ambos os bits são iguais a zero e não houver transporte, coloca-se 0 e passa-se à coluna seguinte;
- se ambos os bits são iguais a zero e houver transporte, coloca-se 1 e passa-se à coluna seguinte;
- se um dos bits é igual a zero e o outro é igual a um e não houver transporte, coloca-se 1 e passa-se à coluna seguinte;
- se um dos bits é igual a zero e o outro é igual a um e houver transporte, coloca-se 0 e passa-se à coluna seguinte, com transporte de 1;

[2] Temos, por vezes, que acrescentar zeros à esquerda a um ou a ambos os números, de modo a ambos terem o mesmo número de bits e a podermos estar sempre numa destas situações.

- se ambos os bits são iguais a um e não há transporte, coloca-se 0 e passa-se à coluna seguinte, com transporte de 1;

- se ambos os bits são iguais a um e há transporte, coloca-se 1 e passa-se à coluna seguinte, com transporte de 1.

A cada uma destas operações descritas chamamos *operação-bit*.

Se somarmos um número com k bits com um número com p bits (sendo $p \leq k$), iremos efectuar, no máximo, $k + 1$ *operações-bit* e a soma terá, no máximo, $k + 1$ bits.

Apesar de conhecermos o número de *operações-bit*, o tempo total despendido depende de cada computador[3]. Quando falamos em estimar o tempo que se demora a efectuar uma determinada tarefa, falamos em número de *operações-bit*.

Subtracção de números

Para efectuar a subtracção, os computadores recorrem-se a um artifício, utilizando o conhecimento de que subtrair é o mesmo que adicionar pelo simétrico. A este artifício chamamos *Método do Complemento para Dois*. Na base binária, o complemento para dois de um número consiste na diferença entre a potência de expoente seguinte à potência mais significativa do número pretendido e esse mesmo número. Os computadores, na realidade, não trabalham com os sinais operacionais $+$ e $-$, mas acrescentam um dígito à esquerda do número binário que irá representar o sinal do número (0 para $+$ e 1 para $-$).

Para calcular $1101110 - 10111$, o computador completa o subtractivo de modo a ficar com o mesmo número de bits do aditivo ($1101110 - 0010111$), troca os zeros por uns e vice-versa ao subtractivo (0010111 fica 1101000), soma 1 ao número invertido (no caso apresentado, fica 1101001), soma o aditivo a este número ($1101110 + 1101001$) e ignora, nesta soma, o último transporte.

[3]Sabemos que periodicamente os fabricantes lançam computadores mais evoluídos e mais rápidos. Num computador existem vários programas abertos simultaneamente, consumindo velocidade do computador em causa. A maneira mais simples de medir esta velocidade é um número chamado "Gigahertz"(abrevia-se Ghz). Um computador de 3 Ghz é capaz de efectuar, no máximo, 3×10^9 *operações-bit* por segundo.

```
  1 1  1
   01101110
+  01101001
   11010111
```

Ignorando o último transporte, ficamos com 1010111, que é o resultado de

$$1101110 - 10111.$$

À luz deste exemplo, vemos que se subtraírmos um número com k bits com um número com p bits (sendo $p \leq k$), ignorando o tempo que se demora a efectuar as trocas dos bits do subtractivo, iremos efectuar, no máximo, $k + 1$ operações-bit.

Esta ideia assenta no facto de que, sendo M um número binário com um mínimo de k bits, e N um número binário com k bits,

$$M - N = M + ((2^{k+1})_{10} - N) - (2^{k+1})_{10}.$$

Se quisermos subtrair dois números na base binária, procedemos de forma semelhante à subtracção na base decimal, com as seguintes regras:

- $1 - 0 = 0$;
- $0 - 0 = 0$;
- $1 - 1 = 0$;
- $0 - 1 = 1$ e pede-se 1 emprestado ao bit mais significativo seguinte do aditivo;

No exemplo anteriormente apresentado, repare-se que 10111 tem 5 bits, pelo que iremos calcular, em primeiro lugar, $(2^6)_{10} - 10111 = 100000 - 10111)$:

```
   ****
  100000
-  10111
    1001
```

Somando 1101110 com 1001:

```
     *
 1101110
+   1001
--------
 1110111
```

Resta apenas fazer $1110111 - 1000000 = 1010111$.

Multiplicação de números

Vamos estudar o tempo que se demora a multiplicar dois números, na base binária. Multipliquemos 11011101 por 10011.

```
       11011101
×         10011
---------------
   000011011101
   000110111010
   000000000000
   000000000000
+  110111010000
---------------
  1000001100111
```

Quando multiplicamos 1 pelo primeiro factor, na prática fazemos uma cópia do número inicial; quando multiplicamos 0 pelo número inicial, colocamos uma linha de zeros, com tantos bits quantos o primeiro factor, à esquerda desse zero. No final de cada uma destas multiplicações, acrescenta-se um zero à direita, se tivermos que efectuar outra multiplicação e antes de a fazermos, de forma semelhante ao que se faz na multiplicação usual na base decimal.

Se multiplicarmos um número com k *bits* com um número com p *bits*, na pior das hipóteses, fazemos p cópias do primeiro factor, isto é, iremos ter sempre p linhas, cada uma delas com um máximo de $k + p$ bits (a primeira parcela tem, no máximo, k-*bits*).

No final, temos, na pior das hipóteses, p adições entre dois números e a soma terá, no máximo, $k+p$ bits.

Podemos concluir que

$$\text{tempo}(k\text{-bit} \times p\text{-bit}) \leq p(k+p) \leq k(k+k) = 2k^2 \text{ operações-bit},$$

com $p \leq k$. Observe-se que negligenciámos o tempo que se demora a fazer uma cópia do primeiro factor, assim como o tempo que se demora a deslocar um número para a esquerda ou a aceder à memória do computador. O tempo que se demora a fazer estas operações é tão curto, comparado com o tempo que se demora a fazer uma operação elementar, que podemos negligenciá-lo.

Fazendo $k = [\log_2 n] + 1 \leq \dfrac{\log n}{\log 2}$, obtemos um majorante para o tempo que se demora a multiplicar um k-bit com um p-bit, em função de n.

3.2 Estimativas de tempo

Definição. (ordem) Sejam f e g duas funções aritméticas, cujo conjunto de partida é $\mathbb{Z}^+$ e o conjunto de chegada é $\mathbb{R}$ (ou mesmo $\mathbb{C}$). Escrevemos

$$f(n) = O(g(n)),$$

se existir $C > 0$ e $n_0 \in \mathbb{Z}$ tal que, para todo $n \geq n_0$,

$$|f(n)| \leq C \cdot g(n).$$

Mais genericamente, dizemos que

$$f(n_1, n_2, \cdots, n_k) = O(g(n_1, n_2, \cdots, n_k)),$$

se existir $C > 0$ tal que

$$|f(n_1, n_2, \cdots, n_k)| = Cg(n_1, n_2, \cdots, n_k).$$

Exemplo. *Se* $f(n) =$ *número de dígitos de* $n = k$*, então*

$$f(n) = O\left(\frac{\log n}{\log 2}\right) = O(\log n),$$

isto é, existe uma constante $C > 0$ *e* $n_0 \in \mathbb{Z}$ *tal que, para todo* $n \geq n_0$, $|f(n)| \leq C \cdot \log n$.

Exemplo. *tempo* $(n \cdot m) = O(\log n \cdot \log m)$ *operações bit.*

Exemplo. *Sendo a divisão a operação contrária da multiplicação,*

$$\textit{tempo}\ (n \div m) = O(\log n \cdot \log m)\ \textit{operações bit.}$$

Exemplo. *Para determinarmos o tempo que se demora a fazer a raiz quadrada de* m*, no sistema binário, suponhamos que* m *tem* $k+1$ *bits. Para determinarmos uma boa aproximação de* $\sqrt{m}$*, que é* $[\sqrt{m}]$*, escrevemos 1 seguido de* $\left[\frac{k-1}{2}\right]$ *zeros. Para encontrarmos os bits de* $[\sqrt{m}]$*, procedemos do seguinte modo:*

- *trocamos o primeiro bit igual a 0 por 1. Seguidamente, calculamos o quadrado do número obtido. Se o número obtido for inferior a* m*, então a raiz quadrada de* m *é esse número. Caso contrário, "destrocamos"o* 1 *por* 0 *e prosseguimos com o algoritmo.*
- *trocamos o bit seguinte, que é o segundo* 0*, para* 1 *e elevamos este número ao quadrado. Se este for inferior a* m*, encontrámos a raiz quadrada de* m*, caso contrário "destrocamos"o* 1 *por* 0*.*

E assim sucessivamente.

Obtemos $[\sqrt{m}]$ *quando, ao elevarmos ao quadrado, o número for inferior a* m*. Isto requer, no máximo,* $[k/2]$ *multiplicações, cada uma delas entre dois números com* $[k/2]+1$ *algarismos. Então,*

$$\textit{tempo}([\sqrt{m}]) = O\left(\log m \cdot \log^2 m\right) = O(\log^3 m).$$

Potenciação (n^n e b^n)

Sabemos que se n for um número binário tem

$$[\log_2 n] + 1 = \left[\frac{\log n}{\log 2}\right] + 1 \ \text{ bits.}$$

n^2 tem, no máximo, o dobro do número de *bits* de n. Sabemos, ainda, que

$$\text{tempo(n} \cdot \text{n)} = O\left(2 \cdot \log^2 n\right).$$

Por outro lado,

$$\text{tempo}((\text{n} \cdot \text{n}) \cdot \text{n}) = O\left(2 \cdot \log^2 n\right) + O\left(2 \cdot (2 \log n) \cdot \log n\right).$$

Usando recursivamente este raciocínio, obtemos

$$\begin{aligned}
tempo(((n \cdot n) \cdot n) \cdot \cdots) \cdot n &= O\left(\sum_{k=1}^{n-1} \left(k \cdot 2 \cdot \log^2 n\right)\right) \\
&= O\left(2 \cdot \log^2 n \cdot \sum_{k=1}^{n-1} k\right) \\
&= O\left(2 \cdot \log^2 n \cdot \frac{(1+n-1)(n-1)}{2}\right) \\
&= O\left(n^2 \log^2 n - n \log^2 n\right) \\
&= O\left(n^2 \log^2 n\right)
\end{aligned}$$

Concluindo,

$$tempo(b^n) = O\left(n^2 \log^2 b\right).$$

Potenciação modular ($b^n \mod m$)

Ao longo deste trabalho vamos ter que efectuar cálculos do tipo $b^n \bmod m$ (isto é, encontrar o menor resíduo não negativo). Ao invés de calcularmos primeiro b^n e depois determinarmos o resíduo de $b^n \bmod m$, iremos calcular de uma forma mais rápida.

Designemos o produto parcial por a. Quando tivermos percorrido o algoritmo, a designará o menor resíduo não negativo de $b^n \bmod m$.

Começamos com $a = 1$. Sejam $n_{k-1}, \cdots, n_1, n_0$ os dígitos binários de n, ou seja,

$$n = n_0 + 2n_1 + 2^2 n_2 + \cdots + 2^{k-1} n_{k-1},$$

sendo k o número de bits de n. Cada n_j, com $0 \leq j \leq k-1$, é igual a 0 ou 1.

- Se $n_0 = 0$, então $a := 1$. Se $n_0 := 1$, então $a := b$.
- Calcula-se $b_1 := b^2 \bmod m$. Se $n_1 = 0$, então a permanece com o mesmo valor. Se $n_1 = 1$, então $a := a \times b_1 \bmod m$ (obtemos, assim, $b^3 \bmod m$).
- Calcula-se $b_2 := b_1^2 \bmod m$. Se $n_2 = 0$, então a permanece com o mesmo valor. Se $n_2 = 1$, então $a := a \times b_2 \bmod m$.

- ... e assim sucessivamente. Ao fim de $n-1$ passos obtemos $a \equiv b^n \mod m$.

Exemplo. *Vamos calcular* $11^{25} \mod 17$. *Isto equivale a dizer que queremos calcular* $11^{(11001)_2} \mod 17$.

Como $n_0 = 1$, *então* $a := b$, *isto é,* $a := 11$.

$$\begin{aligned} b_1 :&= b^2 \mod m \\ &\equiv 11 \times 11 \mod 17 \\ &\equiv 121 \mod 17 \\ &\equiv 2 \mod 17 \end{aligned}$$

Como $n_1 = 0$, *então* $a = 11$.

$$\begin{aligned} b_2 :&= b_1 \times b_1 \mod m \\ &\equiv 2 \times 2 \mod 17 \\ &\equiv 4 \mod 17 \end{aligned}$$

Como $n_2 = 0$, *então* $a = 11$.

$$\begin{aligned} b_3 :&= b_2 \times b_2 \mod m \\ &\equiv 4 \times 4 \mod 17 \\ &\equiv 16 \mod 17 \end{aligned}$$

Como $n_3 = 1$, *então*

$$\begin{aligned} a :&= a \times b_3 \\ &\equiv 11 \times 16 \mod 17 \\ &\equiv 176 \mod 17 \\ &\equiv 6 \mod 17. \\ b_4 :&\equiv b_3 \times b_3 \mod m \\ &\equiv 16 \times 16 \mod 17 \\ &\equiv 256 \mod 17 \\ &\equiv 1 \mod 17 \end{aligned}$$

Como $n_4 = 1$, *então*

$$\begin{aligned} a :&= a \times b_4 \\ &\equiv 6 \times 1 \mod 17 \\ &\equiv 6 \mod 17. \end{aligned}$$

Logo, $11^{25} \equiv 6 \mod 17$.

Repare-se que, em cada passo, fez-se, no máximo, duas multiplicações e duas divisões (as divisões são necessárias para calcular o módulo m). Cada multiplicação é feita entre dois números, cada um deles com um máximo de l bits, sendo l o número de bits de m. Cada divisão é feita entre dois números, um deles com um máximo de $2l$ bits e o outro com l bits. Assim, cada produto demora, no máximo, $O(\log^2 m)$ e cada divisão demora, no máximo, $O(\log(m^2)\log m) = O(\log^2 m)$. Em todo o algoritmo, são feitos, no máximo, $O(\log n)$ passos. Assim,

$$tempo(b^n \mod m) = O(\log n \log^2 m).$$

Algoritmo de Euclides

Teorema 3.1. *Suponhamos que $a > b$. Então,*

$$tempo\,(determinar\ mdc(a,b)\ usando\ o\ algoritmo\ de\ Euclides) = O\left(\log^3 a\right).$$

Demonstração: O algoritmo de Euclides consiste em fazer divisões sucessivas, onde o resto vai decrescendo sucessivamente, conforme já foi referido no teorema 1.1.

Para estimarmos quantas operações-bit são necessárias no algoritmo de Euclides, é necessário sabermos quantas divisões precisamos de efectuar e quantas operações-bit tem cada divisão.

Vamos provar que

$$r_{i+2} < \frac{1}{2} r_i,$$

ou seja, que a sucessão composta pelos sucessivos restos obtidos pelo algoritmo de Euclides, não só decrescem, mas decrescem de uma forma relativamente rápida.

Se $r_{i+1} \leq \frac{1}{2} r_i$, sendo

$$0 = r_{k+1} < r_k < \cdots < r_i < \cdots < r_2 < r_1,$$

então $r_{i+2} < r_{i+1} \leq \frac{1}{2} r_i$.

Se $r_{i+1} > \frac{1}{2} r_i$, então

$$\frac{1}{2} r_i < r_{i+1} < r_i.$$

Logo, na divisão seguinte do algoritmo de Euclides obtemos $r_i = 1 \cdot r_{i+1} + r_{i+2}$. Daqui concluimos que

$$r_{i+2} = r_i - r_{i+1} = \left(\frac{1}{2}r_i - r_{i+1}\right) + \frac{1}{2}r_i < \frac{1}{2}r_i.$$

Acabámos de provar que $r_{i+2} < \frac{1}{2}r_i$. Como em cada dois passos do algoritmo de Euclides, o resto decresce, no mínimo, para metade, e como o resto nunca é inferior a 1, temos, no máximo, $2[\log_2 a]$ divisões.

Sabemos que $2[\log_2 a] = O(\log a)$.

Cada divisão envolve números com um máximo de bits igual ao número de bits de a. Sabemos, ainda, que o número de operações-bit em cada divisão é $O(\log^2 a)$.

Logo, o algoritmo de Euclides demora $O(\log^3 a)$ operações-bit. $\square$

Utilizando a segunda parte do teorema 1.1 e o teorema 3.1, obtemos o seguinte resultado:

Teorema 3.2. *Se $mdc(a, n) = 1$, com $a < n$, o inverso de $a \mod n$ pode ser encontrado em $O(\log^3 n)$ operações-bit.*

3.2.1 RSA

Quando falámos do RSA (secção 2.2.1), referimos que o par de números (n, e) é do conhecimento público e que o par $(\phi(n), d)$ é do conhecimento privado, não podendo ser divulgado.

De facto, conhecer p e q é equivalente a conhecer $\phi(n)$.

Relembremos que n é o produto de dois números primos distintos p e q, que

$$\phi(n) = (p - 1) \cdot (q - 1) = n - (p + q) + 1, \tag{3.1}$$

que

$$mdc(d, \phi(n)) = 1$$

e que

$$ed \equiv 1 \mod \phi(n).$$

Como n é ímpar, se formos conhecedores de p e de q, conseguimos descobrir $\phi(n)$ através de (3.1). Para conhecermos $\phi(n)$, necessitamos apenas de duas adições e de uma subtracção, ou seja, descobrimos $\phi(n)$ em $O(\log n)$ operações-bit.

Suponhamos, agora, que $\phi(n)$ e n são conhecidos e que n é o produto de dois primos ímpares distintos desconhecidos.

Como $pq = n$ e $p + q = (n+1) - \phi(n)$, então p e q são soluções da equação do segundo grau

$$x^2 - (n + 1 - \phi(n))x + n = 0. \tag{3.2}$$

Claramente que $p + q$ é par.

Designemos $p + q = n + 1 - \phi(n)$ por $2b$.

A equação (3.2) é equivalente a

$$x = b \pm \sqrt{b^2 - n},$$

e p e q podem ser descobertos através das soluções desta equação. Portanto, $\phi(n)$ não deve ser tornado público.

Além de somas, subtracções, multiplicações e divisões, temos que efectuar uma raiz quadrada. No que diz respeito ao tempo gasto, a raiz quadrada é a que demora mais tempo.

Logo, p e q podem ser descobertos, a partir de $\phi(n)$ e de n em $O(\log^3 n)$ operações-bit.

Capítulo 4

Primalidade

Os testes de primalidade servem para averiguar se um dado número inteiro é primo ou não. Numa área como o RSA, onde se necessita de recorrer a números primos "muito grandes", estes testes são de vital importância. É necessário saber se o p e o q escolhidos são primos.

É evidente que podemos sempre recorrer ao algoritmo da divisão, ou seja, dado n ímpar, fazer $n \div m$, com $m \in \left\{2,\ 3,\ 4,\ ...,\ [\sqrt{n}]\right\}$, até encontrar um valor para m, tal que $m \mid n$. Este processo demora $O\left(\sqrt{n}\log^2 n\right)$ operações-bit. Podemos poupar no número de testes a efectuar se dividirmos n apenas por todos os números ímpares positivos, superiores a 1 e inferiores a $[\sqrt{n}]$, mas demoraríamos, na mesma, $O\left(\sqrt{n}\log^2 n\right)$ operações-bit. Poupamos, ainda mais, no número de testes a efectuar, se verificarmos a primalidade de n dividindo n por todos os números primos até n. Utilizando o Teorema dos Números Primos[1], conseguiríamos reduzir o tempo gasto para $O\left(\sqrt{n}\log n\right)$ operações-bit.

Como os números utilizados no algoritmo do RSA são muito grandes, qualquer um destes processos torna-se extremamente moroso. O objectivo de qualquer atacante é descobrir a chave secreta em tempo útil.

Veja-se um exemplo de algoritmo em Maple da divisão, que testa se n é composto.

Algoritmo 4.1.

```
n := -1:
for i while n <= 2 or type (n, integer) = false do
  n := readstat ("Introduza um número inteiro maior que 2, seguido
        de ';': "):
end do:
contador := 0:
```

[1] Se $x \to \infty$, então $\pi(x) \sim \dfrac{x}{\log x}$, sendo $\pi(x)$ o número de primos inferiores ou iguais a x.

```
for i from 2 while i <= isqrt(n) do
   if ((n mod i) = 0) then
    printf ("O número %d é composto. \n" , n):
    printf("%d = %d * %d", n, i, n/i);
    contador := contador + 1:
    break:
  end if:
  i := nextprime(i)-1:
end do:
if contador = 0 then
  printf ("O número %d é primo.", n):
end if:
```

Dado a morosidade deste teste para números muito grandes, vamos analisar alguns testes de primalidade, cujos tempos de execução são, comparativamente, muito menores.

4.1 Teste de Fermat

O teste de Fermat baseia-se no Pequeno Teorema de Fermat, já anteriormente enunciado e demonstrado.

Teorema 1.7. *(Pequeno Teorema de Fermat) Se p é primo, então*

$$a^p \equiv a \mod p,$$

para qualquer inteiro a.

Vejamos como o teste funciona. Seja n o número que queremos averiguar se é primo e seja a tal que $1 < a < n$. O teste consiste em averiguar se

$$a^n \equiv a \mod n, \tag{4.1}$$

qualquer que seja o valor de a primo com n.

Se existir um valor para a para o qual a condição anterior não se verifica, então n seguramente não é primo. Se a condição (4.1) se verificar, então provavelmente n é primo. Se n é composto e verifica o teste de Fermat para uma base a, dizemos que n é pseudoprimo para a base a.

Na prática, começamos por calcular $2^n \mod n$. Se $2^n \not\equiv 2 \mod n$, concluimos que n não é primo. Se $2^n \equiv 2 \mod n$, calculamos $3^n \mod n$. Se $3^n \not\equiv 3 \mod n$, conclui-se que n não é primo, caso contrário calculamos $5^n \mod n$, e assim sucessivamente.

Infelizmente, se n passar no teste de Fermat para toda a base a inteira positiva, não podemos concluir que n é primo. Podemos estar perante números compostos que passam no teste de Fermat, para qualquer base inteira positiva a - são os chamados *números de Carmichael*.

Exemplo. *Consideremos o primeiro número de Carmichael,* 561*, que é composto (*$561 = 3 \cdot 11 \cdot 17$*).*

Pelo teorema de Euler (teorema 1.6), se

$$mdc(a,\, 3) = mdc(a,\, 11) = mdc(a,\, 17) = 1,$$

então

$$\begin{cases} (a^2)^{280} \equiv 1 \mod 3 \\ (a^{10})^{56} \equiv 1 \mod 11 \\ (a^{16})^{35} \equiv 1 \mod 17 \end{cases}$$

pelo que

$$\begin{cases} a^{561} \equiv a \mod 3 \\ a^{561} \equiv a \mod 11 \\ a^{561} \equiv a \mod 17 \end{cases}$$

Logo, para qualquer base inteira a,

$$a^{561} \equiv a \mod n.$$

Em anexo (apêndice C) encontra-se o algoritmo, no *Maple*, do Teste de Fermat. Neste algoritmo, se $n \leq 10000$, optámos por elaborar o teste de Fermat para todas as bases primas e primas com n, até n, já que existem apenas 1229 números primos até 10000. Se n for superior a este número, e para evitar grandes gastos de tempo, optámos por fazer o teste de Fermat para 50 valores de a, com a aleatório e $1 < a < n$.

Observemos que basta aplicar o teste de Fermat para bases primas e primas com n. Suponhamos que para um determinado n_1, existem bases primas e primas com n_1, digamos a_1 e a_2 tais que

$$a_1^{n-1} \equiv 1 \mod n_1 \quad \text{e} \quad a_2^{n-1} \equiv 1 \mod n_1.$$

Claramente que $a_1 a_2$ não é um número primo.

$$\begin{aligned} (a_1 a_2)^{n-1} \mod n_1 &\equiv a_1^{n-1} a_2^{n-1} \mod n_1 \\ &\equiv 1 \cdot 1 \mod n_1 \\ &\equiv 1 \mod n_1 \end{aligned}$$

Por outro lado, se $a > n$, claramente que a é congruente com um número menor que n, pelo que, para testarmos a primalidade de n basta aplicarmos o teste de Fermat para $1 < a < n$.

Repare-se que, no exemplo anteriormente apresentado, o número 561 não é primo com 3, 11 e 17, precisamente os factores primos da decomposição de 561. Quando encontramos uma base a_1 tal que $1 < a_1 < n$ e

$$mdc(a_1, n_1) = m \neq 1,$$

sendo a_1 é um número primo, então n_1 tem um divisor não trivial, donde concluímos que n_1 não é primo.

Teorema 4.1. *Seja n um número inteiro, ímpar e composto.*

(a) *n é pseudoprimo para a base b, sendo $mdc(n,\ b) = 1$, se e só se a ordem de b em $(\mathbb{Z}/n\mathbb{Z})^*$ (ou seja, o menor expoente positivo tal que b elevado a esse expoente é congruente com 1 módulo n) divide $n - 1$;*

(b) *Se n é pseudoprimo para as bases b_1 e b_2 (sendo $mdc(b_1, n) = 1$ e $mdc(b_2, n) = 1$), então n é pseudoprimo para a base $b_1 b_2$ e é pseudoprimo para a base $b_1 b_2^{-1}$, onde b_2^{-1} designa o inverso multiplicativo de b_2 módulo n;*

(c) *Se n falha (4.1) para uma base $b_k \in (\mathbb{Z}/n\mathbb{Z})^*$, então n falha (4.1) para pelo menos metade das bases possíveis $b \in (\mathbb{Z}/n\mathbb{Z})^*$.*

Demonstração: Como as demonstrações de (a) e de (b) são elementares, deixamo-las a cargo do leitor.

(c) Suponhamos que n falha (4.1) para uma base b_k e suponhamos que n é pseudoprimo para todas as bases em $\{b_1,\ b_2,\ \cdots,\ b_s\}$, com $mdc(b_i,\ n) = 1$, sendo $1 \leq i \leq s$. Por (b), se n fosse pseudoprimo para cada uma das bases $b_k b_i$, com $1 \leq i \leq s$, então n seria pseudoprimo para a base $(b_k b_i) b_i^{-1}$, o que contradiz o facto de n não ser pseudoprimo para a base b_k. Note-se que, para cada $1 \leq i \leq s$, $mdc(b_k b_i,\ n) = 1$. Então, n não é pseudoprimo para as s bases $\{b_k b_1,\ b_k b_2,\ \cdots,\ b_k b_s\}$.

Concluindo, se n é composto, há, no mínimo, tantas bases para as quais n não é pseudoprimo quantas as bases que testemunham a não primalidade de n. □

Vimos, no capítulo 3, que calcular b^{n-1}, módulo n, demora $O\left(\log^3 n\right)$ operações-bit. No máximo, temos $O(\log n)$ testes, pelo que o algoritmo do teste de Fermat demora $O(\log^4 n)$ operações-bit.

4.2 Teste de Miller-Rabin

Como vimos no teste de Fermat, há números que passam no teste e não são primos (números de Carmichael).

O teste probabilístico de Miller-Rabin, desenvolvido por Gary Miller [23] e Michael O. Rabin [27], é um teste probabilístico, que avalia a primalidade de um número, não garantindo, com 100% de fiabilidade, a primalidade de um número. Apenas garante que se um número não passar no teste, este será composto. Como veremos adiante, se um número passar neste teste, ele será provavelmente primo, com um grau de certeza mínimo de 75%. O grau de certeza aumenta se aplicarmos o teste para várias bases.

Antes de explicarmos como funciona o teste de Miller-Rabin, preocupemo-nos com alguns conhecimentos necessários. Qualquer número inteiro n, ímpar e maior ou igual a três pode ser escrito na forma

$$n - 1 = 2^s d,$$

sendo d um número ímpar positivo e s um número inteiro positivo.

Vejamos, agora, dois teoremas necessários para falarmos do teste de Miller Rabin.

Teorema 4.2. *Se p é um número primo e a é um inteiro positivo menor que p, então $a^2 \equiv 1 \mod p$ se e só se $a \equiv 1 \mod p$ ou $a \equiv -1 \mod p$.*

Demonstração: Afirmar que

$$a^2 \equiv 1 \mod p$$

é equivalente a afirmar que

$$p \mid (a^2 - 1),$$

ou seja,

$$p \mid (a-1)(a+1).$$

Pelo teorema 1.2, esta afirmação é equivalente a

$$a \equiv 1 \mod p \bigvee a \equiv -1 \mod p.$$

□

Teorema 4.3. *Seja p um número primo, maior que 2.*

Podemos escrever $p-1=2^s d$, sendo s um número inteiro positivo e d um número inteiro ímpar.

Seja a um inteiro tal que $mdc(a,\, p)=1$.

Então,

$$a^d \equiv 1 \mod p$$

ou

$$a^{2^l d} \equiv -1 \mod p,$$

para algum $0 \leq l \leq s-1$.

Demonstração: Seja p um número primo, maior que 2.

Se a primeira condição se verifica, então fica provado.

Suponhamos que a primeira condição não se verifica. Pelo Teorema de Euler (teorema 1.6),

$$a^{p-1} \equiv 1 \mod p,$$

ou seja,

$$a^{2^s d} \equiv 1 \mod p.$$

Se olharmos para a sequência de números

$$a^d \mod p,\ a^{2d} \mod p,\ a^{2^2 d} \mod p,\ \cdots,\ a^{2^s d} \mod p,$$

sabemos que o último termo da sequência é igual a 1. Verificamos, ainda, que cada termo é o quadrado do termo anterior. Então, acontece uma das seguintes situações:

- todos os termos da sequência acima referida são iguais a 1 e, assim, chegamos à primeira condição;

- há algum termo da sequência que não é igual a 1, mas o seu sucessor, que é seu quadrado módulo p, é igual a 1. Pelo teorema 4.2, o único número que satisfaz estas condições é $-1 \mod p$. Então, a sequência contém um elemento congruente com $-1 \mod p$, verificando-se a segunda condição.

□

Definição (Pseudoprimo forte)**.** Seja n um número composto ímpar e

$$n - 1 = 2^s d,$$

sendo d um inteiro ímpar. Seja, ainda, a um inteiro, primo com n. Se n verifica alguma das condições do teorema 4.3 para a base a, dizemos que n é *pseudoprimo forte* para a base a.

Definição (Testemunha)**.** Seja a um número inteiro. Dizemos que a é testemunha de que n é composto se n não verifica as condições do teorema 4.3 para a base a.

Passemos a descrever o teste de Miller-Rabin:

1. Como n é um inteiro ímpar, podemos escrever $n - 1$ na forma $2^s d$, sendo s um inteiro positivo e d um inteiro ímpar;

2. Escolhemos um número a aleatório, tal que $2 \leq a \leq n - 1$;

3. Verificamos se $a^d \equiv 1 \mod n$ ou se $a^{2^l d} \equiv -1 \mod n$, para algum $0 \leq l \leq s - 1$. Se alguma destas condições se verificar, concluímos n é pseudoprimo forte para a base a; caso contrário, concluímos que n é composto e dizemos que a é testemunha disso.

Em anexo (apêndice D) encontramos o algoritmo, construído em Maple, do teste de Miller Rabin.

No teste de Fermat, vimos que há números que passam no teste e não são primos (números de Carmichael). Provámos isso mesmo com $n = 561$. Vejamos o que acontece ao aplicarmos o teste de Miller Rabin a este número, utilizando o algoritmo apresentado no apêndice D.

Exemplo.
```
> Insira um número inteiro positivo e ímpar, seguido de ';': 561;
561 = 2^4 * 35
2^35 mod 561 = 263
2^(2^0 * 35) mod 561 = 263
2^(2^1 * 35) mod 561 = 166
2^(2^2 * 35) mod 561 = 67
2^(2^3 * 35) mod 561 = 1
561 não é primo e 2 é testemunha disso.
```

Se introduzirmos $n = 967$, que é primo, o algoritmo permite efectuar 10 testes de Miller Rabin antes de concluir que n é primo ou pseudoprimo forte para as 10 primeiras bases primas:

Exemplo.

```
> Insira um número inteiro positivo e ímpar, seguido de ';': 967;
967 - 1 = 2^1 * 483
2^483 mod 967 = 1
3^483 mod 967 = 966
3^(2^0 * 483) mod 967 = 966
5^483 mod 967 = 966
5^(2^0 * 483) mod 967 = 966
7^483 mod 967 = 966
7^(2^0 * 483) mod 967 = 966
11^483 mod 967 = 1
13^483 mod 967 = 966
13^(2^0 * 483) mod 967 = 966
17^483 mod 967 = 1
19^483 mod 967 = 966
19^(2^0 * 483) mod 967 = 966
23^483 mod 967 = 966
23^(2^0 * 483) mod 967 = 966
29^483 mod 967 = 966
29^(2^0 * 483) mod 967 = 966
Provavelmente 967 é primo (com 99.99990463% de probabilidade) ou
pseudoprimo forte para todas as bases primas até 29 (inclusivé).
```

Para que este teste seja eficiente, é necessário que, quando n seja um número composto, haja uma quantidade suficiente de bases que testemunhem esse facto. Na realidade, o teorema que se segue garante que, nestas circunstâncias, existem muitas testemunhas.

Teorema 4.4. *Se n é um número inteiro, composto e ímpar, então n é pseudoprimo forte para uma base b, com $0 < b < n$, para, no máximo, 25% de todas as bases possíveis.*

Como a demonstração deste teorema é extensa, aconselhamos o leitor a consultar Rabin [27] ou Koblitz [18].

À luz do teorema 4.4, se n é um número composto, a probabilidade de escolher uma base b que não testemunhe que n é composto é, no máximo, $\frac{1}{4}$. Então, a probabilidade de escolhermos k bases que não testemunhem que n é composto é menor ou igual a $\frac{1}{4^k}$.

Se um inteiro n passa k vezes no teste de Miller-Rabin, é provável que n seja primo, com uma probabilidade mínima de $1 - \frac{1}{4^k}$. Note-se que, neste caso, a primalidade de n não está garantida (se n passar no teste 10 vezes, a probabilidade de n ser composto é

de cerca de 1 em 1 milhão). Apenas podemos garantir que se n for primo, seguramente que irá passar no teste de Miller-Rabin, para qualquer base.

Alford, Granville e Pomerance concluíram, em 1994 [1], que, dado um n composto, não existe uma constante B fixa, independente de n, para o qual existe sempre uma base a, inferior a B, que testemunhe a não primalidade de n, isto é, dado B fixo, existem inteiros compostos n para os quais a menor testemunha é maior que B. Neste artigo, é referido que se n for composto e se a Hipótese de Riemann Extendida (E.R.H.)[2] for verdadeira, o menor primo que testemunha a não primalidade de n é inferior a $2\log^2 n$ (ver [3]).

Ainda no mesmo artigo, Alford, Granville e Pomerance fazem referência que, dado n, não há necessidade de escolher bases muito grandes. Por exemplo, o primeiro inteiro composto cuja menor testemunha é 11 é 3215031751 e o primeiro inteiro composto cuja menor testemunha é 23 é 341550071728321. É igualmente feita referência ao facto de Arnault (ver [2]), ter descoberto um número composto, com 337 dígitos, que é pseudoprimo forte para todas as bases primas até 199.

Segundo Rabin (ver [27]), o teste de Miller-Rabin demora $O(k\cdot\log^3 n)$ operações-bit, sendo k o número de bases utilizadas no teste.

4.3 Teste n-1, de Lucas

Contrariamente aos dois testes anteriores, este teste de primalidade é determinístico, pelo que conseguimos garantir a primalidade (ou não) de um número. Apesar de ter como base o teorema de Fermat, a conclusão acerca da primalidade de n advém do estudo de $n-1$.

Passamos a descrever a ideia lançada por Édouard Lucas [22], em 1876:

Teorema 4.5 (Lucas). *Se a e n são inteiros, com $n > 1$ e*

$$a^{n-1} \equiv 1 \mod n, \tag{4.2}$$

mas

$$a^{\frac{n-1}{q}} \not\equiv 1 \mod n, \tag{4.3}$$

para qualquer primo $q|(n-1)$, então n é primo.

[2] A Hipótese de Riemann é uma conjectura, colocada em 1859, por Bernhard Riemann e, até ao momento, ainda não foi provada.

Demonstração: Uma vez que estamos a trabalhar com equações modulares, podemos supor, sem perda de generalidade, que $0 < a < n$. De ambas as condições do teorema, é fácil concluir que $a \neq 1$.

Por outro lado, $mdc(a,\, n) = 1$, caso contrário, seja $d = mdc(a,\, n)$. Então, existem inteiros não nulos k_1 e k_2 tais que $a = k_1 d$ e $n = k_2 d$. Se

$$a^{n-1} \equiv 1 \mod n,$$

então,

$$a^{n-1} \equiv 1 \mod d.$$

Portanto,

$$1 \equiv (k_1 d)^{n-1} \equiv 0 \mod d,$$

o que só acontece se $d = 1$. Logo, $d = 1$.

De (4.2) concluímos que $ord_n(a) \mid (n-1)$, ou seja, $ord_n(a)$ é um divisor de $n-1$.

De (4.3) podemos inferir que

$$ord_n(a) \not| \frac{n-1}{q},$$

para qualquer primo q que divide $n-1$, o que equivale a dizer que $ord_n(a)$ não é um divisor próprio de $n-1$, pelo que concluímos que $ord_n(a) = n-1$.

Pelo teorema de Euler (1.6), $a^{\phi(n)} \equiv 1 \mod n$, o que quer dizer que $ord_n(a) \mid \phi(n)$, ou seja, $\phi(n) \geq n-1$.

Se n for composto, existem dois inteiros $0 < a, b < n$ tais que $ab = n$. Então, $\phi(n) \leq n-2$, o que é incompatível com o facto de $\phi(n) \geq n-1$.

Logo, n é primo. □

Definição (Raiz primitiva)**.** Se $mdc(a,\, n) = 1$ e $ord_n(a) = \phi(n)$, dizemos que a é uma *raiz primitiva* de n.

Note-se que no teorema anterior (teorema 4.5), a base a que verifica as condições 4.2 e 4.3 é uma raiz primitiva.

O facto de este teste de primalidade utilizar a decomposição de $n-1$ para concluir se n é primo ou não, pode não ser eficaz nos casos em que a factorização de $n-1$ é difícil de encontrar.

Segundo Crandall e Pomerance [11], se $n > 200560490131$ for primo, o número de tentativas para encontrar uma base a tal que $1 \leq a \leq n-1$ e que passe no teste $n-1$, de Lucas, é inferior a $2 \log \log n$.

Capítulo 5

Ataques mais conhecidos ao RSA

Com este capítulo, pretendemos descrever alguns ataques possíveis ao RSA, nomeadamente ataques de factorização e outros ataques advindos, por exemplo, de uma má escolha dos expoentes e ou d ou de um fraco conhecimento do seu funcionamento.

5.1 Ataques de factorização

Uma das formas de descobrir a chave secreta consiste no ataque de força bruta. Uma vez que n é público, o atacante pode tentar factorizá-lo.

Imaginemos que a chave secreta é um número com um máximo de 64 bits. No sistema decimal, o limite máximo deste intervalo corresponde ao número

$$18\,446\,744\,073\,709\,551\,615$$

(quase dezoito triliões e meio). Um atacante, conhecendo n, pode factorizá-lo, de modo a obter p e q e, assim, quebrar o RSA. Para factorizar n com pleno sucesso, apesar de poder ser extremamente moroso, poderá aplicar o algoritmo da divisão por exaustão, similar ao que apresentámos no capítulo 4. Para encontrar um divisor de n, e consequentemente encontrar a chave certa, basta testar a divisibilidade de todos os números primos até $[\sqrt{n}]$. Se n for um número com 64 bits, o atacante terá que efectuar, no máximo, 203 280 233 divisões, já que existem 203 280 233 primos até $[\sqrt{18\,446\,744\,073\,709\,551\,615}] = 4\,294\,967\,296$.

Sabemos, do capítulo 3, que

$$\text{tempo}(n \div m) = O(\log n \cdot \log m) \text{ operações-bit.}$$

Neste caso, cada divisão demora, no máximo, $2 \cdot 64^2 = 8192$ operações-bit. Como temos, no máximo, 203 280 233 divisões, iremos demorar, no máximo, 1 665 271 668 736 operações-bit. Se utilizarmos um computador com um processador Intel Core 2 Duo T8300 a 2.4GHz, onde a velocidade para operar com números inteiros é de cerca de 224,3 milhões de instruções por segundo (MIPS)[1], este computador irá demorar, no máximo, 7424 segundos, ou seja, pouco menos de 2h05min a descobrir um factor de 18 446 744 073 709 551 615.

Como pudemos verificar, o computador efectua muitos testes, o que implica um gasto de tempo relativamente elevado. Esse gasto de tempo aumenta proporcionalmente com o tamanho de n. Importa estimar quanto tempo iremos gastar a quebrar o RSA. Três dias? Três meses? Três anos? Há que referir que à medida que a tecnologia evolui, os computadores vão sendo cada vez mais rápidos e, assim, torna-se mais rápido descobrir uma chave. É importante que quem utilize o RSA mude (menos espaçadamente) ou aumente (mais espaçadamente) o tamanho da chave que utiliza.

Em 1991, os *RSA Laboratories* lançaram um desafio, com um prémio monetário, à primeira pessoa ou organização que conseguisse factorizar cada um dos números compostos de uma determinada lista. Estavam convictos de que estes números eram difíceis de factorizar. Com este desafio pretendia-se promover o incremento de estudos sobre a aplicação da Teoria dos Números à Computação, além de ajudar os utilizadores do RSA a escolherem chaves com um nível de segurança desejável. O menor destes números tinha 100 algarismos, mas havia também números com mais de 500 algarismos. Em 2007, o desafio foi cancelado, sendo que nem todos os números chegaram a ser factorizados.

Em todas estas chaves foram utilizados dezenas de computadores, repartindo, entre si, os testes a fazer. Assim, o tempo necessário para um simples atacante (que não tem a possibilidade de ter este número de computadores a trabalhar para o mesmo fim) quebrar uma chave aumenta drasticamente.

O tamanho da chave a utilizar no RSA depende do grau de segurança que se pretende. Quanto maior for a chave, maior será a segurança, mas também mais lento será efectuar o algoritmo do RSA. No site dos *RSA Laboratories* podemos ler que recomendam que as empresas utilizem chaves com 1024 bits.

Na assinatura digital de um Despacho Conjunto dos Ministérios das Finanças e da Administração Pública e da Educação, datado de 15 de Julho de 2008, podemos observar que a chave RSA utilizada pela senhora Ministra da Educação, Maria de

[1] Para sabermos a velocidade de um computador para operar com números inteiros, basta correr um *software* de teste de *performance* ao CPU, os quais existem disponíveis na internet.

Lurdes Rodrigues, tem um tamanho de 1024 bits, a Entidade Certificadora Comum do Estado utiliza uma chave RSA de 2048 bits e a Entidade Certificadora Raiz do Estado utiliza uma chave RSA de 4096 bits (figura 5.1).

Como iremos ver adiante, os tempos de pesquisa não são o único problema na descoberta das chaves secretas. Por exemplo, se o atacante souber qual foi o programa gerador de chaves que o cifrador utilizou, poderá testar todas as chaves geradas por esse programa até encontrar a que foi utilizada pelo cifrador. Este procedimento foi testado pela *Netscape*. Daí que, além da chave ter um número suficiente de bits para dificultar a sua descoberta, o programa gerador de chaves deverá ser, igualmente, muito bom.

Iremos descrever, de seguida, alguns ataques de factorização, todos eles de força bruta: má escolha de n (factorização de Fermat), método ρ, crivo quadrático, crivo geral dos corpos de números e curvas elípticas.

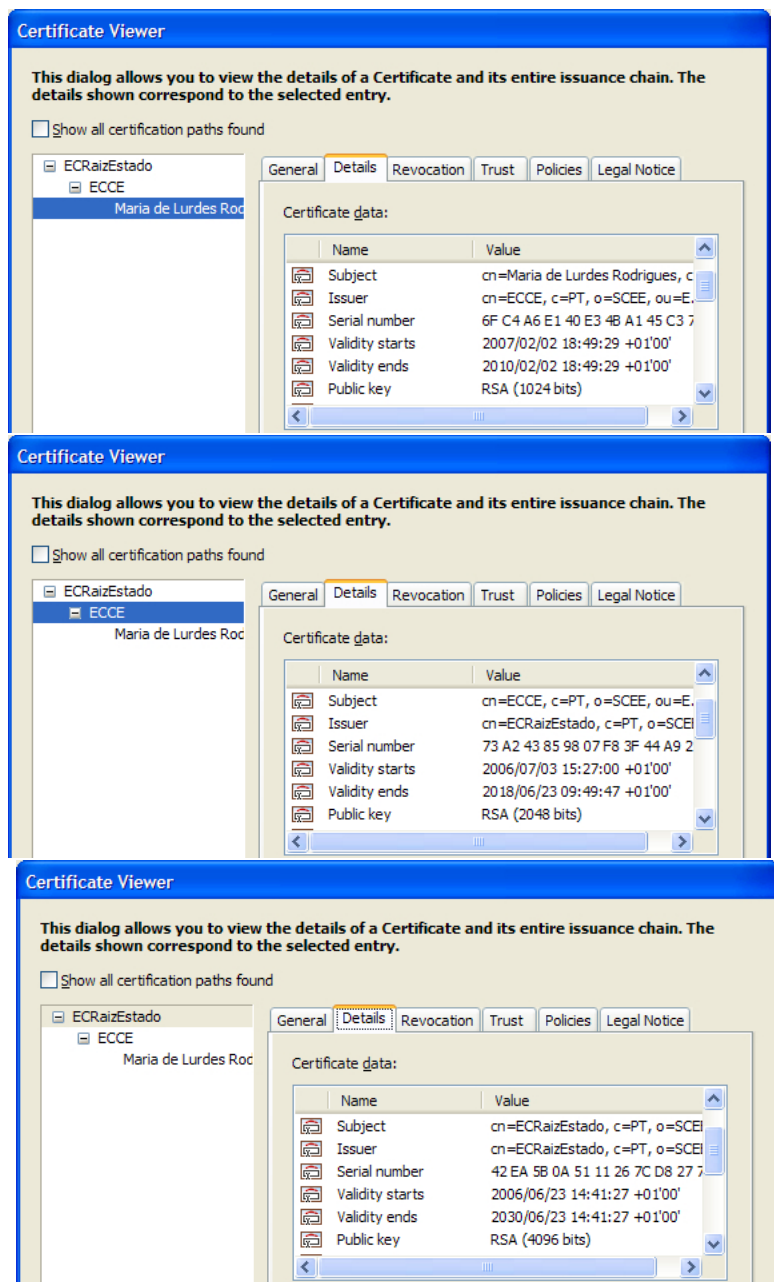


Figura 5.1: A Ministra da Educação utiliza uma chave de 1024 bits, a Entidade Certificadora Comum do Estado utiliza uma chave de 2048 bits e a Entidade Certificadora Raiz do Estado utiliza uma chave de 4098 bits.

5.1.1 Má escolha de n (factorização de Fermat, método $p-1$ de Pollard)

Factorização de Fermat

Para n ser difícil de factorizar ou para evitar que o RSA seja quebrado, p e q devem ter o mesmo número de *bits*. No entanto, se p e q forem próximos um do outro, este ataque, muito eficaz nestes casos, pode encontrá-los.

Suponhamos que n é o produto de dois primos p e q, próximos um do outro. Podemos escrever p na forma $t+s$ e q na forma $t-s$, onde s é um inteiro pequeno. Obtemos a expressão $n = t^2 - s^2$.

Teorema 5.1. *Seja n um número inteiro, positivo e ímpar. Há uma correspondência bijectiva entre a factorização de n na forma $n = ab$, com $a \geqslant b > 0$ e representações de n na forma $t^2 - s^2$, onde t e s são inteiros não negativos.*

A correspondência é dada pelas equações

$$t = \frac{a+b}{2}, \quad s = \frac{a-b}{2}, \quad a = t+s \quad \text{e} \quad b = t-s.$$

Definição. Sendo n um número real, definimos a *parte inteira* de n, e denotamos por $[n]$, como o maior número inteiro igual ou menor que n. Definimos a *parte fraccionária* de n, e denotamos por $\{n\}$, como a diferença entre o número real dado e a sua parte inteira.

Repare-se que como $n = pq = t^2 - s^2$, p e q são números próximos um do outro,

$$t = \frac{p+q}{2} \quad \text{e} \quad s = \frac{p-q}{2},$$

então s é pequeno e t será um número um pouco maior que $\sqrt{n}$.

Então, para encontrar os factores de n, bastará verificar se algum dos $[\sqrt{n}\,]+a$ divide n, começando com $a = 1,\ 2,\ 3,\ \cdots$, até encontrar um $k \geqslant 1$ tal que $([\sqrt{n}] + k)^2 - n$ é um quadrado perfeito. Quando encontrarmos esse k, descobrimos que

$$t = \left[\sqrt{n}\,\right] + k \quad \text{e} \quad s = \sqrt{\left(\left[\sqrt{n}\,\right] + k\right)^2 - n},$$

e assim conseguimos descobrir

$$p = t+s \quad \text{e} \quad q = t-s.$$

Exemplo. *Executando o algoritmo do apêndice F no Maple e fazendo*

$$n = 22787018393, \quad k = 1 \quad e \quad numTestes = 50,$$

obtemos

```
Os factores de 22787018393 são 152639 e 149287.
Foram necessárias 10 tentativas.
```

Quando os factores de n estão muito afastados um do outro, a factorização de Fermat, apesar de funcionar, pode tornar-se muito moroso. Existe, para estes casos, uma generalização da factorização de Fermat, que consiste em escolher um multiplicador k pequeno e fazer

$$t = \left[\sqrt{kn}\right] + 1, \quad \left[\sqrt{kn}\right] + 2, \quad \left[\sqrt{kn}\right] + 3, \quad \cdots, \quad \left[\sqrt{kn}\right] + j$$

até obtermos um j tal que $t^2 - kn$ é um quadrado perfeito. Desta forma,

$$kn = (t+s)(t-s).$$

Se determinarmos $d = mdc\,(t+s,\, n)$, obtemos um factor de n distinto de 1.

Se utilizarmos o algoritmo em anexo (apêndice F), e fizermos

$$n = 37408784321, \quad k = 1 \quad \text{e} \quad numTestes = 10000,$$

verificamos que

```
Os factores de 37408784321 são 250583 e 149287.
Foram necessárias 6522 tentativas.
```

Por outro lado, se introduzirmos

$$n = 37408784321, \quad k = 8 \quad \text{e} \quad numTestes = 10000,$$

verificamos que

```
Os factores de 37408784321 são 149287 e 250583.
Foram necessárias 2101 tentativas.
```

O número de tentativas diminuiu para cerca de um terço.

Como sabemos que valor escolher para k? Para respondermos a esta questão, teremos que recorrer ao método de R. Lehman, uma generalização da factorização de Fermat.

A ideia deste método é, numa primeira fase, verificar se n tem um factor próprio d, tal que $d \leq \sqrt[3]{n}$ (ver [11]). Em caso afirmativo, temos o nosso problema resolvido; caso

contrário, avançamos para a segunda fase: experimentamos todos os valores k, tais que $1 \leq k \leq [\sqrt[3]{n}]$, e, para cada um desses k's, experimentamos todos os valores inteiros t tais que

$$\left\lceil 2\sqrt{kn} \right\rceil \leq t \leq \left\lfloor 2\sqrt{kn} + \frac{\sqrt[6]{n}}{4\sqrt{k}} \right\rfloor,$$

onde

$$\lceil x \rceil = max\{n \in \mathbb{Z} | n \leq x\}$$

e

$$\lfloor x \rfloor = min\{n \in \mathbb{Z} | n \geq x\}.$$

Se $t^2 - 4kn$ for um quadrado perfeito, facilmente obtemos s^2 e conseguimos determinar $d = mdc(t + s,\ n)$, um factor próprio de n.

Para provarmos que, de facto, este método funciona, precisamos do teorema seguinte:

Teorema 5.2 (da aproximação de Dirichlet)**.** *Para qualquer número real θ e qualquer inteiro positivo m, existem inteiros a e b, com $1 \leq a \leq m$ tais que*

$$|a\theta - b| \leq \frac{1}{m+1}.$$

Demonstração: Ver [14], teorema 36. □

Teorema 5.3. *O método de Lehman está correcto e demora $O(\sqrt[3]{n}\log^3 n)$ operações-bit.*

Demonstração: Suponhamos que o método de Lehman está correcto e que nos devolve um divisor de n. Vejamos quantas operações-bit demora. Para efectuar a primeira fase do método, em que se verifica se n tem um factor próprio $d \leq \sqrt[3]{n}$, esta fase demora, no máximo, $O(\sqrt[3]{n}\log^2 n)$ operações-bit. Se, em vez de testarmos a divisibilidade de n por cada um dos inteiros até $\sqrt[3]{n}$, testarmos a divisibilidade de n por todos os primos até $\sqrt[3]{n}$, esta primeira fase irá demorar $O(\sqrt[3]{n}\log n)$.

Se passarmos à segunda fase, teremos que verificar

$$\sum_{k=1}^{[\sqrt[3]{n}]} \left[\frac{\sqrt[6]{n}}{6\sqrt{k}}\right] = O(\sqrt[3]{n})$$

vezes se $t^2 - 4kn$ é um quadrado perfeito (cada verificação demora $O(\log^3 n)$. Apenas no caso em que a raiz quadrada de $t^2 - 4kn$ é um número inteiro é que é necessário

determinar $mdc(t+s,\ n)$, o que demora $O(\log^3 n)$. Portanto, o método de Lehman demora $O(\sqrt[3]{n}\log^3 n)$.

Falta provar que o método de Lehman está correcto. Iremos assumir que n é um número composto, com apenas dois divisores primos, p e q. Iremos, também, assumir que $p,\ q > \sqrt[3]{n}$.

Pretendemos provar que existe um inteiro $k \leq [\sqrt[3]{n}]$, com $k = uv$ e tal que

$$|uq - vp| < \sqrt[3]{n}.$$

Seja

$$m = \left[n^{\frac{1}{6}} q^{\frac{1}{2}} p^{-\frac{1}{2}}\right]. \tag{5.1}$$

Pelo Teorema 5.2 (da Aproximação de Dirichlet), existem inteiros a e b, com $1 \leq a \leq m$ tais que

$$\left| a\frac{p}{q} - b \right| \leq \frac{1}{m+1}. \tag{5.2}$$

No entanto,

$$|ap - bq| \leq \frac{q}{m+1} < \frac{q}{\sqrt[6]{n}\sqrt{\frac{q}{p}}} = \frac{\sqrt{n}}{\sqrt[6]{n}} = \sqrt[3]{n}.$$

Se tomarmos $u = b$ e $v = a$, então

$$|uq - vp| < \sqrt[3]{n}.$$

Falta mostrar que $k = uv \leq [\sqrt[3]{n}]$.

De (5.2) tiramos que

$$\frac{u}{v} < \frac{p}{q} + \frac{1}{mv}.$$

Por este resultado, pelo facto de $v \leq m$ e por (5.1),

$$uv = \frac{u}{v}v^2 < \frac{p}{q}v^2 + \frac{v}{m} \leq \frac{p}{q}m^2 + 1 \leq \frac{p}{q}n^{\frac{1}{3}}\frac{q}{p} + 1 = n^{\frac{1}{3}} + 1.$$

Consideremos que

$$c = up + vp \quad \text{e que} \quad e = |uq - vp|.$$

Claramente que $c^2 - e^2 = 4kn$. Então, $c \geq 2\sqrt{kn}$.

Note-se que $e \leq n^{\frac{1}{3}}$. Seja $E = c - 2\sqrt{kn}$. Então,

$$4kn + 4E\sqrt{kn} \leq (2\sqrt{kn} + E)^2 = c^2 = 4kn + e^2 < 4kn + n^{\frac{2}{3}}.$$

Daqui concluimos que

$$E < \frac{n^{\frac{1}{6}}}{4\sqrt{k}}.$$

Logo,

$$2\sqrt{kn} \leq c < 2\sqrt{kn} + \frac{n^{\frac{1}{6}}}{4\sqrt{k}}.$$

Falta, agora, provar que $mdc(c+e, n)$ é um factor não trivial de n. Como $n|(c^2-e^2)$, ou seja, $n \mid (c-e)(c+e)$, basta mostrar que $c+e < n$. De facto,

$$c + e < 2\sqrt{kn} + \frac{n^{\frac{1}{6}}}{4\sqrt{k}} + n^{\frac{1}{3}} < 2\sqrt{(n^{\frac{1}{3}}+1)n} + \frac{n^{\frac{1}{6}}}{4\sqrt{n^{\frac{1}{3}}+1}} + n^{\frac{1}{3}} < n,$$

sendo que a última desigualdade é verdadeira apenas para $n > 21$.

$\square$

Método $p-1$ de Pollard

Como o próprio nome indica, este método de factorização foi publicado por John M. Pollard [25], em 1974.

Seja $n > 1$ um número composto e p o menor factor primo (cujo valor se desconhece) de n. Este método é particularmente eficaz nos casos em que $p-1$ não tem factores primos muito grandes. Suponhamos, então, que $p-1$ tem factores primos pequenos.

Seja k um número divisível por todos os números inteiros até um limite B. Por exemplo, podemos tomar $k = B!$ ou $k = mmc(1, 2, 3, \cdots, B)$. Se B for suficientemente grande para que $p-1$ divida k, como p é primo, do Pequeno Teorema de Fermat (1.7) retiramos que

$$a^k \equiv 1 \mod p.$$

Como $p \mid n$ e $p \mid a^k - 1$, então $p \mid mdc(n,\ a^k-1)$.

Se

$$mdc(n,\ a^k-1) \neq 1 \quad \text{e} \quad mdc(n,\ a^k-1) \neq n,$$

então encontrámos um factor próprio de n, caso contrário teremos que escolher outro a e/ou um outro valor para k.

Esquematicamente:

- Escolhe-se um inteiro k, que é o produto de todos ou de quase todos os números inteiros até um limite B, tal que $1 < B < n$. Por exemplo, k pode ser igual a $B!$ ou pode ser igual a $mmc\,(1, 2, 3, ..., B)$;

- Escolhe-se um inteiro a, com $2 \leq a \leq n-2$;

- Calcula-se $a^k \mod n$, através do método descrito na secção **potenciação modular**, no capítulo 3;
- Calcula-se $d = mdc\left(a^k - 1 \mod n,\ n\right)$, através do algoritmo de Euclides;
- Se d não for um factor próprio de n, escolhe-se um novo a e/ou um novo B (maior que o B escolhido anteriormente).

Se os factores de n forem grandes, o método $p-1$ de Pollard não é eficiente. Por outro lado, o valor de B não pode ser muito pequeno, caso contrário não conseguimos tirar conclusões. Vejamos o seguinte exemplo:

Como é do nosso interesse considerar o método $p-1$ de Pollard como um ataque ao sistema criptográfico RSA, vamos aplicar este método apenas a valores de n tais que $n = pq$, sendo p e q números primos.

Há duas formas de o método $p-1$ falhar:

$$\text{quando}\, mdc(a^k - 1,\ n) = 1 \ \text{ ou quando } \ mdc(a^k - 1,\ n) = n.$$

Se $mdc(a^k - 1,\ n) = 1$, isto significa que

$$p \ \not| (a^k - 1) \ \text{ e } \ q \ \not| (a^k - 1),$$

ou seja,

$$a^k \not\equiv 1 \mod p \ \text{ e } \ a^k \not\equiv 1 \mod q.$$

Daqui concluímos que

$$ord_p(a) \ \not| \ k \ \text{ e } \ ord_q(a) \ \not| \ k.$$

Também podemos concluir que

$$p - 1 \ \not| \ k \ \text{ e que } \ q - 1 \ \not| \ k.$$

Uma boa solução poderá passar por aumentarmos o valor de a ou de k.

Exemplo. *Consideremos o número composto* $n = 442661$.

Observemos o que acontece se escolhermos $B = 20$*:*

$$\begin{aligned} k &= mdc(1, 2, 3, \cdots, 20) \\ &= 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \\ &= 232792560. \end{aligned}$$

Fazendo $a = 2$,

$$2^{232792560} \mod 442661 = 287741$$

e

$$mdc(287741 - 1,\ 442661) = 1,$$

o que não nos permite descobrir qualquer factor de n.

Fazendo $a = 3$,

$$3^{232792560} \mod 442661 = 122694$$

e

$$mdc(122694 - 1,\ 442661) = 1.$$

À luz do que foi dito anteriormente,

$$mdc(2^{232792560} - 1 \mod 442661,\ 442661) = 1$$

porque

$$ord_p(2) \not| \ 232792560 \ \ e \ \ ord_q(2) \not| \ 232792560.$$

Analogamente, conluímos que

$$ord_p(3) \not| \ 232792560 \ \ e \ \ ord_q(3) \not| \ 232792560.$$

Como n é pequeno, facilmente descobrimos, através do Maple, que

$$442661 = 599 \cdot 739.$$

Ainda através do Maple, se corrermos o algoritmo:

```
with(numtheory);
n := 442661;
print (n, " = ", ifactor(n));
print (598, " = ", ifactor(598));
print (738, " = ", ifactor(738));
B := 20;
k := 1:
for i from 2 to B do
  k := ilcm (k, i):
end do:
print ("k = ", k);
for i from 2 to 20 do
  mdc    := igcd   (i &^ k - 1 mod n, n):
  printf ("mdc(%d ^ %d - 1 mod %d, %d ) = %d", i, k, n, n, mdc);
  print  ();
  ordem  := order (i, 599);
  printf ("ord_{599} (%d) = %d", i, ordem);
```

```
  print ();
  printf ("'ord_{599} (%d) divide %d' = %s.", i, k, type  (k / ordem, integer));
  print ();
  ordem := order (i, 739);
  printf ("ord_{739} (%d) = %d", i, ordem);
  print ();
  printf ("'ord_{739} (%d) divide %d' = %s.", i, k, type  (k / ordem, integer));
  print ();
  print ();
  print ();
end do:
```

conseguimos observar o seguinte:

```
mdc(15 ^ 232792560 - 1 mod 442661, 442661 ) = 739
ord_{599} (15) = 299
'ord_{599} (15) divide 232792560' = false.
ord_{739} (15) = 18
'ord_{739} (15) divide 232792560' = true.

mdc(19 ^ 232792560 - 1 mod 442661, 442661 ) = 599
ord_{599} (19) = 13
'ord_{599} (19) divide 232792560' = true.
ord_{739} (19) = 738
'ord_{739} (19) divide 232792560' = false.
```

Para todas as bases até 20, $mdc(a^B - 1 \mod n,\ n) = 1$, *excepto para as bases* $a = 15$ *e* $a = 19$. *Para estas bases, conseguimos, então, encontrar um factor próprio de* 442661.

Também conseguiríamos factorizar 442661 *fazendo* $B = 23$, *já que teríamos*

$$k = 5354228880$$

e

$$mdc(2^{5354228880} - 1 \mod 442661,\ 442661) = 599.$$

Se $mdc(a^k - 1,\ n) = n$, então $n \mid (a^k - 1)$, ou seja,

$$a^k \equiv 1 \mod n.$$

Daqui retiramos que

$$ord_n(a) \mid k.$$

Sabemos, do Teorema de Euler (1.6) que, se $mdc(a,\ n) = 1$, então

$$a^{\phi(n)} \equiv 1 \mod n. \tag{5.3}$$

Uma das hipóteses para (5.3) acontecer é o facto de $\phi(n) \mid k$, o que quer dizer que todos os factores de $\phi(n)$ dividem k, ou seja, k é muito alto e, portanto, há necessidade de escolher um valor mais pequeno para k.

Outra das hipóteses para (5.3) acontecer é o facto de $ord_n(a)$ ser muito pequena. Neste caso, devemos escolher outro valor para a.

Como não conhecemos $\phi(n)$, não é possível sabermos em qual dos casos estamos. Uma forma de contornarmos este problema será experimentarmos vários valores para a e, caso continuemos a ter sempre

$$mdc(a^k - 1,\ n) = n,$$

devemos reduzir o valor de k.

Exemplo. *No exemplo anterior, em que considerámos* $n = 442661$*, se* $B = 41$*, obtemos* $k = 219060189739591200$ *e, qualquer que seja a base* a*,*

$$mdc(a^{219060189739591200} - 1 \mod 442661,\ 442661) = 442661.$$

Isto irá acontecer para qualquer valor de $B \geq 41$*.*

Se olharmos atentamente para a factorização de $(p-1)(q-1)$*, onde*

$$(p-1)(q-1) = 2^2 \cdot 3^2 \cdot 13 \cdot 23 \cdot 41,$$

facilmente verificamos que $\phi(442661)$ *divide* k*, desde que* $B \geq 41$*.*

Definição. Seja k um número igual ou superior a 2. Diz-se que n é k*-suave* se todos os divisores primos de n forem iguais ou inferiores a k.

Relembremos que, no sistema criptográfico RSA, $n = pq$, com p e q primos. Sendo n público, há que escolher p e q de forma que o ataque através do método $p-1$ de Pollard não seja eficaz. Segundo Childs (ver [9], página 218), uma das formas de tornar este método extremamente ineficaz pode passar por escolher p (que já sabemos ser grande) de tal forma que $p-1 = 2q$. Desta forma, ambos os factores de n teriam sensivelmente o mesmo número de bits e o menor valor k para o qual p seria k-suave será $k = q$. Neste caso, refere o autor, é mais fácil factorizar n através do método das divisões sucessivas do que através o método $p-1$ de Pollard.

Falemos de complexidade. Neste algoritmo, no caso em que $k = B!$, obtemos

$$a^{B!} \mod n.$$

Assim sendo, neste caso existem $B-1$ exponenciações modulares. Vimos, no capítulo 3, que cada exponenciação modular demora $O(\log B \cdot \log^2 n)$. Então, calcular $a^{B!} \bmod n$ demora $O(B \cdot \log B \cdot \log^2 B)$. Novamente do capítulo 3, recorrendo-nos ao algoritmo de Euclides

$$\text{tempo}\left(mdc\left(a^k - 1 \mod n,\ n\right)\right) = O\left(\log^3 n\right).$$

Então, cada aplicação do método $p-1$ de Pollard demora

$$O\left(B \cdot \log B \cdot \log^2 n + \log^3 n\right).$$

Há que notar que se escolhermos um B demasiadamente pequeno, poderemos não ter sucesso a factorizar n. Por outro lado, se aproximarmos B a $\sqrt{n}$, o método $p-1$ torna-se mais lento que factorizar n por divisões sucessivas até $\sqrt{n}$, sendo que, neste caso, factorizar n demora

$$O\left(\frac{\sqrt{n}}{\log n}\right)$$

operações-bit.

5.1.2 Método ρ (método de Monte Carlo)

Quando um dos testes de primalidade de um número n falha, apenas sabemos que n poderá ser factorizável e, em caso afirmativo, nada se sabe acerca dos seus factores.

Este método de factorização é igualmente descrito por Pollard (ver [25]). Fazendo algumas assumpções teóricas, Pollard refere que este método frequentemente encontra um factor primo p de um número n em $O(p^{\frac{1}{2}})$ operações aritméticas (divisões), enquanto que o método das divisões sucessivas por primos inferiores ou iguais a $\sqrt{n}$ encontra p em $O(p)$ operações aritméticas.

Vejamos como funciona: suponhamos que conhecemos n e que sabemos que ele é composto. Chamemos p ao menor dos seus factores primos. Sabemos que $0 < p < n$. Mais preciso ainda, sabemos que $0 < p \leq \sqrt{n}$. Se escolhermos inteiros a e b tais que

$0 < a,\ b < n$, só no caso em que $a = b$ é que acontece $a \equiv b \mod n$. No entanto, é possível existirem a e b tais que $0 < a,\ b < n$ e

$$a \not\equiv b \mod n,$$

mas

$$a \equiv b \mod p.$$

Se $a \equiv b \mod p$, então $p\,|\,(a-b)$. Como $p\,|\,n$, então $mdc(a-b,\, n)$ será um múltiplo de p e, assim conseguimos descobrir um dos factores de n. Esta é a ideia base do método *rho*.

Então, para levarmos a cabo este método de factorização:

- escolhe-se uma correspondência $f : \mathbb{Z}/n\mathbb{Z} \longrightarrow \mathbb{Z}/n\mathbb{Z}$, podendo f ser uma função polinomial de grau igual ou superior a 2, nomeadamente polinómios com coeficientes inteiros (por exemplo, $f(x) = x^2 + 1$);

- escolhe-se um valor arbitrário para x_0 (pode ser 1 ou pode-se iniciar com um outro valor arbitrário). No seu artigo [25], Pollard escolheu $x_0 = 2$;

- calcula-se

$$x_1 = f(x_0),$$
$$x_2 = f(x_1) = f(f(x_0)),$$
$$x_3 = f(x_2) = f(f(f(x_0))),$$
$$\vdots$$

  Obtemos, assim, a sequência iniciada por x_0 e cujos termos seguintes obtêm-se fazendo $x_{j+1} = f(x_j)$, com $j = 0, 1, 2, ...$

- fazem-se comparações entre diferentes x_i's, na expectativa de encontrar x_j e x_k, com $j \neq k$, tais que

$$x_j \not\equiv x_k \mod n,$$

  mas

$$x_j \equiv x_k \mod p,$$

  para algum divisor p de n. Se isto acontecer, $p|(x_j - x_k)$ e como $p|n$ e $n \not| (x_j - x_k)$, o $mdc(x_j - x_k,\, n)$ irá devolver um divisor próprio de n, permitindo-nos factorizar n.

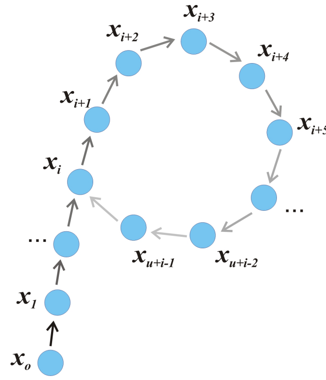


Figura 5.2: Ilustração do método rho.

Como $\mathbb{Z}/n\mathbb{Z}$ é finito, a partir de um determinado momento, a sequência acima referida irá tornar-se cíclica. A ideia encontra-se descrita na figura 5.2.

No quarto item acima descrito, pressupõe-se que calculamos sucessivamente os termos $x_k = f(x_{k-1})$ e comparamos x_k com x_j, sendo $j < k$, até encontrarmos um par $(x_k,\, x_j)$ tal que

$$mdc(x_k - x_j,\, n) = r > 1.$$

Para cada k, temos que fazer $k - 1$ comparações, o que torna este processo moroso, quando k é grande. Podemos reduzir o tempo dispendido se compararmos x_k com x_{2k}. Vejamos porquê.

Sejam x_j e x_k tais que $x_j \equiv x_k \mod p$. Sendo u o comprimento do ciclo, então $u \,|\, (j - k)$. Tomemos $a \equiv -k \mod u$. Então, $a + k = ut$, para algum inteiro t. Então,

$$x_{a+k} \equiv x_{a+k+ut} \equiv x_{2(a+k)} \mod p,$$

donde se conclui que basta compararmos x_k com x_{2k}.

Fazemos as comparações até obtermos $mdc(x_{2(i+j)} - x_{i+j},\, n)$ distinto de 1 e de n. Neste caso, iremos obter um d tal que

$$d \,|\, n$$

e

$$d \,|\, (x_{2(i+j)} - x_{i+j}).$$

Este valor d será um factor próprio de n.

Se obtivermos $d = n$, então devermos escolher outro valor para x_0 ou outra função f.

Relativamente ao quarto item da descrição do método ρ, Pollard faz referência, em [25], que se verifica o $mdc(Q_i,\, n)$, onde

$$Q_i \equiv \prod_{j=1}^{i}(x_{2j} - x_j) \mod n.$$

Note-se que, neste item, basta apenas fazer

$$mdc(x_{2j} - x_j,\, n).$$

Em anexo (Apêndice E), encontramos o algoritmo, em Maple, do funcionamento do método ρ, usando a função $f(x) = x^2 + 1$.

Exemplo. *Se corrermos, em Maple, o algoritmo do apêndice E, com*

$$n = 444413 \quad e \quad a = 4,$$

conseguimos, ao fim de 14 ciclos, factorizar n ($444413 = 521 \cdot 853$).

Segundo Koblitz (ver [18], páginas 141 e 142), há grandes probabilidades de o método ρ permitir encontrar um factor de n em

$$O(\sqrt[4]{n}\log^3 n)$$

operações-bit.

5.1.3 Crivo quadrático (*Quadratic Sieve*)

No seu artigo *A Tale of Two Sieves* [26], Pomerance descreve dois métodos para factorizar um número: o Crivo Quadrático (*Quadratic Sieve*) e o Crivo dos Corpos de Números (*Number Field Sieve*). Neste artigo, Pomerance refere que, em 1970, ainda era difícil factorizar alguns números com 20 bits. No entanto, o aparecimento da factorização através das fracções contínuas veio permitir a factorização de números até 50 bits. Em 1990, o seu próprio algoritmo veio a ser muito importante, uma vez que permitiu factorizar números com um maior número de bits. Até 1996, o recorde de factorização tinha sido de um número com 130 bits.

Antes de continuarmos com a descrição do crivo quadrático, precisamos das seguintes definições:

Definição (Resíduo quadrático)**.** Seja n um inteiro positivo e $a \in \mathbb{Z}/n\mathbb{Z}$. Se existir um elemento $b \in \mathbb{Z}/n\mathbb{Z}$ tal que $a \equiv b^2 \mod n$, dizemos que a é um *resíduo quadrático* módulo n. Se esse b não existir, dizemos que a é um *não-resíduo quadrático*.

Se a for um resíduo quadrático, módulo n, dizemos que $x^2 \equiv a \mod n$ é solúvel.

Definição (Símbolo de Legendre)**.** Seja p um primo ímpar e a um inteiro arbitrário, não divisível por p. Definimos o *símbolo de Legendre*, $\left(\frac{a}{p}\right)$, da seguinte forma:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{, se } a \equiv 0 \mod p \\ 1 & \text{, se } a \text{ é um resíduo quadrático de } p \\ -1 & \text{, se } a \text{ não é um resíduo quadrático de } p \end{cases}$$

O objectivo do Crivo Quadrático consiste encontrar um factor próprio de n, através de dois inteiros x e y tais que

$$x^2 \equiv y^2 \mod n$$

e

$$x \not\equiv \pm y \mod n.$$

Isto significa que

$$n \mid (x - y)(x + y),$$

mas

$$n \nmid (x - y) \text{ e } n \nmid (x + y).$$

Vejamos como devemos proceder para encontrar x e y.

Tendo n, seja $m = [\sqrt{n}]$ e $y_i = f(x_i) = (m + x_i)^2 - n$.

1. Defina-se uma base de factores, F, constituída por todos os números primos, p, até um majorante B, tais que n é resíduo quadrático módulo p ou $p = 2$, ou seja, constituída por todos os primos tais que

$$\left(\frac{n}{p}\right) = 1 \text{ ou } p = 2.$$

2. Encontre-se $x_1, x_2, \cdots, x_k$, próximos de m tais que $(m + x_i)^2 - n$ sejam suaves para a base de factores F, ou seja, todos os factores primos de $(m + x_i)^2 - n$ pertencem a F.

3. Utilize-se a Álgebra Linear para descobrir um subconjunto, U, de todos os números x_i tais que $\prod_{x_i \in U} ((m + x_i)^2 - n)$ seja um quadrado perfeito, ou seja, tais que

$$\prod_{x_i \in U} ((m + x_i)^2 - n) = y^2.$$

Seja x o produto de todos os $m + x_i$ utilizados para descobrir y^2. Então,

$$\begin{aligned} x^2 &= \left(\prod_{x_i \in U} (m + x_i)\right)^2 \\ &= \prod_{x_i \in U} (m + x_i)^2 \\ &\equiv \prod_{x_i \in U} \left((m + x_i)^2 - n\right) \mod n \\ &\equiv y^2 \mod n \end{aligned}$$

4. Determine-se $mdc(x-y, n)$. Se $mdc(x-y, n)$ for distinto de 1 e de n, a factorização de n fica conhecida. Caso contrário, devemos regressar ao ponto 3 para encontrar outros x e y ou regressar ao ponto 2 para encontrar mais x_i's.

Resta-nos ver como determinar o majorante B. Pomerance refere (ver [11], página 264), que é vantajoso trabalharmos com um B pequeno, uma vez que não precisamos de trabalhar com muitos resíduos B-suave para encontrarmos um subconjunto cujo produto dos seus elementos seja um quadrado perfeito. No entanto, se B for muito pequeno, a propriedade de ser B-suave é tão especial que podemos não encontrar números B-suave.

Em [26], Pomerance indica que o valor óptimo para B é

$$exp\left(\frac{1}{2}\sqrt{\ln n \ln \ln n}\right).$$

Exemplo. *Em anexo (Apêndice G), encontramos um algoritmo, em Maple, do crivo quadrático para* $n = 16843009$.

Vejamos o resultado que produz:

```
n := 16843009
m := 4104
B := 30
```

i	f(i)	2	3	5	7	13	17
18	147875	0	0	3	1	2	0
25	205632	6	3	0	1	0	1
29	238680	3	3	1	0	1	1
55	454272	7	1	0	1	2	0
83	687960	3	3	1	2	1	0
137	1143072	5	6	0	2	0	0
263	2227680	5	2	1	1	1	1
393	3380000	5	0	4	0	2	0

403	3470040	3	6	1	1	0	1
471	4087616	6	0	0	0	1	3
569	4993920	7	3	1	0	0	2
889	8087040	9	5	1	0	1	0

Considerando os expoentes, módulo 2*, obtemos*

i	f(i)	2	3	5	7	13	17
18	147875	0	0	1	1	0	0
25	205632	0	1	0	1	0	1
29	238680	1	1	1	0	1	1
55	454272	1	1	0	1	0	0
83	687960	1	1	1	0	1	0
137	1143072	1	0	0	0	0	0
263	2227680	1	0	1	1	1	1
393	3380000	1	0	0	0	0	0
403	3470040	1	0	1	1	0	1
471	4087616	0	0	0	0	1	1
569	4993920	1	1	1	0	0	0
889	8087040	1	1	1	0	1	0

A partir daqui, constroem-se os vectores-linha pertencentes a $\mathbb{Z}_2^k$*. Se considerarmos, por exemplo,* $i = 29$*, construímos o vector-linha* $(1,\ 1,\ 1,\ 0,\ 1,\ 1)$*. Com estes vectores-linha, construímos uma matriz* M*. Em geral, se tivermos* k *primos na base de factores* F*, precisamos de* $k+1$ B*-suaves para garantir que os vectores em* $\mathbb{Z}_2^k$ *sejam linearmente dependentes. Neste exemplo, basta que a matriz* M *tenha uma dimensão* 7×6*, pelo que podemos escolher* 7 *dos* 12 *vectores-linha possíveis. Ao resolvermos* $M^T X = O$*, módulo* 2*, sendo* $X^T = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ *e* O *o vector-coluna nulo, de dimensão* 6*, conseguimos verificar se alguns dos* 7 *vectores são linearmente dependentes. Em caso afirmativo, temos o problema resolvido.*

Como temos poucos i*'s, é fácil repararmos que calcular*

$$205632 \times 687960 \times 2227680,$$

é equivalente a calcular $(0, 1, 0, 1, 0, 1) + (1, 1, 1, 0, 1, 0) + (1, 0, 1, 1, 1, 1)$*, que, módulo* 2*, vai dar o vector nulo, pelo que concluímos que estes três vectores são linearmente dependentes.*

Logo,

$$\begin{aligned} 205632 \times 687960 \times 2227680 &= (561375360)^2 \\ &\equiv 5556063^2 \mod 16843009 \end{aligned}$$

pelo que $y = 5556063$ *e*

$$\begin{aligned} x &= (4104 + 25) \cdot (4104 + 83) \cdot (4104 + 263) \\ &= 75497233141 \\ &\equiv 6866803 \mod 16843009. \end{aligned}$$

Logo, $\gcd(x - y, n) = 65537$ *é um factor de* $n = 16843009$.

De facto, $16843009 = 65537 \cdot 257$.

5.1.4 Crivo geral dos corpos de números (***Number Field Sieve***)

No século XX, até finais da década 80, o melhor algoritmo para factorizar números compostos era o *Crivo Quadrático*, descrito na secção anterior. No dia 31 de Agosto de 1988, John Pollard escreveu (ver [35] e [26]) uma carta a A. M. Odlyzko, enviando uma cópia da mesma para R. P. Brent, J. Brillhant, H. W. Lenstra, C. P. Schnorr e H. Suyanna, sobre como factorizar alguns números "bonitos", através de corpos de números algébricos. Estes números "bonitos" caracterizam-se por estarem próximos de potências de números, além de outras propriedades "agradáveis". A ideia foi ilustrada com o 7º número de Fermat ($F_7 = 2^{2^7} + 1$), que já tinha sido factorizado através do método das fracções contínuas, em 1970. Através deste exemplo, Pollard especula que poderá haver outros números idênticos que serão factorizáveis pelo mesmo método. Em 1990, o 9º número de Fermat foi factorizado por este processo. Cedo se concluiu que o método funciona, não só para estes números "bonitos", mas também para qualquer número composto grande, nomeadamente com mais de 130 bits.

Esta ideia veio revolucionar o mundo das factorizações, sendo que o Number Field Sieve é, ainda actualmente, um dos métodos mais utilizados. Uma das grandes vantagens da sua utilização [11] é o facto de ser rápido, produzindo resíduos quadráticos pequenos módulo n (o número que pretendemos factorizar), assim como o facto de podermos utilizar um crivo para averiguarmos quais destes resíduos quadráticos são suaves.

À semelhança do *Crivo Quadrático*, o objectivo principal deste método é tentar obter a relação $x^2 \equiv y^2 \mod n$, sendo que, neste caso,

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

ou seja,

$$pq \mid (x-y)(x+y),$$

donde,

$$p \mid (x-y)(x+y) \;\text{ e }\; q \mid (x-y)(x+y).$$

Como p e q são primos entre si,

$$(p \mid (x-y) \,\wedge\, q \mid (x+y)) \;\vee\; (p \mid (x+y) \,\wedge\, q \mid (x-y)).$$

Obtemos p e q, fazendo $p = mdc(x-y,\, n)$ e $q = mdc(x+y,\, n)$. Existe uma probabilidade de 50% de p e q serem factores não triviais de n.

O algoritmo *Number Field Sieve*, apresenta três variantes: o *Special Number Field Sieve*, dedicado a números da forma $n = c_1 r^t + c_2 s^u$ (esta variante é a mais parecida com a que Pollard propôs); o *General Number Field Sieve*, aplicável a qualquer número composto que não é quadrado perfeito; e uma variante criada por Coppersmith, onde utiliza vários polinómios.

Sendo um algoritmo de factorização mais complicado, iremos apenas dar uma ideia do funcionamento do *General Number Sieve* (ver [8]):

Definir os parâmetros livres

Escolhemos $m \in \mathbb{Z}/n\mathbb{Z}$ e tentamos encontrar um polinómio, f, de coeficientes inteiros, tal que $f(m) \equiv 0 \mod n$. Para encontrarmos f, podemos escrever n na base m, na forma expandida, isto é,

$$n = a_d m^d + a_{d-1} m^{d-1} + \cdots + a_0,$$

e consideramos

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_0.$$

Claramente que $f(m) \equiv 0 \mod n$.

Se f for um polinómio redutível, então $n = f(m) = g(m)h(m)$, sendo que $g(m)$ e $h(m)$ são factores próprios de n, pelo que a factorização de n fica conhecida. Para averiguarmos se f é irredutível, podemos recorrer-nos ao método criado por A. Lenstra, H. Lenstra e L. Lovász [20], para factorizar polinómios com coeficientes racionais.

Segundo Crandall e Pomerance (ver [11], páginas 280, 287 e 288), podemos escolher d, o grau de f, tal que

$$d \sim \left(\frac{3\ln n}{\ln\ln n}\right)^{\frac{1}{3}},$$

e tomar $m = [\sqrt[d]{n}\,]$. Segundo Pomerance (ver [26]), se n tiver entre 100 a 200 bits, d deverá ter 5 ou 6 *bits*.

Descrição da ideia principal do algoritmo

Para avançarmos, precisamos do seguinte teorema:

Teorema 5.4. *Dado um polinómio $f(x)$, com coeficientes inteiros, uma raiz $\alpha \in \mathbb{C}$ de f e um $m \in \mathbb{Z}/n\mathbb{Z}$, tal que $f(m) \equiv 0 \mod n$, existe um homomorfismo único*

$$\phi : \mathbb{Z}[\alpha] \longrightarrow \mathbb{Z}/n\mathbb{Z}$$

tal que

- $\phi(ab) = \phi(a)\phi(b)$, *para todo o* $a, b \in \mathbb{Z}[\alpha]$*;*
- $\phi(a+b) = \phi(a) + \phi(b)$, *para todo o* $a, b \in \mathbb{Z}[\alpha]$*;*
- $\phi(1) \equiv 1 \mod n$*;*
- $\phi(\alpha) \equiv m \mod n$*.*

Tendo, então, $f(x)$ nas condições do teorema, podemos construir o homomorfismo ϕ.

Suponhamos que existem $\beta^2 \in \mathbb{Z}[\alpha]$ e $y^2 \in \mathbb{Z}/n\mathbb{Z}$, que são quadrados perfeitos. Seja, ainda, $x = \phi(\beta)$ e suponhamos que existe um conjunto finito, U, de pares de inteiros $(a,\, b)$ tais que

$$\prod_{(a,\, b) \in U} (a + b\alpha) = \beta^2 \quad \text{e que} \quad \prod_{(a,\, b) \in U} (a + bm) = y^2.$$

Então,

$$\begin{aligned} x^2 &= \phi(\beta) \cdot \phi(\beta) \\ &= \phi(\beta^2) \\ &= \phi\left(\prod_{(a,\, b) \in U} (a + b\alpha)\right) \\ &= \prod_{(a,\, b) \in U} \phi(a + b\alpha) \\ &= \prod_{(a,\, b) \in U} (a + bm) \\ &= y^2, \end{aligned}$$

que é o nosso objectivo.

Como precisamos de um quadrado perfeito em $\mathbb{Z}[\alpha]$ e um quadrado perfeito em $\mathbb{Z}/n\mathbb{Z}$, iremos descrever como podemos encontrá-los.

1) Definir os elementos suaves em $\mathbb{Z}[\alpha]$ e em $\mathbb{Z}$

Uma base racional de factores, $\mathcal{R}$, é um conjunto finito de números primos. Podemos pensar na base de factores $\mathcal{R}$ como sendo

$$\{q \ : \ q \text{ é primo e } q \leq M\},$$

para algum limite natural M. Como já foi referido anteriormente, dizemos que $l \in \mathbb{Z}$ é suave sobre uma base racional de factores, se a decomposição de l for constituída apenas por primos pertencentes a $\mathcal{R}$.

Uma base algébrica de factores, $\mathcal{A}$, é um conjunto finito $\{a + b\alpha\} \subset \mathbb{Z}[\alpha]$, com $a, b \in \mathbb{Z}$, onde para cada elemento $a + b\alpha$, não existem $c, d \in \mathbb{Z}[\alpha]$, distintos da unidade, tais que

$$c \cdot d = a + b\alpha.$$

Aqui, a definição de elemento suave em $\mathcal{A}$ é similar à anterior. Dizemos que $l \in \mathbb{Z}[\alpha]$ é suave para a base algébrica de factores $\mathcal{A}$, se existe $W \subset \mathcal{A}$ tal que

$$\prod_{(c,\, d) \in W} (c + d\alpha) = l.$$

Como é difícil trabalharmos em $\mathbb{Z}[\alpha]$, socorremo-nos do seguinte teorema para formarmos a base algébrica de factores, $\mathcal{A}$:

Teorema 5.5. *Seja $f(x)$ um polinómio com coeficientes inteiros e seja α uma raiz complexa de f. Então, o conjunto de pares $\{(r,\ p)\}$, sendo p um inteiro primo e sendo r tal que $f(r) \equiv 0 \mod p$, está em correspondência bijectiva com o conjunto de elementos $a + b\alpha \in \mathbb{Z}[\alpha]$ que constitui a base algébrica de factores.*

2) Encontrar números suave: crivo

Recordemos que o nosso último objectivo é tentar encontrar um quadrado perfeito em $\mathbb{Z}[\alpha]$ e um quadrado perfeito em $\mathbb{Z}$. Para isso, temos que encontrar pares de números $(a,\ b)$ tais que $a + b\alpha$ é um elemento suave para a base algébrica de factores ($\mathcal{A}$) e $a + bm$ é um elemento suave para a base racional de factores ($\mathcal{R}$).

Para continuarmos, precisamos dos seguintes teoremas, podendo o leitor consultar as respectivas demonstrações em [7]:

Teorema 5.6. *Seja $c + d\alpha \in \mathcal{A}$, que se representa por $(r,\ p)$, e seja $a + b\alpha \in \mathbb{Z}[\alpha]$. Temos que $(c + d\alpha) \mid (a + b\alpha)$ se e só se*

$$a \equiv -br \mod p.$$

Neste caso, também dizemos que $(r,\ p) \mid (a + b\alpha)$.

Teorema 5.7. *Um conjunto finito $\mathcal{U}$ de pares $(r,\, p) \in \mathbb{Z}[\alpha]$ representa uma completa factorização de $a + b\alpha$ se e só se*

$$\prod_{(r_i,\, p_i)\in\mathcal{U}} p_i = (-b)^d f(-a/b),$$

sendo $d = grau(f)$.

Usando estes teoremas, para encontrar elementos suave em $\mathbb{Z}[\alpha]$ e em $\mathbb{Z}/n\mathbb{Z}$, procedemos do seguinte modo:

1. fixamos um valor para b e estabelecemos um limite inteiro positivo N;

2. fazemos a variar de $-N$ até N. Escrevemos, numa lista, todos os números possíveis para $a+b\alpha$ e noutra lista, distinta da anterior, todos os números possíveis para $a + bm$, ou seja,

$$\left|\begin{array}{c} -N + b\alpha \\ (-N+1) + b\alpha \\ \vdots \\ (N-1) + b\alpha \\ N + b\alpha \end{array}\right| \quad \left|\begin{array}{c} -N + bm \\ (-N+1) + bm \\ \vdots \\ (N-1) + bm \\ N + bm \end{array}\right|$$

3. verificamos quais dos elementos $a + bm$ são $\mathcal{R}$-suave, ou seja, quais destes elementos têm, na sua factorização, apenas elementos de $\mathcal{R}$, sabendo que um $q_i \in \mathcal{R}$ divide $a + bm$ se e só se $a \equiv -bm \mod q_i$. À medida que encontramos elementos $q_i \in \mathcal{R}$ que dividem $a + bm$, vamos guardando esses mesmos q_i. No final, interessa-nos guardar todos os elementos $a + bm$ que são $\mathcal{R}$-suaves.

4. verificamos quais dos elementos $a + b\alpha \in \mathbb{Z}[\alpha]$ têm na sua factorização apenas elementos de $\mathcal{A}$, a base algébrica de factores. Relembremos (teorema 5.6) que um elemento $c + d\alpha \in \mathcal{A}$, que se representa por $(r,\, p)$, divide $a + b\alpha$ se e só se

$$a \equiv -br \mod p.$$

   À semelhança do passo anterior, à medida que vamos factorizando $a + b\alpha$, com elementos de $\mathcal{A}$, vamos guardando os factores de $a + b\alpha$. Depois de verificarmos a divisibilidade de cada um dos $a + b\alpha$ pelo último elemento de $\mathcal{A}$, sabemos que a factorização de $a + b\alpha$ está completa ao aplicamos o teorema 5.7, ou seja, se se verificar

$$\prod_{(r_i,\, p_i)\in\mathcal{U}} p_i = (-b)^d f(-a/b),$$

sendo cada $(r_i,\ p_i)$ um divisor de $a + b\alpha$. No final, guardamos os $a + b\alpha$ que verificam este teorema.

5. comparamos as duas listas. Para cada par $(a,\ b)$ para o qual $a+bm$ e $a+b\alpha$ estão marcados, guardamos os dados referentes a estes pares, para utilização posterior.

3) Verificar se existem elementos em $\mathbb{Z}[\alpha]$ e em $\mathbb{Z}$ que são quadrados perfeitos

Sabemos que $s \in \mathbb{Z}$ é um quadrado perfeito se e só se os expoentes que aparecem na decomposição, em factores primos, de s forem pares ou, por outras palavras, se forem congruentes, módulo 2, com 0. A única dificuldade prende-se em saber quando um elemento $l \in \mathbb{Z}[\alpha]$ é um quadrado perfeito.

Teorema 5.8. *Seja $l \in \mathbb{Z}[\alpha]$ com a factorização*

$$l = (a_1 + b_1\alpha)^{e_1} \cdots (a_k + b_k\alpha)^{e_k},$$

onde $\forall j \in \{1, \cdots, k\}$, $a_j + b_j\alpha$ pertence à base algébrica de factores $\mathcal{A}$. Se $l \in \mathbb{Z}[\alpha]$ é um quadrado perfeito, então, para todo o i, $e_i \equiv 0 \mod 2$.

Teorema 5.9. *Seja U o conjunto dos pares (a, b) tais que $\prod_{(a,\, b) \in U} (a+b\alpha)$ é um quadrado perfeito em $\mathbb{Z}[\alpha]$. Então, para todo o $(s,\ q)$ que obedece ao teorema 5.5, tal que*

$$(s,\ q) \not| a + b\alpha,$$

temos que,

$$\prod_{(a,\, b) \in U} \left(\frac{a + bs}{q} \right) = 1.$$

Note-se que os dois teoremas enunciados dão condições necessárias, mas não suficientes para garantir que um elemento $l \in \mathbb{Z}[\alpha]$ é um quadrado perfeito. Na prática, fazemos o seguinte:

1. verificamos se a factorização de $l \in \mathbb{Z}[\alpha]$

   $$l = (a_1 + b_1\alpha)^{e_1}(a_2 + b_2\alpha)^{e_2} \cdots$$

   obedece à condição, $e_i \equiv 0 \mod 2$, para todo o i.

2. consideramos $\mathcal{Q}$, um conjunto de elementos $(s,\ q)$, onde q é um número primo e $s \in \mathbb{Z}/n\mathbb{Z}$ é tal que $f(s) \equiv 0 \mod p$. Escolhemos $\mathcal{Q}$ tal que $(s,\ q) \not| (a + b\alpha)$,

para todo o $a + b\alpha$ que aparece na factorização de l. Verificamos que, para todo o par $(s,\ q) \in \mathcal{Q}$,

$$\prod_{(a,\, b) \in U} \left(\frac{a + bs}{q} \right) = 1,$$

onde $((a+bs)/q)$ é o símbolo de Legendre e sendo $\mathcal{U}$ o conjunto de todos os $(a,\ b)$ tais que $\prod_{(a,\, b) \in \mathcal{U}} (a + b\alpha)$ é um quadrado perfeito. Ao conjunto $\mathcal{Q}$ chamamos base carácter quadrático e cada $(s,\ q) \in \mathcal{Q}$ é chamado um carácter quadrático.

3. se ambas as condições anteriores são satisfeitas, então provavelmente l é um quadrado perfeito em $\mathbb{Z}[\alpha]$. A probabilidade de l ser um quadrado perfeito aumenta à medida que aumentamos o número de elementos de $\mathcal{Q}$.

4) Dos números suave aos quadrados perfeitos

Até ao momento encontrámos um conjunto $\mathcal{U}$ de pares do tipo $(a,\ b)$ tais que $a + bm$ é suave na base $\mathcal{R}$ e $a + b\alpha$ é suave na base $\mathcal{A}$. Iremos descrever como encontrar um quadrado perfeito em $\mathbb{Z}$ e outro quadrado perfeito em $\mathbb{Z}[\alpha]$.

Consideremos que a base $\mathcal{R}$ tem k elementos e a base $\mathcal{A}$ tem l elementos. Iremos escolher uma base carácter quadrática arbitrária, $\mathcal{Q}$, com u elementos. As bases $\mathcal{R}$ e $\mathcal{A}$ servirão para encontrar quadrados perfeitos, enquanto que a base $\mathcal{Q}$ servirá para verificar se o resultado é um quadrado perfeito.

Cada elemento $(a,\ b) \in \mathcal{U}$ pode ser representado por um vector linha, constituído por $1 + k + l + u$ componentes. A primeira componente é 0 ou 1, consoante $a + bm$ seja positivo ou negativo, respectivamente. As k componentes seguintes são dadas através dos expoentes, módulo 2, da decomposição, em factores primos, de $a + bm$. As l componentes seguintes são 0 ou 1, consoante $a + b\alpha$ seja divisível ou não, respectivamente, por um particular elemento de $\mathcal{A}$. As últimas u componentes são dadas através dos elementos de $\mathcal{Q}$. Cada uma destas u componentes é 0 se, para um determinado $(s,\ q) \in \mathcal{Q}$, o símbolo de Legendre $\left(\frac{a + bs}{q} \right)$ é igual a 1; caso contrário, a componente tomará o valor 1.

Supondo que $\mathcal{U}$ tem y elementos, iremos construir X, uma matriz de dimensão

$$y \times (1 + k + l + u),$$

onde cada linha é composta pelas componentes do vector que representa $(a,\ b) \in \mathcal{U}$. Para encontrar $V \subset \mathcal{U}$, tal que o produto de todos os seus elementos é um quadrado

perfeito, temos que encontrar um vector coluna $S = [S_1\ S_2\ \cdots\ S_y]^T$ tal que

$$X^T \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_y \end{bmatrix} \equiv 0 \mod 2.$$

Se $y > 1 + k + l + u$, então este sistema tem solução.

Supondo que encontramos um subconjunto V de $\mathcal{U}$ tal que

$$\prod_{(a_i,\, b_i) \in V} (a_i + b_i m)$$

é um quadrado perfeito em $\mathbb{Z}$ e

$$\prod_{(a_i,\, b_i) \in V} (a_i + b_i \alpha)$$

é um quadrado perfeito em $\mathbb{Z}[\alpha]$, verificamos que:

- $\prod_{(a_i,\, b_i) \in V} (a_i + b_i m)$ é positivo. Se c_j representar a primeira componente do vector-linha de $a_j + b_j m$, então a primeira componente de $\prod_{(a_i,\, b_i) \in V} (a_i + b_i m)$ é congruente, módulo 2, com 0 se e só se $\sum c_j \equiv 0 \mod 2$, ou seja, o número de vectores-linha onde a primeira componente é igual a 1 é par;

- todos os expoentes da decomposição, em factores primos, de

  $$\prod_{(a_i,\, b_i) \in V} (a_i + b_i m)$$

  são pares e este produto é suave na base $\mathcal{R}$. Repare-se que

  $$\sum_{(a_i,\, b_i) \in V} e_i \equiv 0 \mod 2 \quad \Leftrightarrow \quad \sum_{(a_i,\, b_i) \in V} (e_i \mod 2) \equiv 0 \mod 2.$$

- todos os expoentes da decomposição de $\prod_{(a_i,\, b_i) \in V} (a_i + b_i \alpha)$, na base $\mathcal{A}$ são pares. Analogamente ao ponto anterior, se cada um dos $a_i + b_i \alpha$ é suave na base $\mathcal{A}$, então o produto anterior também o é.

- para todo o $(s, q) \in \mathcal{Q}$, $\prod_{(a_i,\, b_i) \in V} \left(\frac{a_i + b_i s}{q} \right) = 1$.

  Então, o número de pares $(s,\, q)$ tal que $\left(\frac{a_i + b_i s}{q} \right) = -1$ deve ser par. Assim,

$$\prod_{(a_i,\, b_i) \in V} \left(\frac{a_i + b_i s}{q} \right) = 1 \Leftrightarrow \sum_{(a_i,\, b_i) \in V} \left\{ \begin{array}{ll} 1, & \left(\frac{a_i + b_i s}{q} \right) = 1 \\ 0, & \text{outra situação} \end{array} \right\} \equiv 0 \mod 2.$$

Se estas quatro condições forem verificadas, então $\prod_{(a_i,\, b_i)\in V} (a+bm)$ é um quadrado perfeito em $\mathbb{Z}$ e $\prod_{(a_i,\, b_i)\in V} (a+b\alpha)$ é um quadrado perfeito em $\mathbb{Z}[\alpha]$ se e só se a soma da representação vectorial de cada $(a_j,\, b_j) \in V$ for congruente, módulo 2, com 0.

O *Number Field Sieve* tem uma complexidade (ver [18]) de

$$O(exp[(c \log n)^{\frac{1}{3}} (\log \log n)^{\frac{2}{3}}]).$$

O valor de c depende do tipo de *Number Field Sieve* que estivermos a utilizar. Se usarmos o *Special Number Field Sieve*, $c = (32/9)^{1/3} \approx 1,527$. Se estivermos a usar o *General Number Field Sieve*, que foi o que descrevemos, $c = (64/9)^{1/3} \approx 1,923$. Se considerarmos a versão criada por Coppersmith, $c = \frac{1}{3}(92 + 26\sqrt{13})^{\frac{1}{3}} \approx 1,902$.

Curiosidade

Em http://www.crypto-world.com/FactorAnnouncements.html, podemos verificar que o último número factorizado, até ao momento, através do *General Number Field Sieve*, foi o RSA-640, um número com 640 bits, a saber,

3107418240490043721350750035888567930037346022842727545720161948 8_
2320644051808150455634682967172328678243791627283803341547107310 8_
5019195485290073377248227835257423864540146917366024776523466 09,

factorizado a 4 de Novembro de 2005, por Bahr, Boehm, Franke e Kleinjung.

Em http://www.crypto-world.com/announcements/rsa640.txt, podemos ler que os factores são

1634733645809253848443133883865090859841783670033092312181110852 3_
89333100104508151212118167511579

e

1900871281664822113126851573935413975471896789968515493666638539 0_
88027103802104498957191261465571

A aplicação do crivo demorou cerca de 3 meses a ser concluído e foi realizado em 80 computadores *Opteron* a *2.2 GHz*. A etapa da matriz demorou cerca de um mês e meio a ser resolvida e envolveu também 80 computadores, todos eles ligados a um servidor. Ao todo, sem contar com o tempo que demoraram a escolher f, foram gastos cerca de 5 meses a factorizar n. O prémio, pela descoberta da factorização do RSA-640, foi de US$20000, atribuídos por *RSA Security*.

Um exemplo

Para mostrarmos como funciona o *Number Field Sieve*, iremos descrever o exemplo explorado em [8]. Pretendemos factorizar $n = 45113$. Apesar de termos feito referência a uma maneira de obter m, Case preferiu escolher $m = 31$. Se escrevermos n na base m, obtemos

$$m^3 + 15m^2 + 29x + 8,$$

pelo que obtemos

$$f(x) = x^3 + 15x^2 + 29x + 8.$$

Sendo α uma raiz complexa de f, definimos o homomorfismo

$$\phi : \mathbb{Z}[\alpha] \longrightarrow \mathbb{Z},$$

onde, para quaisquer $a, b \in \mathbb{Z}$, $\phi(a + b\alpha) = a + bm$.

Precisamos, agora, de definir uma base racional de factores, $\mathcal{R}$, e uma base algébrica de factores $\mathcal{A}$. Se definirmos $M = 30$,

$$\mathcal{R} = \{2,\ 3,\ 5,\ 7,\ 11,\ 13,\ 17,\ 19,\ 23,\ 29\}.$$

Para a base algébrica de factores, $\mathcal{A}$, podemos procurar todas as soluções de

$$f(r) \equiv 0 \mod p,$$

fazendo $2 \leq p \leq 90$. A base algébrica de factores será constituida por todas as soluções desta equação. Sendo assim,

$$\begin{aligned}\mathcal{A} = \{&(\ 0,\ \ 2),\ (\ 6,\ \ 7),\ (13,\ 17),\ (11,\ 23),\ (26,\ 29),\ (18,\ 31),\ (19,\ 41),\\ &(13,\ 43),\ (\ 1,\ 53),\ (46,\ 61),\ (\ 2,\ 67),\ (\ 6,\ 67),\ (44,\ 67),\ (50,\ 73),\\ &(23,\ 79),\ (47,\ 79),\ (73,\ 79),\ (28,\ 89),\ (62,\ 89),\ (73,\ 89)\}\end{aligned}$$

Estamos, agora, em condições de determinar todos os pares $(a,\ b)$ tais que $a + bm$ tem, na sua decomposição, apenas factores primos de $\mathcal{R}$ e tais que $a + b\alpha$ tem, na sua decomposição, apenas elementos de $\mathcal{A}$. Usando o crivo descrito anteriormente, consideramos todos os valores de b entre 1 e 41 e todos os valores de a entre -400 e 400. Seguidamente, guardamos todos os $(a,\ b)$ tais que $a + bm$ é $\mathcal{R}$-suave. Para estes $(a,\ b)$ guardados, verificamos se o respectivo $a + b\alpha$ é $\mathcal{A}$-suave (recorremo-nos, aqui, dos teoremas 5.6 e 5.7). Dos pares $(a,\ b)$ para os quais $a + bm$ é $\mathcal{R}$-suave, guardamos

apenas aqueles em que $a + b\alpha$ é $\mathcal{A}$-suave. Ficam, assim, os seguintes pares:

$$\begin{array}{cccccc}
(-73,\ 1) & (-13,\ 1) & (-6,\ 1) & (-2,\ 1) & (-1,\ 1) & (1,\ 1) \\
(2,\ 1) & (3,\ 1) & (13,\ 1) & (15,\ 1) & (23,\ 1) & (61,\ 1) \\
(1,\ 2) & (3,\ 2) & (33,\ 2) & (2,\ 3) & (5,\ 3) & (19,\ 4) \\
(14,\ 5) & (37,\ 5) & (313,\ 5) & (11,\ 7) & (15,\ 7) & (-7,\ 9) \\
(119,\ 11) & (-247,\ 12) & (175,\ 13) & (5,\ 17) & (-1,\ 19) & (35,\ 19) \\
(17,\ 25) & (49,\ 26) & (375,\ 29) & (9,\ 32) & (1,\ 33) & (78,\ 37) \\
(5,\ 41) & (9,\ 41) & & & &
\end{array}$$

Precisamos, neste momento, de determinar uma base carácter quadrático, $\mathcal{Q}$. Para isso, precisamos de escolher primos q que não estejam em $\mathcal{A}$ (neste exemplo, Case escolheu os primos 97, 101, 103 e 107). Seguidamente, precisamos de encontrar todos os s tais que $f(s) \equiv 0 \mod q$. Obtemos, assim,

$$\mathcal{Q} = \{(28,\ 97),\ (87,\ 101),\ (47,\ 103),\ (4,\ 107),\ (8,\ 107),\ (80,\ 107)\}.$$

Podemos, agora, construir a matriz X. Se, por exemplo, considerarmos o par $(35,\ 19)$, como $35 + 19m$ é positivo, então a primeira componente do vector linha é 0. Sendo

$$35 + 19m = 2^4 \cdot 3 \cdot 13,$$

então as 10 componentes seguintes, módulo 2, são

$$0,\ 1,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0.$$

Como, de entre os elementos da base $\mathcal{A}$, apenas $(44,\ 67)$ e $(73,\ 79)$ dividem $35 + 19\alpha$, então as 20 componentes seguintes são

$$0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0.$$

Por fim, como

$$\left(\frac{35 + 19 \cdot 28}{97}\right) = \left(\frac{35 + 19 \cdot 87}{101}\right) = \left(\frac{35 + 19 \cdot 8}{107}\right) = -1$$

e

$$\left(\frac{35 + 19 \cdot 47}{103}\right) = \left(\frac{35 + 19 \cdot 4}{107}\right) = \left(\frac{35 + 19 \cdot 80}{107}\right) = 1,$$

então as últimas 6 componentes do vector linha representante de $(35,\ 19)$ são

$$1,\ 1,\ 0,\ 0,\ 1,\ 0.$$

Portanto, o vector linha que representa (35, 19) é

(0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0).

Fazendo o mesmo para os restantes elementos de $\mathcal{U}$, obtemos a matriz X.

Resolvendo o sistema

$$X^T \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{38} \end{bmatrix} \equiv 0 \mod 2,$$

obtemos a solução

$$S =< S_1,\ S_2,\ \cdots,\ S_{38} >,$$

tal que

$$
\begin{aligned}
S_1 &= S_{26} = 0 \\
S_2 &= S_{13} = S_{20} = S_{37} + S_{38} \\
S_3 &= S_{31} + S_{36} + S_{37} \\
S_4 &= S_{31} + S_{32} + S_{36} + S_{37} + S_{38} \\
S_5 &= S_{21} = S_{35} + S_{36} + S_{37} + St_{38} \\
S_6 &= S_{31} + S_{32} + S_{35} + S_{37} \\
S_7 &= S_{35} + S_{38} \\
S_8 &= S_{24} = S_{31} + S_{35} + S_{38} \\
S_9 &= S_{32} + S_{35} + S_{36} + S_{37} \\
S_{10} &= S_{32} \\
S_{11} &= S_{14} = S_{31} + S_{32} + S_{35} + S_{36} + S_{37} + S_{38} \\
S_{12} &= S_{31} + S_{37} \\
S_{15} &= S_{32} + S_{35} + S_{36} + S_{38} \\
S_{16} &= S_{35} + S_{36} + S_{37} \\
S_{17} &= S_{31} + S_{32} + S_{35} + S_{36} + S_{37} \\
S_{18} &= S_{31} + S_{32} + S_{37}
\end{aligned}
$$

$$
\begin{aligned}
S_{19} &= S_{32} + S_{37} \\
S_{22} &= S_{34} = S_{36} + S_{37} + S_{38} \\
S_{23} &= S_{31} + S_{32} + S_{36} + S_{38} \\
S_{25} &= S_{31} + S_{32} + S_{37} \\
S_{27} &= S_{32} + S_{35} + S_{36} + S_{37} + S_{38} \\
S_{28} &= S_{36} \\
S_{29} &= S_{31} + S_{32} + S_{35} + S_{36} \\
S_{30} &= S_{38} \\
S_{33} &= S_{36}
\end{aligned}
$$

Fazendo $S_{31} = S_{33} = S_{34} = S_{35} = S_{36} = S_{37} = S_{38} = 0$ e $S_{32} = 1$, obtemos a solução (0,0,0,1,0,1,0,0,1,1,1,0,0,1,1,0,1,1,1,0,0,0,1,0,1,0,1,0,1,0,0,1,0,0,0,0,0,0)

Então, para os pares $(a,\ b) \in V$, com

$$
\begin{aligned}
V = \{&(-2,\ 1),\ (1,\ 1),\ (13,\ 1),\ (15,\ 1),\ (23,\ 1),\ (3,\ 2),\\
&(33,\ 2),\ (5,\ 3),\ (19,\ 4),\ (14,\ 5),\ (15,\ 7),\ (119,\ 11),\\
&(175,\ 13),\ (-1,\ 19),\ (49,\ 26)\}
\end{aligned}
$$

temos que $\prod_{(a,\ b)\in V} (a+bm)$ é um quadrado perfeito em $\mathbb{Z}$ e $\prod_{(a,\ b)\in V} (a+b\alpha)$ é um quadrado perfeito em $\mathbb{Z}[\alpha]$.

Ora,

$$
\prod_{(a,\ b)\in V} (a + bm) = 45999712751795195582606376960000 = y^2,
$$

donde $y = 6782308806873600$, e

$$
\begin{aligned}
&\prod_{(a,\ b)\in V} (a + b\alpha) = \\
&= 58251363820606365x^2 + 149816899035790332x + 75158930297695972 \\
&= \beta^2.
\end{aligned}
$$

Falta-nos, agora, fazer a raiz quadrada de β^2.

Há erro no artigo que estamos a seguir (ver [8]). O cálculo da raiz quadrada, em $\mathbb{Z}[\alpha]$, é complicado (recomendamos o leitor a consultar [7]), pelo que descrevemos apenas o que faríamos posteriormente. Após obtermos β e $\phi(\beta)$, determinávamos $x = \phi(\beta)$ e calculávamos $mdc(y - x,\ n)$, podendo-se obter um factor próprio de n. Se

não obtivermos um factor próprio de n, aplicamos outra vez o crivo, alterando alguns dados iniciais.

5.1.5 Método curvas elípticas - factorização de inteiros, segundo Lenstra

Encontramos, em anexo (apêndice H), noções gerais para melhor compreendermos este capítulo, que o leitor deverá consultar.

Em 1987, Lenstra [21] sugeriu este método probabilístico de factorização, muito utilizado actualmente, e cujo funcionamento passamos a descrever.

Seja n um número composto e ímpar e suponhamos que $1 < p < n$ é um (ainda desconhecido) factor primo de n, maior que 3. Pretendemos encontrar p. Para isso:

- escolhemos uma curva elíptica E sobre F_n, que já sabemos ser da forma

$$y^2 = x^3 + ax + b,$$

  com $a,\ b \in F_n$, tal que $4a^3 + 27b^2 \neq 0$. Escolhemos, também, um ponto P, distinto do ponto no infinito, pertencente à curva E. Para escolhermos $(E,\ P)$ de forma "aleatória", podemos escolher 3 números inteiros aleatórios a, x e y, e calcular o valor de b, sabendo que $b = y^2 - x^3 - ax$. Por simplicidade, iremos assumir que $mdc(4a^3+27b^2,\ n) = 1$ (pelo facto de trabalharmos em F_n, o máximo divisor comum entre estes dois números é um inteiro compreendido entre 1 e n). Se obtivermos $mdc(4a^3 + 27b^2,\ n) > 1$, então ou $n \mid (4a^3 + 27b^2)$ (e, neste caso devemos escolher outros valores para a e b) ou obtemos um factor próprio de n e, neste caso, a factorização de n fica conhecida. O facto de assumirmos que $mdc(4a^3 + 27b^2,\ n) = 1$ implica que $mdc(4a^3 + 27b^2,\ p) = 1$ e, portanto, que E é uma curva elíptica módulo p.

- escolhemos dois limites superiores, B e C, e calculamos k tal que

$$k = \prod_{l \leq B} l^{\alpha_l},$$

  onde l é um divisor primo menor ou igual a B e α_l é o maior expoente tal que $l^{\alpha_l} \leq C$, ou seja, $\alpha_l = [\log C / \log l]$. O valor B irá representar o limite superior dos divisores primos de k, que irá multiplicar por P. Quanto maior for o valor de B, maior é a probabilidade de obtermos $kP \equiv 0 \mod n$ e, consequentemente,

$kP \equiv 0 \mod p$, para algum divisor p de n. Por outro lado, quanto maior for o valor de B, maiores serão os gastos de tempo a calcular kP mod n, pelo que B deve ser escolhido de forma sensata. Devemos recordar que o teorema de Hasse diz que se $p + 1 + 2\sqrt{p} < C$ e a ordem de $E \mod p$ não é divisível por nenhum primo maior que B, então k é um múltiplo desta ordem e, portanto, temos que $kP \equiv 0 \mod p$.

- Calculamos $kP \mod n$, usando as fórmulas

$$\begin{cases} x_3 = \left(\dfrac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2, \\ \\ y_3 = \left(\dfrac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1, \end{cases} \tag{5.4}$$

para adicionar $P = (x_1, y_1)$ e $Q = (x_2, y_2)$, com $x_1 \neq x_2$, e as fórmulas

$$\begin{cases} x_3 = \left(\dfrac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1, \\ \\ y_3 = -y_1 + \left(\dfrac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3). \end{cases} \tag{5.5}$$

para obter $2P$, com $P = (x_1, y_1)$. A única dificuldade que poderemos encontrar no cálculo de $kP \mod n$ será quando $x_2 - x_1$ não for primo com n ou quando $2y_1$ não for primo com n, uma vez que, em qualquer uma das situações, não existe inverso módulo n. De acordo com o teorema H.3, isto acontece quando existe algum múltiplo k_1P (numa soma parcial para obter $kP \mod n$) para o qual existe um $p \mid n$ tal que $k_1P \equiv O \mod p$, isto é, a ordem de $P \mod p$ em $E \mod p$ divide k_1. Ao aplicarmos o algoritmo de Euclides para tentar obter o inverso de $x_2 - x_1$, obtemos $mdc(x_2 - x_1, n)$. Podem acontecer uma das situações seguintes: ou $mdc(x_2 - x_1, n)$ é um factor não trivial de n ou obtemos $mdc(x_2 - x_1, n) = n$ (neste caso, pelo teorema H.3, $k_1P \equiv O \mod p$, para todos os divisores primos p de n, o que, segundo [18], é algo muito improvável de acontecer, já que n é o produto de dois primos grandes). Sendo assim, é quase certo que quando tentamos calcular $k_1P \mod n$, com k_1 múltiplo da ordem de $P \mod p$ para algum p que divide n, iremos obter um factor próprio de n. Se conseguirmos calcular $kP \mod n$ sem qualquer problema, devemos escolher uma nova curva elíptica $E(a, b)$ e/ou um novo ponto P e aplicar novamente o método descrito.

Em anexo (apêndice I) encontra-se um algoritmo, em Maple, do método curvas elípticas, para factorizar $n = 45457$.

Exemplo. *Suponha-se que* $n = 45457$, $B = 20$ *e* $C = 244$.

Ora, $k = 2^7 \cdot 3^5 \cdot 5^3 \cdot 7^2 \cdot 11^2 \cdot 13^2 \cdot 17 \cdot 19 = 1258336903824000$.

Imaginemos que escolhemos trabalhar sobre a curva elíptica

$$E : y^2 = x^3 + 5697x - 26410237138$$

e com o ponto $P = (3002,\ 25708)$.

Como iremos trabalhar sobre módulo n, *interessa-nos olhar para*

$$E \mod n : y^2 = x^3 + 5697x + 7120 \mod 45457.$$

A ideia consiste em calcularmos $2^i P$, *com* $i = 1, 2, \cdots, 50$ *(uma vez que* k, *no sistema binário, tem* 51 *bits), com recurso às fórmulas (5.5); se o último bit de* k *for* 1, *atribuímos* $k_1 P = P$, *caso contrário, atribuímos* $k_0 P = NULL$*; a seguir, olhamos, sucessivamente, para os restantes bits de* k, *da direita para a esquerda e, cada vez que o bit* i *for* 1, *vamos ao* $k_i P \mod n$ *acumulado até ao momento e somamos* $2^i P$, *até chegarmos a* $kP \mod n$. *Caso cheguemos a* $kP \mod P$ *sem que tenhamos qualquer dificuldade em determinar o inverso, módulo* n, *do denominador das fórmulas (5.4) ou (5.5), então devemos escolher nova curva elíptica ou um novo valor de* k, *e devemos começar tudo de novo. Se chegarmos a uma altura em que, usando as fórmulas (5.4) ou (5.5), não exista o inverso, módulo* n, *do denominador, então é porque* $\text{mdc}(\text{denominador}, n) \neq 1$*: se este máximo divisor comum for inferior a* n, *então conseguimos encontrar a factorização de* n*; se for igual a* n, *devemos escolher outra curva elíptica.*

Ora,

$$k = (100011110000011100110110010000101001000000001010000000)_2.$$

Iremos designar k_i *como sendo o número binário a partir do bit* i *de* k, *ou seja,*

$$k_i = k \mod 2^{i+1}.$$

Assim,

- *bit de ordem* $0 = 0$
  $k_0 P \mod n = NULL$

- *bit de ordem* $1 = 0$
  $2P \mod n = (P + P) \mod n = (31117,\ 22502)$
  $k_1 P \mod n = NULL$

- *bit de ordem* 2 $= 0$
  $4P \mod n = 2(2P) \mod n = (24456,\ 28313)$
  $k_2P \mod n = NULL$

- *bit de ordem* 3 $= 0$
  $8P \mod n = 2(4P) \mod n = (30770,\ 35447)$
  $k_3P \mod n = NULL$

- *bit de ordem* 4 $= 0$
  $16P \mod n = 2(8P) \mod n = (35213,\ 5693)$
  $k_4P \mod n = NULL$

- *bit de ordem* 5 $= 0$
  $32P \mod n = 2(16P) \mod n = (15849,\ 29442)$
  $k_5P \mod n = NULL$

- *bit de ordem* 6 $= 0$
  $64P \mod n = 2(32P) \mod n = (36601,\ 19638)$
  $k_6P \mod n = NULL$

- *bit de ordem* 7 $= 1$
  $128P \mod n = 2(64P) \mod n = (24177,\ 747)$
  $k_7P \mod n = P \mod n = (24177,\ 747)$

- *bit de ordem* 8 $= 0$
  $256P \mod n = 2(128P) \mod n = (8494,\ 18185)$
  $k_8P \mod n = (24177,\ 747)$

- *bit de ordem* 9 $= 1$
  $512P \mod n = 2(256P) \mod n = (23830,\ 34647)$
  $k_9 \mod n = k_8 \mod n + 512P \mod n$
  Como $mdc(x_2 - x_1,\ n) = mdc(23830 - 24177,\ 45457) = 347$*, concluimos que* $n = 347 \times 131$.

5.2 Ataques de implementação do RSA

Já vimos que existem quatro parâmetros, no RSA, que devem ser mantidos em segredo: d, p, q e $\phi(n)$. Se o atacante descobrir qualquer um destes parâmetros, consegue descobrir a mensagem original. No entanto, se o cifrador não implementar correctamente o RSA, o atacante consegue quebrar a cifra, mesmo sem conhecer qualquer um dos parâmetros referidos.

Segundo Song Yan [34], os ataques ao RSA com maior sucesso não são aqueles que tentam factorizar n, nem aqueles que se valem do facto de n ser pequeno: são aqueles que tentam descobrir falhas na implementação do sistema RSA. Neste capítulo, iremos descrever alguns destes ataques.

5.2.1 Ataque da procura exaustiva (***Forward search attack*** ou ***Short plaintext attack***)

Se o atacante conhecer C e souber que a mensagem original é pequena (note-se que, apesar de M ser pequena, é possível C ser tão grande quanto n), pode testar todas as mensagens possíveis para M (ou apenas algumas, se tiver suspeitas acerca do conteúdo da mensagem original). Por outras palavras, o atacante cifra todas as mensagens possíveis

$$C_1 \equiv M_1^e \mod n, \ \ C_2 \equiv M_2^e \mod n, \ \ \cdots, \ \ C_k \equiv M_k^e \mod n,$$

e verifica se alguma delas corresponde a C, ou seja, verifica se existe algum $1 \leq i \leq k$ tal que $C_i = C$. Se tiver sucesso, $M = M_i$.

Em alternativa, o atacante pode escolher um limite do tipo 10^r e construir duas sequências:

$$\begin{aligned} &Cx^{-e} \mod n, && \text{para todo o } 1 \leq x \leq 10^r \\ &y^e \mod n, && \text{para todo o } 1 \leq y \leq 10^r \end{aligned}$$

Em seguida, o atacante tenta encontrar um valor na primeira sequência que também esteja presente na segunda sequência. Se encontrar um x_0 e um y_0 tais que

$$Cx_0^{-e} \equiv y_0^e \mod n,$$

então consegue descobrir M, já que

$$C \equiv (x_0 y_0)^e \mod n,$$

e, portanto, $M = x_0 y_0$.

Como pudemos verificar, este ataque funciona bem quando M é o produto de dois números menores que 10^r. Se, por exemplo $r = 8$, então o atacante terá que calcular, no máximo, $2 \cdot 10^8$ números, e conseguirá descobrir M se C for o produto de dois números inferiores a 10^8. Relativamente ao tempo despendido, este método é muito mais eficaz do que o método em que testa todas as 10^{16} possibilidades para M.

Uma forma de evitar este tipo de ataque consiste em acrescentar um conjunto de bits aleatórios no início e/ou no fim da mensagem original, que serão eliminados depois de decifrar C. A este procedimento dizemos que estamos a "salgar"a mensagem.

5.2.2 Ataque do módulo comum (*Common modulus attack*)

Quem alertou para este possível ataque foi Simmons [31], em 1993, cujo trabalho foi, mais tarde, complementado por DeLaurentis [12].

Imaginemos que Alice envia duas mensagens cifradas, cuja mensagem original é igual, n é igual, mas cujos valores de e são primos entre si. Dito por outras palavras, Alice envia

$$C_1 \equiv M^{e_1} \mod n$$

e

$$C_2 \equiv M^{e_2} \mod n,$$

tal que $mdc(e_1, e_2) = 1$.

Pelo Teorema 1.1, existem dois inteiros x e y tais que $e_1 x + e_2 y = 1$. O algoritmo de Euclides permite resolver esta equação.

Ora:

$$C_1^x C_2^y \equiv (M^{e_1})^x (M^{e_2})^y \equiv M \mod n.$$

Repare-se que o atacante não precisa de ter conhecimento de nenhum dos parâmetros secretos do RSA para descobrir a mensagem original, bastando, para isso, descobrir x e y e calcular $C_1^x C_2^y \mod n$.

Exemplo. *Suponhamos que o atacante capta as mensagens cifradas*

$$\begin{aligned} C_1 &= 13384933591224771001409391094599923488508716602967107269 \\ &\quad 22409294903802877621 2, \\ C_2 &= 17178188563886309608842164553692995830796192561975134178 \\ &\quad 17716871569640442354, \end{aligned}$$

onde foram utilizadas, respectivamente, as chaves públicas $(n,\, e_1)$ *e* $(n,\, e_2)$*, sendo*

$$\begin{aligned} n &= 26890251341470508178902701698748099296995699868232969883 \\ &\quad 5760593566623614349 09, \\ e_1 &= 5120583618019, \\ e_2 &= 2900392741007. \end{aligned}$$

O atacante procurará encontrar dois inteiros, x *e* y*, tais que* $e_1x + e_2y = 1$*. Introduzindo os seguintes dados no Maple,*

```
restart;
with(numtheory):
e1 := 5120583618019;
e2 := 2900392741007;
igcdex(e1, e2, 'x', 'y'):
x;
y;
```

facilmente o atacante ficará a saber que

$$\begin{aligned} x &= -56609328383, \\ y &= 99942602754, \end{aligned}$$

e descobrirá M *fazendo*

$$C_1^x \cdot C_2^y \mod n,$$

obtendo

$$M = 48327622032145019220393855611038764839294847628.$$

Apesar de, no Maple, precisarmos apenas de colocar o comando

```
igcdex(e1, e2, 'x', 'y'):
```

para que sejam determinados os inteiros x *e* y *tais que* $e_1x + e_2y = 1$*, o que o Maple faz, na realidade, é o procedimento descrito na página 17.*

5.2.3 Ataque do ponto fixo (*Fixed-point attack*, *Cyclic attack* ou *Superencryption attack*)

Este ataque, à semelhança do anterior, não ataca a factorização de n. Conhecido, em inglês por *fixed-point attack*, por *cyclic attack*, ou ainda por *superencryption attack*, foi descoberto por Simmons e Norris [32], pouco tempo depois do aparecimento do RSA.

Apesar de pouco provável, pode acontecer que, ao cifrarmos M, a mensagem cifrada seja M, ou seja,

$$M^e \equiv M \mod n.$$

Neste caso, dizemos que M é um *ponto fixo* da função que transforma a mensagem original na mensagem cifrada. Noutros casos, pode acontecer que a mensagem original fique descoberta quando ciframos a mensagem original k vezes, ou seja,

$$M^{e^k} \equiv M \mod n.$$

Definição. Seja $0 \leq M < n$. Se existe algum inteiro positivo k tal que

$$M^{e^k} \equiv M \mod n,$$

então dizemos que M é um *ponto fixo* de ordem k do RSA de chaves públicas (e, n).

Suponhamos que C é um ponto fixo de ordem k do RSA de chaves públicas (e, n). Sabemos, do capítulo 2.2.1, que $C \equiv M^e \mod n$, sendo M a mensagem a cifrar. Por definição, existe um inteiro positivo k tal que

$$C^{e^k} \equiv M^e \mod n,$$

isto é,

$$C^{e^{k-1} \cdot e} \equiv M^e \mod n.$$

Como e é invertível, módulo $\phi(n)$,

$$C^{e^{k-1}} \equiv M \mod n.$$

Qualquer mensagem M é um ponto fixo, já que, pelo teorema de Euler (teorema 1.6),

$$e^{\phi(\phi(n))} \equiv 1 \mod \phi(n),$$

pelo que existe um inteiro t tal que

$$e^{\phi(\phi(n))} = 1 + t\phi(n),$$

donde

$$M^{e^{\phi(\phi(n))}} \equiv M \cdot \left(M^{\phi(n)}\right)^t \equiv M \mod n.$$

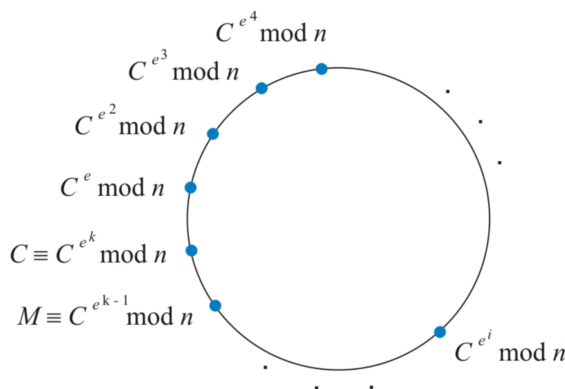
A ideia deste ataque consiste em calcular a sequência de números, módulo n:

$$C^e, \quad C^{e^2}, \quad C^{e^3}, \quad C^{e^4}, \quad \cdots, \quad C^{e^i}, \quad \cdots,$$

até encontrar um k, tal que $C^{e^k} \equiv C \mod n$. Encontrado esse k, obtemos

$$M \equiv C^{e^{k-1}} \mod n.$$

A partir do valor k, a função $f(i) = C^{e^i} \mod n$ torna-se cíclica, conforme ilustra a figura seguinte:



Repare-se que, com este ataque, conseguimos descobrir M, sem termos necessidade de conhecer d, nem p, nem q, nem $\phi(n)$.

Iremos ver como este ataque funciona, na prática. Consideremos o RSA de chaves públicas $n = 4661$ e $e = 19$. Consideremos ainda, $C = 3954$.

Utilizando o algoritmo seguinte no Maple:

```
with(numtheory):
n := 4661;
e := 19;
C := 3954;
for i from 1 to 500 do
  ataque := C &^ (e^i) mod n:
  print(C &^ (e^i) mod n):
  if C = C &^ (e^i) mod n then
    break:
```

```
  end if:
end do:
printf ("k = %d", i);
print  ();
printf ("M = %d", C &^ (e^(i-1)) mod n);
```

os valores de $C^{e^i} \mod n$ produzidos são

$$\begin{array}{cccccc} 119 & 2774 & 3423 & 2007 & 2302 & 1653 \\ 2243 & 2420 & 1889 & 414 & 3718 & 3954 \end{array}$$

Ora, $C \equiv C^{e^{12}} \mod 4661$, pelo que 3954 é um ponto fixo de ordem 12 do RSA de chave pública $(4661, 19)$. Então,

$$k = 12 \text{ e } M = 3718.$$

De facto,

$$3718^{19} \equiv 3954 \mod 4661.$$

Segundo Rivest [28], a quantidade de mensagens originais que podem ser descobertas, com êxito, através deste ataque é de tal forma insignificante, que o atacante preferirá tentar factorizar n. Este ataque só terá sentido se o inteiro positivo k for relativamente pequeno (por exemplo, menor que um milhão). Para mostrar que existe uma probabilidade reduzida de se descobrir a mensagem original M através deste ataque, Rivest utiliza o facto, já parcialmente referido em [29], de que $p-1$ e $q-1$ devem ter, cada um deles, um factor primo grande (digamos p' e q', respectivamente) e os restantes factores pequenos, assim como $p'-1$ e $q'-1$ devem ter, igualmente, cada um deles, um factor primo grande (digamos p'' e q'', respectivamente), devendo os restantes factores serem pequenos.

5.2.4 Expoente público pequeno

Este é um dos ataques mais conhecidos sobre expoentes públicos pequenos. Para tornar o processo de cifragem mais rápido (por exemplo, em *smart cards*, com evitáveis consumos de tempo e de energia), podemos cair na tentação de escolhermos um valor pequeno para e. Se escolhermos, por exemplo, $e = 3$, e se utilizarmos o algoritmo da exponenciação modular (3.2), para cifrarmos M, apenas necessitaríamos de calcular um quadrado e uma multiplicação modulares. Pelo teorema 1.3, facilmente verificamos que 3 é o menor valor que e pode tomar.

Imaginemos que o emissor envia a mesma mensagem para k destinatários, com chaves públicas distintas

$$(n_1,\ e),\ (n_2,\ e),\ \cdots,\ (n_k,\ e),$$

sendo $mdc(n_i,\ n_j) = 1$, com $i \neq j$ (caso contrário seria possível descobrir um factor próprio de algum dos n_i's e, portanto, descobrir M). Imaginemos, ainda, que o atacante consegue captar, no mínimo, $k \geq e$ mensagens cifradas $C_1, C_2, \cdots C_k$, sem que o emissor e o receptor se apercebam. O atacante consegue descobrir M, aplicando o Teorema Chinês do Resto (teorema 1.4) ao sistema

$$\begin{cases} M^e \equiv C_1 \mod n_1 \\ M^e \equiv C_2 \mod n_2 \\ \quad \vdots \\ M^e \equiv C_k \mod n_k \end{cases}, \tag{5.6}$$

obtendo

$$M^e \equiv \left(\sum_{i=1}^{k} C_i \cdot N_i \cdot (N_i^{-1} \mod n_i) \right) \mod N, \tag{5.7}$$

onde $N = \prod_{i=1}^{k} n_i$ e $N_i = \dfrac{N}{n_i}$, com $i \in \{1,\ 2,\ \cdots,\ k\}$.

Como $M^e < N$, basta calcular $\sqrt[e]{M^e} = M$.

Note-se que se os expoentes públicos e forem diferentes e pequenos, este ataque também funciona. Basta tomar $k = mmc(e_1,\ e_2,\ \cdots,\ e_k)$.

Exemplo. *Por uma questão de simplicidade, consideremos que o atacante captou as mensagens cifradas*

$$C_1 = 19234,\quad C_2 = 22192 \quad e \quad C_3 = 41631,$$

de chaves públicas $(n_i,\ e)$, *com* $i \in \{1,\ 2,\ 3\}$,

$$(33277,\ 3),\quad (35821,\ 3) \quad e \quad (45457,\ 3),$$

respectivamente.

O atacante irá resolver o sistema

$$\begin{cases} x \equiv 19234 \mod 33277 \\ x \equiv 22192 \mod 35821 \\ x \equiv 41631 \mod 45457 \end{cases} \tag{5.8}$$

que, segundo o teorema 1.4, tem uma única solução módulo $33277 \cdot 35821 \cdot 45457$.

Calculando N_i *e* $y_i \equiv N_i^{-1} \bmod n_i$ *para* $i \in \{1,\ 2,\ 3\}$ *obtemos:*

$$\begin{aligned} N_1 &= 35821 \cdot 45457 = 1628315197, \\ N_2 &= 33277 \cdot 45457 = 1512672589, \\ N_3 &= 33277 \cdot 35821 = 1192015417, \\ y_1 &\equiv \quad 562 \quad \bmod n_1, \\ y_2 &\equiv 29075 \quad \bmod n_2, \\ y_3 &\equiv \ 7793 \quad \bmod n_3. \end{aligned}$$

Substituindo os valores obtidos em (5.7), obtemos

$$M^3 \equiv 32648337956681 \quad \bmod 54185444810569,$$

donde $M = \sqrt[3]{32648337956681} = 31961$.

É de notar que este ataque só se torna eficaz nos casos em que e é pequeno.

Para prevenirmos este tipo de ataque, devemos utilizar valores de e suficientemente grandes ou acrescentar, no início e/ou no fim da mensagem original, um conjunto de bits pré-definidos e cifrar, utilizando expoentes públicos diferentes. Quando fazemos isto, dizemos que estamos a *camuflar* (*padding*) a mensagem original. Se w representar o número de bits de M (note-se que $w = [\log_2 M] + 1$), o emissor poderá, antes de cifrar M, aplicar a M a função polinomial

$$f_i(M) = i \cdot 2^w + M,$$

e enviar ao receptor R_i,

$$C_i \equiv (f_i(M))^{e_i} \quad \bmod n_i,$$

com $i \in \{1,\ 2,\ \cdots,\ k\}$.

Quando esta for a solução adoptada, o emissor torna público, não só os valores de n e de e, mas também a função $f_i \in \mathbb{Z}_{n_i}[x]$. Apesar de pensarmos que, com esta camuflagem, estamos seguros, Hastad provou o contrário (ver [15]), bastando, para isso, que o atacante consiga captar um número suficientemente grande de mensagens.

Defina-se

$$g_i(x) \equiv (f_i(x))^{e_i} - C_i \quad \bmod n_i.$$

Pretendemos encontrar M tal que

$$g_i(M) \equiv 0 \quad \bmod n_i, \quad i \in \{1,\ 2,\ \cdots,\ k\}.$$

Teorema 5.10 (Hastad [15]). *Sejam $n_1,\ n_2,\ \cdots,\ n_k$ primos entre si e tais que cada um dos n_i's é o produto de dois primos distintos.*

Seja $N_{min} = min\{n_1,\ n_2,\ \cdots,\ n_k\}$.

Sejam, ainda, $g_i \in \mathbb{Z}_{n_i}[x]$, com $1 \leq i \leq k$, k polinómios com grau, no máximo, d. Suponha-se que existe um único $M < N_{min}$ tal que

$$g_i(M) \equiv 0 \mod n_i,$$

para todo o $1 \leq i \leq k$.

Sendo $(n_i,\ g_i)$ público, com $1 \leq i \leq k$, se $k > d$, então é possível descobrir M.

A demonstração deste teorema utiliza o teorema de Coppersmith, um dos que tem mais impacto nos ataques em que e é pequeno:

Teorema 5.11 (Teorema de Coppersmith [5]). *Seja n um inteiro composto e $f \in \mathbb{Z}[x]$ um polinómio mónico, de grau d. Seja $X = n^{\frac{1}{d}-\epsilon}$, para algum $\epsilon \geq 0$. Conhecidos n e f, é possível encontrar, de forma eficiente, todos os inteiros $|x_0| < X$ tais que $f(x_0) \equiv 0 \mod n$. O tempo gasto a descobrir tais inteiros depende directamente do tempo que se demora a correr o algoritmo LLL sobre um reticulado de dimensão $O(w)$, onde $w = min\left(\frac{1}{\epsilon},\ \log_2 n\right)$.*

Iremos, agora, demonstrar o teorema de Hastad:

Demonstração:[teorema 5.10] Note-se que M deverá ser menor que N_{min}. Note-se, também, que o coeficiente guia (o coeficiente do termo de maior grau) de $g_i(x)$ deverá ser invertível módulo n_i, caso contrário o máximo divisor comum entre este coeficiente e n_i é distinto de 1, o que irá permitir encontrar um factor próprio de n_i (teorema 1.3). Podemos, então, supor, sem perda de generalidade, que o lado esquerdo da equação

$$g_i(x) \equiv 0 \mod n_i \tag{5.9}$$

é um polinómio mónico.

Da mesma forma, podemos supor, sem perda de generalidade, que, para cada i tal que $1 \leq i \leq k$, a equação (5.9) tem grau d. Se tal não acontecer, e supondo que

$$d = max\{grau(g_i(x)) :\ 1 \leq i \leq k\},$$

podemos multiplicar ambos os membros da equação (5.9), onde $grau(g_i(x)) < d$, por $x^{d-grau(g_i(x))}$.

Podemos, então escrever

$$g_i(x) \equiv \sum_{j=0}^{d} a_{ij}x^j \mod n_i,$$

onde $a_{id} = 1$.

Sejam $N = n_1 n_2 \cdots n_k$, $N_i = \dfrac{N}{n_i}$ e y_i tal que $y_i \equiv (N_i)^{-1} \mod n_i$.

Pelo Teorema Chinês do Resto (teorema 1.4), conseguimos construir o polinómio

$$G(x) \equiv \sum_{i=1}^{k} g_i(x) N_i y_i \mod N.$$

Repare-se que, sendo M a solução de $G(x)$,

$$G(M) \equiv \sum_{i=1}^{k} g_i(M) N_i y_i \equiv g_i(M) \equiv 0 \mod n_i.$$

Note-se que

$$\sum_{i=1}^{k} a_{id} N_i y_i \equiv a_{id} \equiv 1 \mod n_i.$$

Ora, $G(x)$ é um polinómio mónico de grau d, tal que $G(M) \equiv 0 \mod N$ e com $M < N_{min} \leq N^{\frac{1}{k}} < N^{\frac{1}{d}}$. Pelo teorema de Coppersmith (5.15), é possível encontrar M de forma eficiente. □

Segundo Hastad [15], o número de mensagens captadas pelo atacante deve ser

$$k > \frac{d(d+1)}{2}.$$

Exemplo. *Imaginemos que o atacante captou*

$$\begin{cases} (1 \cdot 2^{13} + x)^3 \equiv 3631 \mod 33277 \\ (2 \cdot 2^{13} + x)^3 \equiv 5782 \mod 35821 \\ (3 \cdot 2^{13} + x)^3 \equiv 33660 \mod 45457 \\ (4 \cdot 2^{13} + x)^3 \equiv 29758 \mod 48361 \\ (5 \cdot 2^{13} + x)^3 \equiv 38678 \mod 57067 \\ (6 \cdot 2^{13} + x)^3 \equiv 42445 \mod 64963 \\ (7 \cdot 2^{13} + x)^3 \equiv 62851 \mod 69373 \end{cases} \tag{5.10}$$

Usando a notação anteriormente definida, deste sistema retiramos que

$$\begin{aligned}
g_1(x) &\equiv 3719 + 742x + 24576x^2 + x^3 \equiv 0 \mod 33277,\\
g_2(x) &\equiv 6082 + 14467x + 13331x^2 + x^3 \equiv 0 \mod 35821,\\
g_3(x) &\equiv 31533 + 23308x + 28271x^2 + x^3 \equiv 0 \mod 45457,\\
g_4(x) &\equiv 32721 + 44345x + 1582x^2 + x^3 \equiv 0 \mod 48361,\\
g_5(x) &\equiv 36321 + 26601x + 8746x^2 + x^3 \equiv 0 \mod 57067,\\
g_6(x) &\equiv 57905 + 30291x + 17530x^2 + x^3 \equiv 0 \mod 64963,\\
g_7(x) &\equiv 8929 + 23662x + 33286x^2 + x^3 \equiv 0 \mod 69373.
\end{aligned}$$

Usando a notação usada no Teorema Chinês do Resto (5.7), temos que

$$\begin{aligned}
N &= 673937319143630624988753417665197,\\
N_1 &= 20252346039115023138767118961,\\
N_2 &= 18814028618509550961412395457,\\
N_3 &= 14825820426856823481284585821,\\
N_4 &= 13935553837671483736662877477,\\
N_5 &= 11809580302865590008038856391,\\
N_6 &= 10374171746126727906481434319,\\
N_7 &= 9714691870664820967649567089,\\
y_1 &= (N_1)^{-1} \mod n_1 = 2557 \mod 33277,\\
y_2 &= 10593 \mod 35821,\\
y_3 &= 23110 \mod 45457,\\
y_4 &= 5194 \mod 48361,\\
y_5 &= 11222 \mod 57067,\\
y_6 &= 23069 \mod 64963,\\
y_7 &= 31904 \mod 69373.
\end{aligned}$$

Aplicando o Teorema Chinês do Resto, obtemos

$$\begin{aligned}
G(x) = {} & 78206496350475732875454945470517\\
& + 288035621314135477760837979802 17\, x\\
& + 136811558418870554939794585590225\, x^2 + x^3 \mod N
\end{aligned}$$

Pelo teorema de Coppersmith, é possível encontrar todas as raízes, x_0*, de* $G(x)$ $\mod N$*, tais que* $|x_0|$ *é inferior a* $n^{\frac{1}{d}-\epsilon}$*, para algum* $\epsilon \geq 0$.

Apesar de Coppersmith ter apresentado um método para descobrir todas as raízes pequenas de $G(x)$, Howgrave-Graham mostrou que havia uma outra forma de o fazer, mais simples que o primeiro (ver [16]).

5.2.5 Expoente privado pequeno

Vimos, na secção anterior, que, ao escolhermos um valor pequeno para e no RSA, esta utilização torna as chaves secretas do RSA vulneráveis. Analogamente, as chaves secretas do RSA poderão ser descobertas quando d for pequeno, apesar de esta utilização tornar o processo de decifragem mais rápido. Wiener mostrou que se escolhermos um d pequeno, é fácil, para o atacante, quebrar o sigilo das chaves secretas.

Definição (Fracção contínua finita)**.** Uma fracção contínua finita (também, por vezes denominada fracção continuada finita) é uma expressão da forma

$$[a_0,\, a_1,\, a_2,\, \cdots,\, a_m] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_m}}}}$$

onde a_0 é um número real e os restantes a_i's, com $0 < i \leq m$, são reais positivos, em número finito.

Quando a_0 é um número inteiro e os restantes a_i's, com $0 < i \leq m$, são inteiros positivos, dizemos que se trata de uma *fracção contínua simples* (finita).

Teorema 5.12. *Sejam m um inteiro, a_0 um número real e $a_1, a_2, \cdots, a_m$ reais positivos.*

Para cada $k \leq m$, consideremos a fracção contínua $[a_0, a_1, a_2, \cdots, a_k]$ e consideremos as sucessões finitas $(P_k)_{k\leq n}$ e $(Q_k)_{k\leq n}$ definidas por:

$$P_{-1} = 1, \quad P_0 = a_0, \quad Q_{-1} = 0, \quad Q_0 = 1,$$

e

$$\begin{cases} P_{i+1} = a_{i+1}P_i + P_{i-1} \\ Q_{i+1} = a_{i+1}Q_i + Q_{i-1} \end{cases} \quad \textit{para todo o } \; 2 \leq i \leq k-1$$

Então, para todo o $k \leq n$, $\quad [a_0,\, a_1,\, a_2,\, \cdots,\, a_k] = \dfrac{P_k}{Q_k}$.

Demonstração: Ver [14]. □

Definição. (Convergentes) Chamamos convergentes às fracções $\dfrac{P_k}{Q_k}$ referidas no teorema anterior (5.12).

Teorema 5.13. *Todo o número racional pode ser escrito como uma fracção contínua simples.*

Demonstração: Ver [14] □

Teorema 5.14 (M. Wiener)**.** *Sejam $n = pq$, com $q < p < 2q$ e $d < \frac{1}{3}\sqrt[4]{n}$.*

Nestas condições, e sendo $(n,\, e)$ a chave pública do RSA, com

$$ed \equiv 1 \mod \phi(n),$$

consegue-se descobrir d.

Demonstração: Como $ed \equiv 1 \mod \phi(n)$, existe um inteiro k tal que

$$ed - k\phi(n) = 1.$$

Sabemos que $\phi(n) = n - p - q + 1$. Por outro lado, como $n = pq > q^2$, temos que $q < \sqrt{n}$.

Então,

$$\begin{aligned}\left|\frac{e}{n} - \frac{k}{d}\right| &= \left|\frac{ed - kn}{dn}\right| \\ &= \left|\frac{ed - k\phi(n) - kn + k\phi(n)}{dn}\right| \\ &= \left|\frac{1 - k(n - \phi(n))}{dn}\right|\end{aligned}$$

Como $0 < n - \phi(n) = p + q - 1 < 3q < 3\sqrt{n}$, então

$$\begin{aligned} \left|\frac{e}{n} - \frac{k}{d}\right| &= \left|\frac{1 - k(n - \phi(n))}{dn}\right| \\ &< \left|\frac{1 - 3k\sqrt{n}}{dn}\right| \\ &< \frac{3k\sqrt{n}}{dn} \\ &= \frac{3k}{d\sqrt{n}} \end{aligned}$$

Como $e < \phi(n)$ e $k\phi(n) < ed < d\phi(n)$, retiramos que

$$k < d < \frac{1}{3}\sqrt[4]{n},$$

donde

$$\begin{aligned} \left|\frac{e}{n} - \frac{k}{d}\right| &< \frac{3k}{d\sqrt{n}} \\ &= \frac{k}{d\sqrt[4]{n} \cdot \frac{1}{3}\sqrt[4]{n}} \\ &< \frac{\frac{1}{3}\sqrt[4]{n}}{d^2\sqrt[4]{n}} \\ &< \frac{1}{2d^2}. \end{aligned}$$

Por um teorema [17] clássico da teoria de aproximação por fracções contínuas[2], $\frac{k}{d}$ é um convergente de $\frac{e}{n}$. □

Apresentamos, a seguir, um possível algoritmo para o ataque do expoente privado pequeno:

Algoritmo 5.1.

input: *uma chave pública* (n, e)

a fracção contínua $\frac{e}{n} = [a_0, a_1, \cdots, a_m]$

output: *divisores não triviais de* n*:* p *e* q

Para i *de* 1 *até* m

Determina o $(i+1)^o$ *convergente de* $\frac{e}{n}$*:* $\frac{k_i}{d_i}$*;*

[2]Toda a fracção irredutível $\frac{a}{b}$ que satisfaz $\left|x - \frac{a}{b}\right| < \frac{1}{b^2}$ é um convergente de x.

Calcula $\phi_i(n) := \dfrac{ed_i - 1}{k_i}$*;*

Se $\phi_i(n)$ *for um inteiro e se* $x^2 - (n - \phi_i(n) + 1)\,x + n = 0$ *tiver solução*

$p :=$ *1ª solução;*

$q :=$ *2ª solução;*

$d := d_i$*;*

Quebra o ciclo;

Fim do ciclo;

Vejamos um exemplo:

Exemplo. *Consideremos* $n = 9684634573$ *e* $e = 6029932925$*.*

Os convergentes de $\frac{e}{n}$ *são:*

$$1,\ \frac{1}{2},\ \frac{2}{3},\ \frac{3}{5},\ \frac{5}{8},\ \frac{28}{45},\ \frac{33}{53},\ \frac{919}{1476},\ \frac{2790}{4481},\ \frac{3709}{5957},\ \frac{6499}{10438},\ \frac{23206}{37271},\ \frac{29705}{47709},\ \frac{82616}{132689},$$

$$\frac{2508185}{4028379},\ \frac{2590801}{4161068},\ \frac{5098986}{8189447},\ \frac{7689787}{12350515},\ \frac{12788773}{20539962},\ \frac{71633652}{115050325},\ \frac{156056077}{250640612},$$

$$\frac{227689729}{365690937},\ \frac{611435535}{982022486},\ \frac{839125264}{1347713423},\ \frac{1450560799}{2329735909},\ \frac{2289686063}{3677449332},\ \frac{6029932925}{9684634573}.$$

Suponhamos que $\frac{k}{d} = \frac{1}{2}$*.*

- $\phi(n) = 12059865849$*;*
- *a equação* $p^2 - (n - \phi(n) + 1)p + n = 0$ *não tem soluções inteiras (tem 2 soluções reais).*

Se $\frac{k}{d} = \frac{2}{3}$*:*

- $\phi(n) = 9044899387$*;*
- *a equação* $p^2 - (n - \phi(n) + 1)p + n = 0$ *não tem soluções inteiras (tem 2 soluções reais).*

Se $\frac{k}{d} = \frac{3}{5}$*:*

- $\phi(n) = 10049888208$*;*
- *a equação* $p^2 - (n - \phi(n) + 1)p + n = 0$ *não tem soluções inteiras (tem 2 soluções reais).*

Se $\dfrac{k}{d} = \dfrac{5}{8}$*,* $\phi(n)$ *não é inteiro, o que não pode ser.*

Se $\dfrac{k}{d} = \dfrac{28}{45}$*,* $\phi(n)$ *não é inteiro, o que também não pode ser.*

Se $\dfrac{k}{d} = \dfrac{33}{53}$*:*

- $\phi(n) = 9684437728$*;*
- *a equação* $p^2 - (n - \phi(n) + 1)p + n = 0$ *tem duas soluções inteiras:* 99989 *e* 96857*;*
- $p = 96857$ *e* $q = 99989$*.*

Podemos chegar à mesma conclusão correndo o algoritmo, em Maple, que se encontra em anexo (ver apêndice J):

```
1° convergente = 1
2° convergente = 1/2
3° convergente = 2/3
4° convergente = 3/5
5° convergente = 5/8
6° convergente = 28/45
7° convergente = 33/53
k = 33
d = 53
p = 99989
q = 96857
phi(n) = 9684437728
```

Vejamos uma outra forma de descobrir d: depois de obtermos os convergentes de $\dfrac{e}{n}$, pretende-se saber quando

$$a^{ed} \equiv a \mod n,$$

para um a arbitrário.

No exemplo apresentado, se escolhermos $a = 2$,

- se $d = 2$, $\quad 2^{ed} \equiv 5035470927 \not\equiv 2 \mod 9684634573$;
- se $d = 3$, $\quad 2^{ed} \equiv 2580341517 \not\equiv 2 \mod 9684634573$;
- se $d = 5$, $\quad 2^{ed} \equiv 8834504314 \not\equiv 2 \mod 9684634573$;
- se $d = 8$, $\quad 2^{ed} \equiv 3110358293 \not\equiv 2 \mod 9684634573$;
- se $d = 45$, $\quad 2^{ed} \equiv 501087853 \not\equiv 2 \mod 9684634573$;

- se $d = 53$, $2^{ed} \equiv 2 \mod 9684634573$,

donde se confirma que $d = 53$.

5.2.6 Ataque dos tempos de Kocher (*Kocher's timing attack*)

Em 1995, Paul Kocher [19], ainda estudante universitário, surpreendeu o mundo da Criptologia com a divulgação de um ataque inovador: conseguia conhecer d, sem ter que factorizar n. Este ataque nada tem a ver com o algoritmo do RSA, em si, mas com o tempo gasto, pelo hardware, na implementação do RSA.

Kocher [19] verificou que, medindo o tempo gasto pelo hardware (por exemplo, um computador ou um *smartcard*) a calcular $C^d \mod n$, para vários valores de C, é possível conhecer d, com recurso a algumas noções de probabilidades e estatística.

Iremos descrever a ideia subjacente a este ataque.

O atacante, conhecedor de n e de C_i, com $0 < i \leq k$, e sabendo o modo como a vítima/decifrador decifra a mensagem recebida, observa o tempo que a vítima gasta a calcular M_i, isto é, regista

$$\begin{aligned} T_1 &= \text{tempo gasto a calcular } M_1 \equiv C_1^d \mod n \\ T_2 &= \text{tempo gasto a calcular } M_2 \equiv C_2^d \mod n \\ &\vdots \\ T_k &= \text{tempo gasto a calcular } M_k \equiv C_k^d \mod n. \end{aligned}$$

Estas medições de tempo que o atacante obtém poderão ser recolhidas remotamente, acedendo a protocolos interactivos.

É importante salientar que, para este ataque funcionar, a chave utilizada para decifrar a mensagem deve ser sempre a mesma.

Suponhamos que a vítima calcula $M = C^d \mod n$ (d tem w bits) da forma descrita no capítulo 3.2 (potenciação modular). Sendo

$$d = d_{w-1} \cdots d_2 d_1 d_0,$$

no sistema binário, um algoritmo possível é:

Algoritmo 5.2.

```
M := 1;
for k from 0 to w-1 do
  if d_{k} = 1 then
    M := M * C mod n;
  end if;
  C := C^{2} mod n;
end do;
print (M);
```

O atacante sabe que $d_0 = 1$, já que $mdc(d,\ (p-1)(q-1)) = 1$ (já referido no capítulo 2.2.1), portanto $2 \not| d$. Logo, $d_0 = 1$.

Seja $t_{j_{d_0}} = t_{j_1}$ o tempo (real) dispendido na passagem do ciclo *for* do algoritmo 5.2, para $d_0 = 1$. Obviamente, d_1 será 0 ou 1.

Seja

$$t_{j_{01}} = t_{j_1} + t_{j_{(d_1=0)}}$$

o tempo para calcular $C_j^{01} \mod n$, sendo $t_{j_{(d_1=0)}}$ o tempo que o computador demora a passar no ciclo *for* se $d_1 = 0$.

Seja

$$t_{j_{11}} = t_{j_1} + \tilde{t}_{j_{(d_1=1)}}$$

o tempo para calcular $C_j^{11} \mod n$, sendo $\tilde{t}_{j_{(d_1=1)}}$ o tempo que o computador demora a passar no ciclo *for* se $d_1 = 1$.

Consideremos que o atacante conhece os últimos l bits de d, isto é, conhece

$$d_{l-1} \cdots d_1 d_0.$$

Para descobrir d_l, que pode ser 0 ou 1, o atacante regista

$$t_{j_{0d_{l-1}\cdots d_0}} = t_{j_{d_0}} + \cdots + t_{j_{d_{l-1}}} + t_{j_{(d_l=0)}}$$

e

$$t_{j_{1d_{l-1}\cdots d_0}} = t_{j_{d_0}} + \cdots + t_{j_{d_{l-1}}} + \tilde{t}_{j_{(d_l=1)}},$$

sendo, respectivamente, o tempo que se demora a calcular

$$C_j^{0d_{l-1}\cdots d_0} \mod n$$

ou

$$C_j^{1d_{l-1}\cdots d_0} \mod n,$$

e sendo $t_{j_{(d_l=0)}}$ e $\tilde{t}_{j_{(d_l=1)}}$ o tempo que o computador demora a passar no ciclo *for* se $d_l = 0$ e se $d_l = 1$, respectivamente.

Estamos, obviamente, a ignorar os tempos gastos na comparação $d_k = 1$, no teste *if* do algoritmo, no incremento de k e na impressão, no ecrã, de M.

A maior descoberta de Kocher reside no facto de que a variância de

$$T_j - t_{j_{d_l \cdots d_0}}$$

é mais pequena no caso em que o bit d_l está correcto do que no caso em que o bit d_l está incorrecto. Sendo assim, o atacante calcula a média de todos os $T_j - t_{j_{d_l \cdots d_0}}$, nos casos em que $d_l = 0$ e nos casos em que $d_l = 1$, ou seja, calcula

$$E(T_j - t_{j_{0d_{l-1} \cdots d_0}}) = \frac{(T_1 - t_{1_{0d_{l-1} \cdots d_0}}) + (T_2 - t_{2_{0d_{l-1} \cdots d_0}}) + \cdots + (T_k - t_{k_{0d_{l-1} \cdots d_0}})}{k},$$

calcula

$$E(T_j - t_{j_{1d_{l-1} \cdots d_0}}) = \frac{(T_1 - t_{1_{1d_{l-1} \cdots d_0}}) + (T_2 - t_{2_{1d_{l-1} \cdots d_0}}) + \cdots + (T_k - t_{k_{1d_{l-1} \cdots d_0}})}{k},$$

e compara as respectivas variâncias, optando pelo bit d_l onde a variância é menor.

Relembremos que a variância (**var**) é determinada da seguinte forma:

$$\mathbf{var}\left(\{T_j - t_{j_{d_l \cdots d_0}}\}_{j=1}^{k}\right) = \frac{1}{k}\sum_{j=1}^{k}\left((T_j - t_{j_{d_l \cdots d_0}}) - E(T_j - t_{j_{d_l \cdots d_0}})\right)^2.$$

Recorrendo a este ataque sucessivamente, o atacante consegue descobrir todos os bits de d.

Kocher refere (ver [19]) que, para evitar este tipo de ataque, deve-se:

- gerar aleatoriamente um número inteiro positivo r, que será secreto, inferior a n e que seja primo com n;
- calcular $C' \equiv C \cdot r^e \mod n$;
- calcular $M' \equiv (C')^d \mod n$;
- calcular $M \equiv M' \cdot r^{-1} \mod n$.

Este procedimento está correcto, já que

$$\begin{aligned} M' \cdot r^{-1} \mod n &\equiv (C')^d \cdot r^{-1} \mod n \\ &\equiv (C \cdot r^e)^d \cdot r^{-1} \mod n \\ &\equiv C^d \cdot r^{ed} \cdot r^{-1} \mod n \\ &\equiv M \mod n. \end{aligned}$$

Se se utilizar sempre o mesmo valor de r, há o risco de o atacante descobri-lo através deste ataque, pelo que Kocher aconselha a que se utilize diferentes valores de r em cada mensagem.

5.2.7 Ataque com parte da chave privada exposta (*Partial private key exposure attack*)

Suponhamos que o atacante conhece os $\lceil n/4 \rceil$ bits menos significativos de d. Conseguirá o atacante descobrir todos os bits de d?

Em 1998, Boneh, Durfee e Frankel [6] mostraram que, sendo $n = pq$ um número com β bits, se e for pequeno, e se, pelo menos, os $\lceil \beta/4 \rceil$ bits menos significativos de d forem conhecidos, então o atacante consegue descobrir d em tempo polinomial, em n e e. Segundo estes autores, alguns ataques, como o ataque do tempo de Kocher, podem não ser eficazes na descoberta de todos os bits de d.

Se o atacante conhece os $\lceil \beta/4 \rceil$ bits menos significativos de d (chamemos-lhes d_0), então sabe que

$$d \equiv d_0 \mod 2^{\lceil \beta/4 \rceil}.$$

Sabemos, do capítulo 2.2.1, que $ed \equiv 1 \mod \phi(n)$. Por definição, existe um inteiro k tal que

$$ed - k\phi(n) = 1,$$

ou seja,

$$ed - k(n - p - q + 1) = 1. \tag{5.11}$$

Reduzindo a equação (5.11), módulo $2^{\lceil \beta/4 \rceil}$, fazendo $q = n/p$ e multiplicando ambos os membros por p, obtemos

$$ed_0 p - kp(n - p + 1) + kn \equiv p \mod 2^{\lceil \beta/4 \rceil},$$

ou, de forma equivalente,

$$kp^2 + (ed_0 - k(n+1) - 1)p + kn \equiv 0 \mod 2^{\lceil \beta/4 \rceil}. \tag{5.12}$$

Como $\phi(n) \geq d$, então $0 < k \leq e$. Por este motivo, para cada valor possível k' de k, do conjunto $\{1, 2, \cdots, e\}$, o atacante conseguirá resolver a equação modular (5.12), em ordem a p. Note-se que Boneh, Durfee e Frankel referem (ver [6]) que se considera que e é "pequeno" quando for exequível resolver a equação 5.12 para cada um dos valores de $0 < k \leq e$.

Nesta etapa, o atacante deverá conhecer valores candidatos a $p_0 \equiv p \mod 2^{\lceil \beta/4 \rceil}$.

Para sabermos como o atacante procede com este ataque, de modo a descobrir todos os bits de d, precisamos do teorema e do corolário que se seguem.

Teorema 5.15 (Teorema de Coppersmith [10]). *Seja $f(x,\, y)$ um polinómio com duas variáveis sobre $\mathbb{Z}$, cujo termo de maior grau em cada uma das variáveis é δ. Considere-se que os coeficientes de f são primos entre si. Sejam, ainda, $(X,\, Y)$ os limites superiores das abcissas e das ordenadas das soluções (x_0, y_0), de f.*

Defina-se $\tilde{f}(x,\, y) := f(Xx,\, Yy)$ e seja D o coeficiente de maior valor absoluto de $\tilde{f}(x,\, y)$.

Se $XY < D^{\frac{2}{3\delta}}$, então podemos encontrar, num tempo polinomial em $\log D$ e 2^δ, todos os pares de números inteiros $(x_0,\, y_0)$ tais que $f(x_0,\, y_0) = 0$, $|x_0| < X$ e $|y_0| < Y$.

Se o leitor tiver curiosidade em conhecer a demonstração deste teorema, poderá consultar [10].

Corolário 5.16. *Seja $n = pq$, um número com β bits, e tal que p e q são primos e $p \neq q$. Considere-se que $r \geq 2^{\lceil \beta/4 \rceil}$ e $p_0 := p \mod r$ são conhecidos. Então, é possível factorizar n em tempo polinomial.*

Demonstração: (ver [6]) Consideremos que

$$\frac{n^{\frac{1}{2}}}{2} < p < q < 2 \cdot n^{\frac{1}{2}}.$$

Se p e r não são primos entre si, a demonstração é trivial. Consideremos, então que $mdc(p,\, r) = 1$. Então, $mdc(p_0,\, r) = 1$.

A partir de

$$p_0 := p \mod r,$$

podemos encontrar

$$q_0 := q \equiv \frac{n}{p_0} \mod r.$$

Procuremos a solução $(x_0,\, y_0)$ de

$$f(x,\, y) = (rx + p_0)\,(ry + q_0) - n.$$

Como $p < q < 2 \cdot n^{\frac{1}{2}} < 2 \cdot (2^\beta)^{\frac{1}{2}} = 2^{\frac{\beta}{2}+1}$,

$$(rx_0 + p_0) \cdot (ry_0 + q_0) = pq < 2^{\frac{\beta}{2}+1} \cdot 2^{\frac{\beta}{2}+1}.$$

Como

$$rx_0 + p_0 = p < 2^{\frac{\beta}{2}+1},$$

então

$$rx_0 < 2^{\frac{\beta}{2}+1}$$

e, portanto,

$$x_0 < X = \frac{2^{\frac{\beta}{2}+1}}{r}.$$

Analogamente, concluímos que

$$y_0 < Y = \frac{2^{\frac{\beta}{2}+1}}{r}.$$

Como o máximo divisor comum entre os coeficientes de $f(x,\ y)$ é r, e, para aplicarmos o teorema de Coppersmith 5.15 é necessário que os coeficientes sejam primos entre si, precisamos definir

$$g(x,\ y) = \frac{f(x,\ y)}{r}.$$

Defina-se, ainda,

$$\begin{aligned}\tilde{g}(x,\ y) &= g(Xx, Yy)\\ &= rXYxy - q_0Xx - p_0Yy + \frac{p_0q_0 - n}{r}\end{aligned}$$

O coeficiente de $\tilde{g}(x,\ y)$ com maior valor absoluto é $D = \dfrac{2^{\beta+2}}{r}$.

Para estarmos em condições de aplicar o teorema de Coppersmith 5.15, precisamos que

$$XY = \frac{2^{\beta+2}}{r^2} < \left(\frac{2^{\beta+2}}{r}\right)^{\frac{2}{3}},$$

o que irá acontecer se

$$2^{\frac{1}{3}(n+2)} < r^{\frac{4}{3}},$$

isto é, se

$$r > 2^{\frac{n+2}{4}}.$$

Então, pelo teorema de Coppersmith (5.15), para $r > 2^{\frac{n+2}{4}}$, podemos encontrar, num tempo polinomial em $\log D$ e 2^{δ}, todos os pares de números inteiros $(x_0,\ y_0)$ tais que $f(x_0, y_0) = 0$, $|x_0| < X$ e $|y_0| < Y$, sendo, portanto, possível factorizar n em tempo polinomial. □

Um possível algoritmo para o ataque com parte da chave privada exposta pode ser:

Algoritmo 5.3.

input: *uma chave pública* $(n,\ e)$, *com* $n = \beta$ *bits*

$d_0 = \lceil \beta/4 \rceil$ *bits menos significativos de* d

output: *o expoente privado d*

Calcular $e \cdot d0 \equiv 1 + k(n-s+1) \mod 2^{n/4}$.

Atribuir a k *um valor possível, com* $1 \leq k \leq e$.

Resolver a equação $kp^2 + (ed_0 - kn - k - 1)p + kn = 0 \mod 2^{\lceil \beta/4 \rceil}$, *para encontrar* $p_0 \equiv p \mod 2^{\lceil \beta/4 \rceil}$.

Se se encontrar $p_0 \equiv p \mod 2^{\lceil \beta/4 \rceil}$:

- *encontrar* $q_0 \equiv q \mod 2^{\lceil \beta/4 \rceil}$ *tal que* $p_0 q_0 \equiv n \mod 2^{\lceil \beta/4 \rceil}$.
- *encontrar* x *e* y *no polinómio* $p(x,\, y) = (rx + p_0) \cdot (ry + q_0) - n$, *onde* $r = 2^{\lceil \beta/4 \rceil}$, *tal que* $p(x,\, y) = 0$.

Se $p(x,\, y) = 0$ *tiver solução:*

- *determinar* $p = rx + p_0$ *e* $q = ry + q_0$.
- *calcular* $\phi(n) = (p-1) \cdot (q-1)$.
- *encontrar* d *tal que* $ed \equiv 1 \mod \phi(n)$.

Em anexo (apêndice K) podemos encontrar um algoritmo deste ataque, em Maple, que poderá ser usado para o exemplo que se segue.

Exemplo. *Consideremos* $n = 177299 = (101011010010010011)_2$, *um número com* 18 *bits. Consideremos, ainda, que* $e = 11$ *e que se conhecem os últimos 5 bits de* d, $d_0 = (10011)_2 = 19$.

Sabemos que

$$edp - kp(n-p+1) + kn \equiv 1 \mod 2^5,$$

isto é,

$$11 \cdot 19 \cdot p - kp(177299 - p + 1) + k \cdot 177299 \equiv 1 \mod 32,$$

ou seja,

$$kp^2 + (16 + 12k)p + 19k \equiv 0 \mod 32.$$

Se fizermos $k = 1$, *concluímos que*

$$p_0 \equiv 11 \mod 32, \quad p_0 \equiv 9 \mod 32, \quad p_0 \equiv 27 \mod 32 \quad ou \quad p_0 \equiv 25 \mod 32.$$

Se, $p_0 \equiv 11 \bmod 32$, *então* $11 \cdot q_0 \equiv n \mod 32$ *e, portanto,* $q_0 \equiv 25 \mod 32$.

Ora,

$$(r \cdot x + p_0) \cdot (r \cdot y + q_0) - n = 0, \tag{5.13}$$

isto é,

$$(32 \cdot x + 11) \cdot (32 \cdot y + 25) - 177299 = 0. \tag{5.14}$$

Facilmente podemos verificar, no Maple, que (3, 51) *é solução da equação (5.14), pelo que podemos concluir que*

$$p = 32 \cdot 3 + 11 = 107,$$

$$q = 32 \cdot 51 + 25 = 1657,$$

$$\phi(n) = 175536$$

e

$$d \equiv e^{-1} \equiv 95747 \mod \phi(n).$$

Apêndice A

Algoritmo para o Maple, com o intuito de cifrar / decifrar mensagens:

Exemplo.
```
with(numtheory):
with(StringTools):

p := readstat ("Introduza um número inteiro para 'p', seguido do símbolo ';': ");
q := readstat ("Introduza um número inteiro para 'q', seguido do símbolo ';': ");
n := p * q;
fiDeN := (p-1)*(q-1);
e := readstat ("Introduza um número inteiro para 'e', seguido do símbolo ';': ");
d := e^(-1) mod fiDeN;

msg := readstat ("Introduza a mensagem a cifrar, entre aspas, seguido de ';': ");
m1 := " ":
length(msg):
tamanhoAlf := 256;

# Tamanho de cada bloco para cifrar
for i from 1 do
  if tamanhoAlf^i < n and n < tamanhoAlf^(i+1) then
    printf ("Cada bloco é constituído por %d símbolos.", i):
    break:
  end if:
end do:
print():

# Verifica se o tamanho da mensagem é múltiplo de i
j := length(msg) mod i:

# Caso a mensagem não tenha tamanho múltiplo de i, acrescenta os espaços
em branco necessários, de modo a sê-lo
if length(msg) mod i <> 0 then
   for k from 1 to i-j do
```

```
      msg := cat(msg, m1):
   end do:
end if:

numBlocos := length(msg) / i:

# Divide a mensagem em blocos de tamanho i e calcula o numeral de cada bloco
for k from 1 to numBlocos do
   bloco[k]  := substring(msg, k*i-(i-1)..k*i);
   blocos[k] := 0:
   for l from 1 to i do
      blocos[k] := blocos[k] * tamanhoAlf + Ord(substring(bloco[k], l..l)):
   end do;
end do:

# Cifragem
printf ("Mensagem cifrada: "):
for k from 1 to numBlocos do
   blocos[k] := blocos[k]&^e mod n:
   printf ("%d ", blocos[k]):
end do:
print();

# Escrever a mensagem cifrada no alfabeto usual
for k from 1 to numBlocos do
  msgCifrada[k] := Char(blocos[k] mod tamanhoAlf):
  blocos[k] := (blocos[k] - (blocos[k] mod tamanhoAlf)) / tamanhoAlf:
  for l from 1 while blocos[k] >= tamanhoAlf do
    msgCifrada[k] := cat(Char(blocos[k] mod tamanhoAlf), msgCifrada[k]):
    blocos[k] := (blocos[k] - (blocos[k] mod tamanhoAlf)) / tamanhoAlf:
  end do:
  msgCifrada[k] := cat(Char(blocos[k] mod tamanhoAlf), msgCifrada[k]):
end do:

# Escreve no ecrã a mensagem cifrada
printf ("A Alice envia a seguinte mensagem (cifrada) a Bob:"):
colar := ‘‘:
for k from 1 to numBlocos do
  if length(msgCifrada[k]) = i then
    msgCifrada[k] := cat(Char(0), msgCifrada[k]):
  end if:
  colar := cat(cat(colar, msgCifrada[k]), Char(32)):
end do:
```

```
colar := substring(colar, 1..length(colar)-1):
print (colar);

# Decifragem
for k from 1 to numBlocos do
  blocos[k] := 0:
  if length(msgCifrada[k]) = i then
    for l from 1 to i do
      blocos[k] := blocos[k] * tamanhoAlf + Ord(substring(msgCifrada[k], l..l));
    end do;
  else
    for l from 1 to i+1 do
      blocos[k] := blocos[k] * tamanhoAlf + Ord(substring(msgCifrada[k], l..l));
    end do;
  end if:
end do:

# Mensagem Decifrada
printf ("Mensagem decifrada: "):
for k from 1 to numBlocos do
   blocos[k] := blocos[k]&^d mod n:
   printf ("%d ", blocos[k]):
end do:
print();

# Escrever a mensagem decifrada no alfabeto usual
for k from 1 to numBlocos do
  msgDecifrada[k] := Char(blocos[k] mod tamanhoAlf):
  blocos[k] := (blocos[k] - (blocos[k] mod tamanhoAlf)) / tamanhoAlf:
  for l from 1 while blocos[k] >= tamanhoAlf do
    msgDecifrada[k] := cat(Char(blocos[k] mod tamanhoAlf), msgDecifrada[k]):
    blocos[k] := (blocos[k] - (blocos[k] mod tamanhoAlf)) / tamanhoAlf:
  end do:
  msgDecifrada[k] := cat(Char(blocos[k] mod tamanhoAlf), msgDecifrada[k]):
end do:

# Escreve no ecrã a mensagem decifrada
printf ("Bob lê a seguinte mensagem (decifrada):"):
colar := ‘‘:
for k from 1 to numBlocos do
  if length(msgDecifrada[k]) = i then
    msgDecifrada[k] := cat(Char(0), msgDecifrada[k]):
  end if:
```

```
  colar := cat(colar, msgDecifrada[k]):
end do:
print (colar);
```

Apêndice B

Tabela ASCII

BINÁRIO	DECIMAL	GLIFO	DESCRIÇÃO
0000 0000	00		carácter nulo
0000 0001	01	□	início do cabeçalho
0000 0010	02	□	início do texto
0000 0011	03	□	fim do texto
0000 0100	04	□	fim de transmissão
0000 0101	05	□	interroga identidade do terminal
0000 0110	06	□	confirmação
0000 0111	07	□	campainha
0000 1000	08	□	retrocesso
0000 1001	09		tabulação horizontal
0000 1010	10		avanço de linha/nova linha
0000 1011	11	□	tabulação vertical
0000 1100	12	□	avanço de página/nova pág.
0000 1101	13		retorno do carro (enter)
0000 1110	14	□	saída do *shift* (passa a usar caracteres de baixo da tecla – minúsculas, etc.)
0000 1111	15	□	entrada do *shift* (passa a usar caracteres de cima da tecla – maiúsculas, caracteres especiais, etc.)
0001 0000	16	□	saída de ligação de dados
0001 0001	17	□	controlo do dispositivo 1
0001 0010	18	□	controlo do dispositivo 2
0001 0011	19	□	controlo do dispositivo 3
0001 0100	20	□	controlo do dispositivo 4
0001 0101	21	□	não confirmação
0001 0110	22	□	sincronismo em estado inactivo
0001 0111	23	□	fim do bloco de transmissão
0001 1000	24	□	*cancel*
0001 1001	25	□	final de suporte
0001 1010	26	□	substituir
0001 1011	27	□	*escape*
0001 1100	28	□	separação de ficheiros
0001 1101	29	□	separação de grupos
0001 1110	30	□	separação de registos
0001 1111	31	□	separação de unidades
0111 1111	127	□	*delete*

BINÁRIO	DECIMAL	GLIFO
0010 0000	32	ESPAÇO
0010 0001	33	!
0010 0010	34	"
0010 0011	35	#
0010 0100	36	$
0010 0101	37	%
0010 0110	38	&
0010 0111	39	'
0010 1000	40	(
0010 1001	41	)
0010 1010	42	*
0010 1011	43	+
0010 1100	44	,
0010 1101	45	-
0010 1110	46	.
0010 1111	47	/
0011 0000	48	0
0011 0001	49	1
0011 0010	50	2
0011 0011	51	3
0011 0100	52	4
0011 0101	53	5
0011 0110	54	6
0011 0111	55	7
0011 1000	56	8
0011 1001	57	9
0011 1010	58	:
0011 1011	59	;
0011 1100	60	<
0011 1101	61	=
0011 1110	62	>
0011 1111	63	?

BINÁRIO	DECIMAL	GLIFO
0100 0000	64	@
0100 0001	65	A
0100 0010	66	B
0100 0011	67	C
0100 0100	68	D
0100 0101	69	E
0100 0110	70	F
0100 0111	71	G
0100 1000	72	H
0100 1001	73	I
0100 1010	74	J
0100 1011	75	K
0100 1100	76	L
0100 1101	77	M
0100 1110	78	N
0100 1111	79	O
0101 0000	80	P
0101 0001	81	Q
0101 0010	82	R
0101 0011	83	S
0101 0100	84	T
0101 0101	85	U
0101 0110	86	V
0101 0111	87	W
0101 1000	88	X
0101 1001	89	Y
0101 1010	90	Z
0101 1011	91	[
0101 1100	92	\
0101 1101	93	]
0101 1110	94	^
0101 1111	95	_

BINÁRIO	DECIMAL	GLIFO
0110 0000	96	`
0110 0001	97	a
0110 0010	98	b
0110 0011	99	c
0110 0100	100	d
0110 0101	101	e
0110 0110	102	f
0110 0111	103	g
0110 1000	104	h
0110 1001	105	i
0110 1010	106	j
0110 1011	107	k
0110 1100	108	l
0110 1101	109	m
0110 1110	110	n
0110 1111	111	o
0111 0000	112	p
0111 0001	113	q
0111 0010	114	r
0111 0011	115	s
0111 0100	116	t
0111 0101	117	u
0111 0110	118	v
0111 0111	119	w
0111 1000	120	x
0111 1001	121	y
0111 1010	122	z
0111 1011	123	{
0111 1100	124	|
0111 1101	125	}
0111 1110	126	~
1000 0000	128	€

BINÁRIO	DECIMAL	GLIFO
1000 0001	129	□
1000 0010	130	‚
1000 0011	131	ƒ
1000 0100	132	„
1000 0101	133	…
1000 0110	134	†
1000 0111	135	‡
1000 1000	136	ˆ
1000 1001	137	‰
1000 1010	138	Š
1000 1011	139	‹
1000 1100	140	Œ
1000 1101	141	□
1000 1110	142	Ž
1000 1111	143	□
1001 0000	144	□
1001 0001	145	‘
1001 0010	146	’
1001 0011	147	“
1001 0100	148	”
1001 0101	149	•
1001 0110	150	–
1001 0111	151	—
1001 1000	152	˜
1001 1001	153	™
1001 1010	154	Š
1001 1011	155	›
1001 1100	156	œ
1001 1101	157	□
1001 1110	158	ž
1001 1111	159	Ÿ
1010 0000	160	

BINÁRIO	DECIMAL	GLIFO
1010 0001	161	¡
1010 0010	162	¢
1010 0011	163	£
1010 0100	164	¤
1010 0101	165	¥
1010 0110	166	¦
1010 0111	167	§
1010 1000	168	¨
1010 1001	169	©
1010 1010	170	ª
1010 1011	171	«
1010 1100	172	¬
1010 1101	173	-
1010 1110	174	®
1010 1111	175	¯
1100 1000	176	°
1100 1001	177	±
1100 1010	178	²
1100 1011	179	³
1100 1100	180	´
1100 1101	181	µ
1100 1110	182	¶
1100 1111	183	·
1101 0000	184	¸

BINÁRIO	DECIMAL	GLIFO
1101 0001	185	¹
1101 0010	186	º
1101 0011	187	»
1101 0100	188	¼
1101 0101	189	½
1101 0110	190	¾
1101 0111	191	¿
1101 1000	192	À
1101 1001	193	Á
1101 1010	194	Â
1101 1011	195	Ã
1101 1100	196	Ä
1101 1101	197	Å
1101 1110	198	Æ
1101 1111	199	Ç
1100 1000	200	È
1100 1001	201	É
1100 1010	202	Ê
1100 1011	203	Ë
1100 1100	204	Ì
1100 1101	205	Í
1100 1110	206	Î
1100 1111	207	Ï
1101 0000	208	Ð

BINÁRIO	DECIMAL	GLIFO
1101 0001	209	Ñ
1101 0010	210	Ò
1101 0011	211	Ó
1101 0100	212	Ô
1101 0101	213	Õ
1101 0110	214	Ö
1101 0111	215	×
1101 1000	216	Ø
1101 1001	217	Ù
1101 1010	218	Ú
1101 1011	219	Û
1101 1100	220	Ü
1101 1101	221	Ý
1101 1110	222	Þ
1101 1111	223	ß
1110 0000	224	à
1110 0001	225	á
1110 0010	226	â
1110 0011	227	ã
1110 0100	228	ä
1110 0101	229	å
1110 0110	230	æ
1110 0111	231	ç
1110 1000	232	è

BINÁRIO	DECIMAL	GLIFO
1110 1001	233	é
1110 1010	234	ê
1110 1011	235	ë
1110 1100	236	ì
1110 1101	237	í
1110 1110	238	î
1110 1111	239	ï
1111 0000	240	ð
1111 0001	241	ñ
1111 0010	242	ò
1111 0011	243	ó
1111 0100	244	ô
1111 0101	245	õ
1111 0110	246	ö
1111 0111	247	÷
1111 1000	248	ø
1111 1001	249	ù
1111 1010	250	ú
1111 1011	251	û
1111 1100	252	ü
1111 1101	253	ý
1111 1110	254	þ
1111 1111	255	ÿ

Apêndice C

Teste de Fermat (algoritmo)

Exemplo. `with(numtheory):`

```
n := readstat("Insira um inteiro positivo para n, maior que 1, seguido de ';': "):
if n <= 10000 then
  if n=2 then
    printf("O número %d é primo.", n):
  else
    a := 2;
    if igcd(a,n)<> 1 then
      for i from 1 while igcd(a,n) <> 1 do
        a := nextprime(a):
      end do:
    end if:
    if a^(n-1) mod n <> 1 then
      printf ("O número %d é composto.", n):
    else
      for j from 1 while a <= n do
        a := nextprime(a):
        for i from 1 while igcd(a,n) <> 1 do
          a := nextprime(a):
        end do:
        if a^(n-1) mod n <> 1 then
          printf ("O número %d é composto.", n):
        end if:
      end do:
      if a^(n-1) mod n = 1 then
        printf ("O número %d é pseudoprimo para todas as bases primas
                 e primas com %d até %d.", n, n, n):
      end if:
    end if:
  end if:
```

```
else
  a := RandomTools[Generate](integer(range=2..n-1));
  if igcd(a,n)<>1 then
    for i from 1 while igcd(a,n) <> 1 do
      a := RandomTools[Generate](integer(range=2..n-1));
    end do:
  end if:
  if a^(n-1) mod n <> 1 then
     printf ("O número %d é composto.", n):
  else
    for j from 1 to 50 do
      a := RandomTools[Generate](integer(range=2..n-1));
      for i from 1 while igcd(a,n) <> 1 do
        a := RandomTools[Generate](integer(range=2..n-1));
      end do:
      if a^(n-1) mod n <> 1 then
        printf ("O número %d é composto.", n):
        break:
      else
        printf ("O número %d é pseudoprimo para a base %d.", n, a):
        print():
      end if:
    end do:
  end if:
end if:
```

Apêndice D

Teste de Miller-Rabin (algoritmo)

Exemplo.

```
n1 := 0:
for i while type(n1, integer) = false or n1 <= 0 or n1 mod 2 = 0 do
  n1     := readstat("Insira um número inteiro positivo e ímpar,
            seguido de ';': "):
end do;
d      := n1 - 1:
a      :=  1:
cont   :=  0:
l      :=  0:
limSup := 10:
primo  :=  0:

# Escrever n - 1 na forma 2^s * d
for s from 0 while d mod 2 = 0 do
  d := d / 2;
end do:
printf("%d - 1 = 2^%d * %d", n1, s, d);
print ();

for i from 1 to limSup do
  if nextprime(a) < n1 then
    a := nextprime(a):
  else
    break:
  end if:
  cont := cont + 1:

  # testa a primeira condição
  printf("%d^%d mod %d = %d", a, d, n1, a &^ d mod n1);
```

```
  print ();
  if a &^ d mod n1 <> 1 mod n1 then
    # testa a segunda condição
    for l from 0 to s-1 do
      printf("%d^(2^%d * %d) mod %d = %d", a, l, d, n1, a &^((2^l)*d) mod n1);
      print ();
      if a &^((2^l)*d) mod n1 <> -1 mod n1 then
        primo := 0:
      else
        primo := 1:
        break:
      end if:
      if l = s-1 and primo = 0 then
        printf ("%d não é primo e %d é testemunha disso.", n1, a);
        print ();
        break:
      end if;
    end do;
  else
    primo := 1:
  end if;
  if l = s-1 and primo = 0 then
    break:
  end if;
end do:

if cont = limSup or (l = s-1 and primo <> 0) or (nextprime(a) >= n1
          and primo <> 0) then
  printf("Provavelmente %d é primo (com 99.99990463%% de probabilidade)
          ou pseudoprimo forte para todas as bases primas até %d
          (inclusive).", n1, a);
  print ();
end if;
```

Apêndice E

Método ρ de Pollard (algoritmo)

Exemplo.

```
with(numtheory):
n := readstat ("Introduza o número n, seguido de ';': ");
a := readstat ("Introduza o valor inicial da sequência: ");
b := a;
for i from 1 do
   printf ("i = %d", i);
   print();
   a := a &^ 2 + 1 mod n;
   printf ("x_%d = %d", i, a);
   print();
   b := b &^ 2 + 1 mod n;
   b := b &^ 2 + 1 mod n;
   printf ("x_%d = %d", 2*i, b);
   print();
   d := igcd (b-a, n);
   printf ("mdc (x_%d - x_%d, %d) = %d", 2*i, i, n, d);
   print();
   if d <> 1 then
      break:
   end if:
end do:
```

Apêndice F

Factorização de Fermat (algoritmo)

Exemplo. `n := 0: k := 0:`

```
numTestes := 0:
for i from 1 while type(n, integer) = false or n <= 0 do
  n := readstat("Introduza um inteiro positivo para n, seguido de ';': ");
end do;
for i from 1 while type(k, integer) = false or k <= 0 do
  k := readstat("Introduza um inteiro positivo para k, seguido de ';': ");
end do;
raizQuadradaDeN := floor(sqrt(k*n));
for i from 1 while type(numTestes, integer) = false or numTestes <= 0 do
  numTestes := readstat("Introduza um inteiro positivo para numTestes, seguido
                        de ';': ");
end do;
for i from 1 to numTestes do
  if issqr((raizQuadradaDeN + i)^2 - k*n) = true then
    t := raizQuadradaDeN + i;
    s := sqrt((raizQuadradaDeN + i)^2 - k*n):
    printf ("Os factores de %d são %d e %d.", n, igcd(t+s, n), igcd(t-s, n)):
    print  ():
    printf("Foram necessárias %d tentativas.", i):
    break:
  end if:
end do:
if i = numTestes + 1 and issqr((raizQuadradaDeN + i - 1)^2 - n) = false then
   printf ("Com %d testes, não conseguimos factorizar %d através da
            factorização de Fermat.", numTestes, n):
end if:
```

Apêndice G

Crivo quadrático (algoritmo)

Exemplo. `with(numtheory):`

```
n := 16843009;
m := floor(sqrt(n));
B := floor(evalf(exp(1/2 * sqrt(ln (n) * ln(ln (n))))));
b[1]  := 2:
primo := 3:
contador := 1:
for i from 2 while primo <= B do
  if legendre (n, primo) = 1 then
    contador := contador + 1;
    b[contador] := primo:
  end if:
  if nextprime(primo) <= B then
    primo := nextprime(primo):
  else
    break:
  end if:
end do:

# Guardar números e expoentes
limSup := 1000:
cont2 := 1:
for num from 1 to limSup do
  tabAux1[cont2] := num:
  tabAux2[cont2] := (m + num)^2 - n:
  tabAux3[cont2] := tabAux2[cont2]:
  for i from 1 to contador do
     cont3 := 0:
     for k while tabAux3[cont2] mod b[i] = 0 do
       tabAux3[cont2] := tabAux3[cont2] / b[i]:
```

```
      cont3 := cont3 + 1:
    end do:
    tabAux4[cont2][i] := cont3:
  end do:
  if tabAux3[cont2] = 1 then
    cont2 := cont2 + 1:
  end if:
  if num = limSup and tabAux3[cont2] <> 1 then
   cont2 := cont2 - 1:
  end if:
end do:

# Imprimir números B-suaves e expoentes
printf ("         i       f(i) "):
for i from 1 to contador do
  printf ("%6d ", b[i]):
end do:
print();
for i from 1 to cont2 do
  printf ("%10d ", tabAux1[i]):
  printf ("%10d ", tabAux2[i]):
  for j from 1 to contador do
    printf ("%6d ", tabAux4[i][j]):
  end do:
  print();
end do:
print();
print();
print();

# Imprimir números B-suaves e expoentes módulo 2
printf ("         i       f(i) "):
for i from 1 to contador do
  printf ("%6d ", b[i]):
end do:
print();
for i from 1 to cont2 do
  printf ("%10d ", tabAux1[i]):
  printf ("%10d ", tabAux2[i]):
  for j from 1 to contador do
    printf ("%6d ", tabAux4[i][j] mod 2):
  end do:
  print(); end do:
```

Apêndice H

Curvas elípticas - noções gerais

Para melhor percebermos a aplicação das *curvas elípticas*, iremos introduzir algumas noções básicas.

É importante salientar que as *curvas elípticas* não são elipses, mas derivam do termo *integral elíptica*, uma função f que pode ser expressa como

$$f(x) = \int_c^x R(t,\ P(t))dt,$$

sendo R uma função binária racional, P a raiz quadrada de um polinómio de grau 3 ou 4, com nenhuma raiz repetida, e c uma constante.

Definição. Um *corpo* é um conjunto $\mathcal{K}$ munido de duas operações, normalmente designadas por soma e multiplicação e denotadas por $+$ e $\cdot$, que obedecem às seguintes propriedades:

- fecho para a adição e para a multiplicação: $\forall a, b \in \mathcal{K},\ \ a + b \in \mathcal{K} \quad \text{e} \quad a \cdot b \in \mathcal{K}$;
- associativa para a adição e para a multiplicação;
- comutativa para ambas as operações;
- existência de elemento neutro para a adição (0) e para a multiplicação (1);
- existência de elemento simétrico para a adição ($-a$) e de elemento inverso para a multiplicação (a^{-1});
- distributiva da multiplicação sobre a adição.

Dado um corpo $\mathcal{K}$, considere-se a seguinte sucessão formada pelo elemento identidade: $1,\ 1+1,\ 1+1+1,\ \cdots$ Há duas possibilidades:

- Todos os termos da sucessão são diferentes de zero. Neste caso, dizemos que o corpo $\mathcal{K}$ tem *característica* 0.

- Alguns termos da sucessão são iguais a zero. Neste caso, diz-se que o corpo $\mathcal{K}$ tem característica p, onde p é o menor número natural tal que $1 + 1 + \cdots + 1(p$ vezes$) = 0$.

Definição (Curva elíptica)**.** Seja $\mathcal{K}$ um corpo com característica diferente de 2 e de 3 e seja $x^3 + ax + b$, com $a, b \in \mathcal{K}$, um polinómio cúbico sem raízes de multiplicidade superior a 1. Uma *curva elíptica* sobre $\mathcal{K}$ é o conjunto de pontos $(x, y) \in \mathcal{K}^2$, que satisfaz a equação

$$y^2 = x^3 + ax + b, \tag{H.1}$$

juntamente com um elemento singular, denotado por O, o chamado *ponto no infinito* (mais adiante iremos falar sobre este elemento).

Se $\mathcal{K}$ é um corpo de característica 2, a curva elíptica sobre $\mathcal{K}$ é o conjunto de pontos que satisfazem

$$y^2 + cy = x^3 + ax + b \quad \text{ou} \quad y^2 + xy = x^3 + ax_2 + b,$$

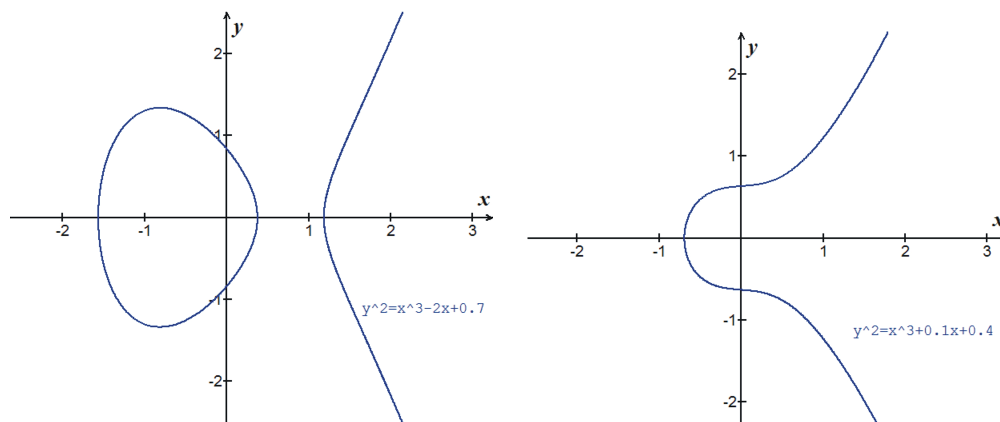
(aqui não nos preocupamos se o polinómio do segundo membro tem, ou não, raízes de multiplicidade superior a 1), mais o *ponto no infinito* O.

Se $\mathcal{K}$ é um corpo de característica 3, a curva elíptica sobre $\mathcal{K}$ é o conjunto de pontos que satisfazem

$$y^2 = x^3 + ax^2 + bx + c,$$

(o polinómio do segundo membro não pode ter raízes múltiplas), mais o *ponto no infinito* O.

Consoante o valor que atribuimos às constantes a e b, as curvas elípticas pode ter aspecos diversos. Vejamos o exemplo em que $a = -2$ e $b = 0,7$:



O corpo dos números reais tem característica 0 e é sobre este corpo que iremos focar, para já, a nossa atenção.

Definição. Seja E uma curva elíptica sobre o corpo dos reais, e sejam P e Q dois pontos de E. Definimos o negativo de P e a soma $P+Q$ da seguinte maneira:

- Se P é o *ponto no infinito* O, definimos $-P = O$ e $P+Q = Q$. Por outras palavras, O é o elemento neutro do conjunto de pontos. Iremos assumir, sempre que nada dissermos em contrário, que P e Q são distintos do *ponto no infinito* O.

- O negativo $-P$ é o ponto com a mesma abcissa, mas com a ordenada simétrica à de P, i.e., $-(x,y) = (x,-y)$. Pelo item anterior, verificamos que é óbvio que se (x,y) pertencem a E, então $(x,-y)$ também pertence a E;

- Se P e Q têm diferentes abcissas, não é difícil de ver que a linha $\ell = \overline{PQ}$ intersecta a curva num ponto R, distinto de P e Q (a menos que ℓ seja tangente à curva em P e, neste caso, tomamos $R = P$, ou tangente à curva em Q e, neste caso, tomamos $R = Q$). Depois, definimos $P+Q$ como sendo $-R$, i.e., o simétrico de R, em relação ao eixo dos xx (ver figuras H.1 e H.2).

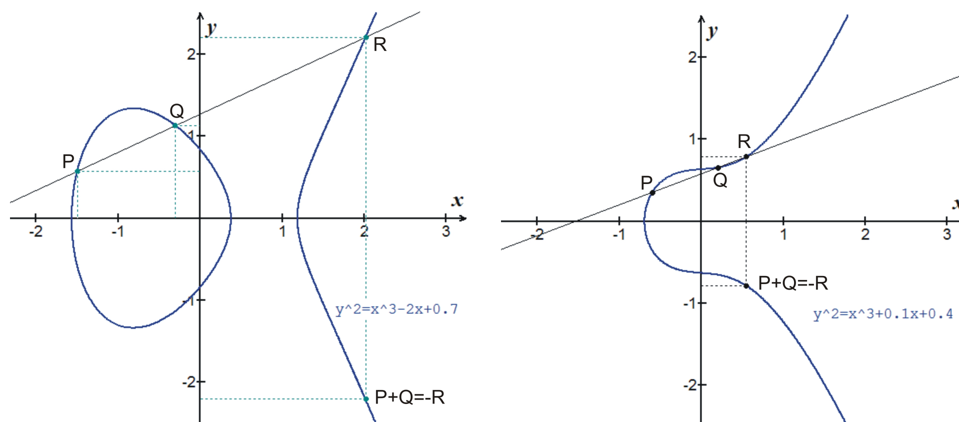


Figura H.1: Soma de dois pontos

- Se $Q = -P$, i.e., Q tem a mesma abcissa que P, mas ordenada simétrica, definimos $P+Q = O$, o *ponto no infinito* (ver figura H.3);

- Se $P = Q$, seja ℓ a tangente à curva em P, R o outro ponto que resulta da intersecção de ℓ com a curva. Definimos $P+Q = -R$.

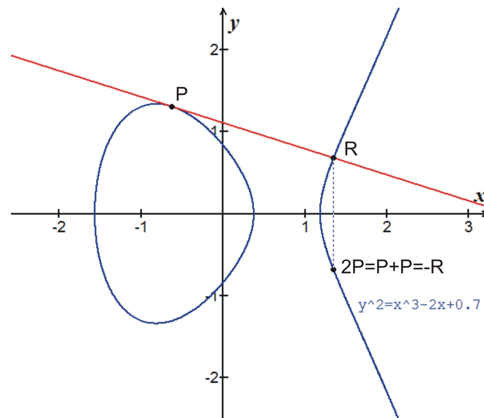


Figura H.2: O dobro de um ponto.

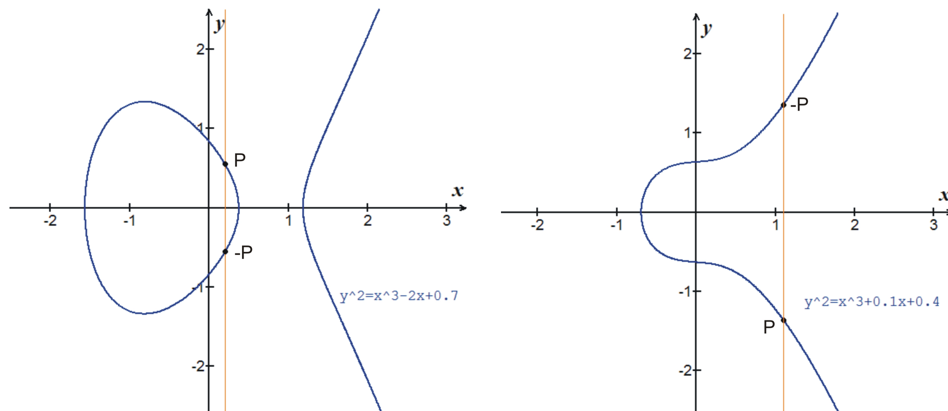


Figura H.3: Soma de um ponto com o seu negativo.

Vimos anteriormente que a soma de dois pontos de abcissas distintas é igual à reflexão, sobre o eixo das abcissas, do ponto de intersecção entre a curva elíptica e a recta ℓ que passa por aqueles dois pontos. Vamos mostrar que, quando adicionamos dois pontos de abcissas distintas, a recta ℓ intersecta forçosamente a curva elíptica. Aproveitaremos também para definir as coordenadas do ponto resultante da soma, em função das coordenadas dos dois pontos a adicionar.

Sejam $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $P + Q = (x_3, y_3)$ e considere-se a curva elíptica (E) de equação

$$y^2 = x^3 + ax + b,$$

que passa pelos pontos P e Q.

Para continuarmos com a nossa prova, precisamos do resultado seguinte:

Teorema H.1. *A soma das raízes de um polinómio mónico, em x, é igual ao simétrico do coeficiente do segundo termo de maior grau.*

Demonstração: No caso de termos um polinómio mónico do primeiro grau, a demonstração é trivial.

Consideremos o polinómio de grau 2:

$$f(x)(x-\alpha_2) = x^2 - (\alpha_1+\alpha_2)x + \alpha_1\alpha_2.$$

Neste caso, também se verifica que o coeficiente do segundo termo de maior grau é igual ao simétrico da soma das raízes de $f(x)(x-\alpha_2)$.

Consideremos, agora, o polinómio mónico de grau p,

$$x^p + a_{p-1}x^{p-1} + \cdots + a_1x + a_0$$

e suponhamos, por hipótese, que este polinómio tem p raízes, digamos

$$\alpha_1,\ \alpha_2,\ \cdots,\ \alpha_p$$

e que a soma destas raízes é igual a $-a_{p-1}$.

Considere-se, ainda, $\alpha_{p+1} \in F$.

Ao fazermos

$$(x-\alpha_{p+1})(x^p + a_{p-1}x^{p-1} + \cdots + a_1x + a_0)$$

obtemos um polinómio mónico de grau $p+1$. Como nos interessa olhar para o coeficiente do segundo termo de maior grau (o termo de ordem x^p), vamos centrar a nossa atenção no produto de x (do 1º factor) com $a_{p-1}x^{p-1}$ (do 2º factor) e o produto de $-\alpha_{p+1}$ (do 1º factor) com x^p (do 2º factor). No primeiro caso, obtemos $a_{p-1}x^p$ e, no segundo caso, $-\alpha_{p+1}x^p$, pelo que o coeficiente do segundo termo em x de maior grau é igual a $a_{p-1} - \alpha_{p+1}$. Como, por hipótese, $-a_{p-1} = \alpha_1 + \alpha_2 + \cdots + \alpha_p$, então

$$a_{p-1} - \alpha_{p+1} = -(\alpha_1 + \alpha_2 + \cdots + \alpha_p + \alpha_{p+1}),$$

o que prova o teorema. □

Como vimos anteriormente, quando somamos P com Q, passamos uma recta ℓ por P e Q para encontrar R. A sua expressão analítica desta recta será da forma

$$y = \alpha x + \beta.$$

Como assumimos que P e Q têm abcissas distintas, não corremos o risco de a recta ser vertical.

Sabemos que

$$\alpha = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{e} \quad \beta = y_1 - \alpha x_1.$$

Um ponto da recta ℓ será da forma $(x,\ \alpha x + \beta)$. Este ponto pertencerá à curva elíptica se e só se

$$(\alpha x + \beta)^2 = x^3 + ax + b, \tag{H.2}$$

isto é,

$$x^3 - (\alpha x + \beta)^2 + ax + b = 0.$$

A equação (H.2) é uma curva elíptica, que sabemos ter apenas raízes simples.

Como P e Q pertencem à curva elíptica,

$$(x_1,\ y_1) = (x_1,\ \alpha x_1 + b)$$

e

$$(x_2,\ y_2) = (x_2,\ \alpha x_2 + b)$$

são duas soluções desta equação, pelo que existe uma outra solução de (H.2), distinta destas.

Sabemos, do teorema anterior, que a soma das raízes de um polinómio mónico é igual ao simétrico do coeficiente do segundo maior termo em x, pelo que

$$x_3 = \alpha^2 - x_1 - x_2$$

e

$$y_3 = -\left(\alpha x_3 + \beta\right).$$

Escrevendo x_3 e y_3 em função de x_1, x_2, y_1 e y_2, temos:

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \tag{H.3}$$

e

$$\begin{aligned} y_3 &= -\alpha\left(\alpha^2 - x_1 - x_2\right) - \beta \\ &= -\alpha\left(\alpha^2 - 2x_1 - x_2\right) - y_1 \\ &= -\alpha(\alpha^2 - x_1 + x_3 - \alpha^2) - y_1 \\ &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 \end{aligned}$$

À medida que Q se aproxima de P, a linha ℓ aproxima-se da tangente da curva elíptica, em P. No caso em que $Q = P$, os cálculos são similares, excepto o facto

de que α é a derivada dy/dx em P. Calculando a derivada implícita da equação $y^2 = x^3 + ax + b$ ficamos com

$$2y\frac{dy}{dx} = 3x^2 + a,$$

ou seja,

$$\frac{dy}{dx} = \frac{3x^2 + a}{2y}.$$

Então,

$$\alpha = \frac{3x_1^2 + a}{2y_1}.$$

Obtemos, assim,

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$$

e

$$y_3 = -y_1 + \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3).$$

Multiplicar um ponto por um inteiro maior que 2, digamos l, equivale a somar esse ponto l vezes, i.e.,

$$lP = P + P + \cdots + P \quad (l \text{ vezes}).$$

Após este breve estudo sobre curvas elípticas, falta fazer algumas referências ao ponto O, o ponto no infinito. Como já nos apercebemos, este ponto é o elemento neutro para a adição. Podemos também vê-lo como sendo o "terceiro" ponto de intersecção entre uma linha vertical e a curva elíptica (neste caso, os pontos de intersecção são $(x_1,\ y_1)$, $(x_1,\ -y_1)$ e O).

Curvas elípticas sobre corpos finitos primos (F_p)

Consideremos $F_p = \{0,\ 1,\ 2,\ \cdots,\ p-1\}$.

Uma curva elíptica E sobre o corpo finito F_p é definida por todos os pontos

$$(x,\ y) \in F_p \times F_p$$

que obedecem a uma equação do tipo:

$$y^2 \mod p = x^3 + ax + b \mod p,$$

com $a, b \in F_p$, mais o *ponto no infinito* O. Dizemos que $(x,\ y) \in E(F_p)$.

Para garantirmos que o segundo membro da equação $(x^3 + ax + b)$ não tenha raízes múltiplas, temos que impôr que o discriminante[1] seja igual a zero, isto é,

$$4a^3 + 27b^2 \mod p \neq 0.$$

Sejam $P = (x_1,\ y_1)$, $Q = (x_2,\ y_2)$, $P + Q = (x_3,\ y_3) \in E(F_p)$. Utilizando os resultados a que chegámos na secção anterior (H),

- se $x_1 \neq x_2$, então

$$P + Q = \big(\alpha^2 - x_1 - x_2 \mod p, \quad -y_1 + \alpha(x_1 - x_3\big) \mod p),$$

  onde

$$\alpha = \left(\frac{y_2 - y_1}{x_2 - x_1}\right) \mod p. \tag{H.4}$$

- se $x_1 = x_2$ e $y_1 \neq 0$, então

$$P + Q = \big(\alpha^2 - 2x_1 \mod p, \quad -y_1 + \alpha(x_1 - x_3) \mod p\big),$$

  onde

$$\alpha = \left(\frac{3x_1^2 + a}{2y_1}\right) \mod p. \tag{H.5}$$

Exemplo. *Considere-se a curva elíptica $E(F_{23})$ de equação*

$$y^2 = x^3 + 2x + 1. \tag{H.6}$$

Reparemos que o discriminante desta equação é não nula:

$$4 \cdot 2^3 + 27 \cdot 1^2 \equiv 13 \not\equiv 0 \mod 23.$$

O ponto $(13,\ 4)$ *pertence à curva, já que:*

$$4^2 \mod 23 \equiv 13^3 + 2 \cdot 13 + 1 \mod 23.$$

[1] O discriminante de um polinómio cúbico do tipo $ax^3 + bx^2 + cx + d$ é igual a

$$b^2c^2 - 4ac^3 - 4b^3d - 27a^2d^2 + 18abcd.$$

Se esta expressão for igual a 0, então o polinómio tem uma raiz de multiplicidade superior a 1.

Para determinarmos todos os pontos de $E(F_{23})$, procedemos do seguinte modo:

1° *Determinar* $y^2 \mod 23$, *para todo o* $0 \leq y < 23$*:*

$$\begin{array}{ll}
0^2 \equiv 0 \mod 23 & 6^2 \equiv 17^2 \equiv 13 \mod 23 \\
1^2 \equiv 22^2 \equiv 1 \mod 23 & 7^2 \equiv 16^2 \equiv 3 \mod 23 \\
2^2 \equiv 21^2 \equiv 4 \mod 23 & 8^2 \equiv 15^2 \equiv 18 \mod 23 \\
3^2 \equiv 20^2 \equiv 9 \mod 23 & 9^2 \equiv 14^2 \equiv 12 \mod 23 \\
4^2 \equiv 19^2 \equiv 16 \mod 23 & 10^2 \equiv 13^2 \equiv 8 \mod 23 \\
5^2 \equiv 18^2 \equiv 2 \mod 23 & 11^2 \equiv 12^2 \equiv 6 \mod 23
\end{array}$$

2° *Determinar* $x^3 + 2x + 1 \mod 23$, *para todo o* $0 \leq y < 23$*:*

$$\begin{array}{ll}
0^3 + 2 \cdot 0 + 1 \equiv 1 \mod 23 & 12^3 + 2 \cdot 12 + 1 \equiv 5 \mod 23 \\
1^3 + 2 \cdot 1 + 1 \equiv 4 \mod 23 & 13^3 + 2 \cdot 13 + 1 \equiv 16 \mod 23 \\
2^3 + 2 \cdot 2 + 1 \equiv 13 \mod 23 & 14^3 + 2 \cdot 14 + 1 \equiv 13 \mod 23 \\
3^3 + 2 \cdot 3 + 1 \equiv 11 \mod 23 & 15^3 + 2 \cdot 15 + 1 \equiv 2 \mod 23 \\
4^3 + 2 \cdot 4 + 1 \equiv 4 \mod 23 & 16^3 + 2 \cdot 16 + 1 \equiv 12 \mod 23 \\
5^3 + 2 \cdot 5 + 1 \equiv 21 \mod 23 & 17^3 + 2 \cdot 17 + 1 \equiv 3 \mod 23 \\
6^3 + 2 \cdot 6 + 1 \equiv 22 \mod 23 & 18^3 + 2 \cdot 18 + 1 \equiv 4 \mod 23 \\
7^3 + 2 \cdot 7 + 1 \equiv 13 \mod 23 & 19^3 + 2 \cdot 19 + 1 \equiv 21 \mod 23 \\
8^3 + 2 \cdot 8 + 1 \equiv 0 \mod 23 & 20^3 + 2 \cdot 20 + 1 \equiv 14 \mod 23 \\
9^3 + 2 \cdot 9 + 1 \equiv 12 \mod 23 & 21^3 + 2 \cdot 21 + 1 \equiv 12 \mod 23 \\
10^3 + 2 \cdot 10 + 1 \equiv 9 \mod 23 & 22^3 + 2 \cdot 22 + 1 \equiv 21 \mod 23 \\
11^3 + 2 \cdot 11 + 1 \equiv 20 \mod 23 &
\end{array}$$

3° *Olhando para os valores obtidos nos passos anteriores, determinar os pares* (x, y) *que obedecem à equação (H.6):*

$$\begin{array}{llllll}
(0, 1), & (0, 22), & (1, 2), & (1, 21), & (2, 6), & (2, 17), \\
(4, 2), & (4, 21), & (7, 6), & (7, 17), & (8, 0), & (9, 9), \\
(9, 14), & (10, 3), & (10, 20), & (13, 4), & (13, 19), & (14, 6), \\
(14, 17), & (15, 5), & (15, 18), & (16, 9), & (16, 14), & (17, 7), \\
(17, 16), & (18, 2), & (18, 21), & (21, 9), & (21, 14), & O.
\end{array}$$

Marcando os pontos obtidos num referencial cartesiano, a curva elíptica terá o seguinte aspecto:

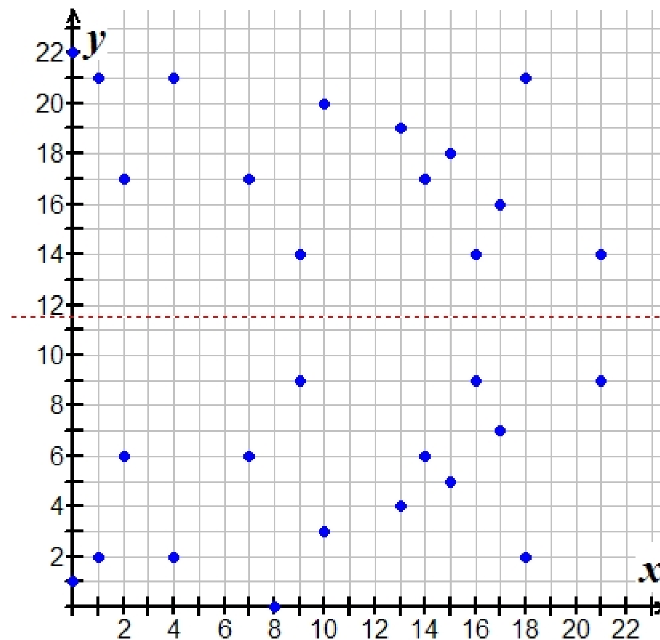


Figura H.4: Curva elíptica $y^3 = x^3 + 2x + 1$ sobre F_{23}.

Apesar do aparente caos da localização dos pontos, existe uma simetria relativamente a $y = 11,5$.

Para este trabalho interessa-nos trabalhar com curvas elípticas sobre um corpo finito $F_p = \mathbb{Z}/p\mathbb{Z}$, com p primo maior que 3. É fácil verificarmos que qualquer curva elíptica sobre este corpo tem, no máximo, $2p+1$ pontos: o ponto no infinito e os pontos $(x,\ y) \in F_p \times F_p$ que satisfazem

$$y^2 = x^3 + ax + b,$$

com $a, b \in F_p$ (note-se que, neste caso, F_p tem característica igual a q, distinto de 2 e de 3). Por cada um dos q valores para x há dois valores para y que satisfazem H.1. Vejamos quantas soluções em $F_p \times F_p$ tem uma curva elíptica com $a, b \in F_q$. Consideremos χ a função que, a cada $x \in F_p$, devolve 0 (quando $x = 0$), 1 (quando $x \neq 0$ é um quadrado perfeito) ou -1 (quando x não é um quadrado perfeito). Como p é primo

$$\chi(x) = \left(\frac{x}{p}\right),$$

o símbolo de Legendre. Se considerermos a equação $y^2 = u$, esta tem $1+\chi(u)$ soluções. Então, a curva elíptica de equação

$$y^2 = x^3 + ax + b$$

tem

$$1 + \sum_{x \in F_p} (1 + \chi(x^3 + ax + b)) = 1 + p + \sum_{x \in F_p} \chi(x^3 + ax + b)$$

soluções (contando com o ponto no infinito). Hasse descobriu que o valor de

$$\sum_{x \in F_p} \chi(x^3 + ax + b)$$

é limitado por $\sqrt{2p}$.

Teorema H.2. *(Hasse) Seja N o número de soluções de uma curva elíptica definida em F_p. Então,*

$$|N - (p + 1)| \leq \sqrt{2p}.$$

Podemos encontrar uma demonstração deste teorema no capítulo V de [30].

Vejamos, agora, quantas curvas elípticas existem [21] num corpo F_p, sendo p um primo superior a 3. O número total de curvas elípticas da forma (H.1) equivale ao número de pontos $(a, b) \in F_p \times F_p$, com $4a^3 + 27b^2 \neq 0$. Existem p^2 pares de números $(a,\ b)$ em $F_p \times F_p$. Por outro lado, a condição

$$4a^3 + 27b^2 \not\equiv 0 \mod n,$$

acontece se e só se $a = -3c^2$ e $b = 2c^3$, para algum $c \in F_p$. Este elemento c é determinado por a e por b, de forma única, por $c = -\dfrac{3b}{2a}$, com $a \neq 0$. Então $4a^3+2b^2 = 0$ é verificada para exactamente p pares de números $(a, b) \in F_p \times F_p$. Logo, o número de curvas elípticas em F_p é $p^2 - p$.

Definição (Ordem de um ponto de uma curva elíptica)**.** A ordem, n, de um ponto P de uma curva elíptica é o menor inteiro positivo tal que $nP = O$. A ordem de um ponto de uma curva elíptica nem sempre existe.

Curvas elípticas - redução módulo n

Definição. Sejam m um inteiro arbitrário.

Sendo x_1 e x_2 racionais com denominadores primos com m, dizemos que $x_1 \equiv x_2 \mod m$ se $x_2 - x_1$, escrito sob a forma de fracção irredutível, tiver numerador primo com m.

Para qualquer racional x_1 com denominador primo com m, existe um inteiro x_2, com $0 \leq x_2 \leq m-1$ tal que $x_1 \equiv x_2 \mod m$.

Teorema H.3. *Seja E uma curva elíptica de equação $y^2 = x^3 + ax + b$, com $a, b \in \mathbb{Z}$. Considere-se que mdc$(4a^3 + 27b^2,\ n) = 1$. Sejam P_1 e P_2 (com $P_1 \neq -P_2$) dois pontos de E cujas coordenadas têm denominadores primos com n.*

Então, os denominadores das coordenadas de $P_1 + P_2 \in E$ são primos com n se e só se não existe um divisor p de n tal que

$$P_1 \mod p + P_2 \mod p = O \mod p.$$

Demonstração: Sejam $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ e considere-se que $P_1 + P_2 \in E$. Suponha-se, ainda, que os denominadores das coordenadas de $P_1 + P_2$ são primos com n.

Pretendemos provar que $P_1 \mod p + P_2 \mod p \neq O \mod p$.

Suponhamos que $x_1 \not\equiv x_2 \mod p$. Então, de acordo com a descrição feita para a soma de dois pontos com abcissas distintas (apêndice H),

$$P_1 \mod p + P_2 \mod p \neq O \mod p.$$

Suponhamos, agora, que $x_1 \equiv x_2 \mod p$. Se $P_1 = P_2$, estamos no caso 5.5. Obtemos as coordenadas de P_1 mod $P + P_2$ mod P com a substituição, em 5.5, de cada um dos valores, módulo p, e escrevendo o resultado de cada uma das coordenadas módulo p. Precisamos mostrar que $2y_1 \not\equiv 0 \mod p$, ou seja, que $p \not| (2y_1)$. Se $p \mid (2y_1)$, como, por hipótese, o denominador da abcissa de $P_1 + P_2$ não é divisível por p, então teríamos que ter o numerador divisível por p, ou seja, teríamos que ter $3x_1^2 + a$ divisível por p. Neste caso, teríamos

$$y_1^2 \equiv x_1^3 + ax_1 + b \mod p$$

e

$$2y_1 = 3x_1^2 + a \mod p.$$

Como x_1 é solução de E mod p e da sua primeira derivada, então[2] é uma raiz de multiplicidade, no mínimo, 2, o que contradiz o facto de $E \mod p$ ser uma curva elíptica (só admite raízes simples). Admitamos, agora, que $P_1 \neq P_2$ e que $x_1 \neq x_2$. Como $x_1 \equiv x_2 \mod p$ e $x_1 \neq x_2$, então podemos escrever que $x_2 = x_1 + p^r x$, sendo $r \geq 1$ e sendo x um número racional cujo numerador e denominador não são divisíves por p. Como, por hipótese, o denominador das coordenadas de $P_1 + P_2$ não é divisível por p, podemos concluir, de 5.4, que $y_2 - y_1 \equiv 0 \mod p$, isto é, que $y_1 \equiv y_2 \mod p$. Por outras palavras, podemos escrever y_2 na forma $y_1 + p^r y$. Por outro lado, podemos escrever

$$\begin{aligned} y_2^2 &= (x_1 + p^r x)^3 + a(x_1 + p^r x) + b \\ &\equiv x_1^3 + ax_1 + b + p^r x(3x_1^2 + a) \mod p^{r+1} \\ &\equiv y_1^2 + p^r x(3x_1^2 + a) \mod p^{r+1}. \end{aligned} \tag{H.7}$$

Como $x_1 \equiv x_2 \mod p$ e $y_1 \equiv y_2 \mod p$, podemos concluir que

$$P_1 \mod p = P_2 \mod p.$$

Sendo assim, $P_1 \mod p + P_2 \mod p = 2P_1$ e devemos utilizar as fórmulas de (5.5) para obter as coordenadas de $2P_1$. Ora, $2P_1$ só é igual a O mod p se e só se $y_1 \equiv y_2 \equiv 0$ mod p. Se esta congruência acontecer e como $y_2 = y_1 + p^r y$, então

$$(y_2 + y_1) \cdot (y_2 - y_1) = y_2^2 - y_1^2 \equiv 0 \mod p^{r+1}.$$

Logo, em (H.7), obtemos $3x_1^2 + a \equiv 0 \mod p$, o que é impossível, uma vez que x_1 seria raiz de $x^3 + ax + b \mod p$ e da sua derivada e, portanto, seria uma raiz múltipla, o que não pode acontecer, uma vez que $E \mod p$ é uma curva elíptica. Então, P_1 mod $p + P_2 \mod p \neq O \mod p$, como pretendíamos.

Reciprocamente, suponhamos que, para todo o $p \mid n$, se tem

$$P_1 \mod p + P_2 \mod p \neq O \mod p.$$

Pretendemos provar que os denominadores das coordenadas de $P_1 + P_2$ são primos com p, ou seja, não são divisíveis por p, para todo o $p \mid n$. Fixemos um determinado p, divisor de n.

[2]Se f for k vezes continuamente derivável, na vizinhança de α, e

$$f(\alpha) = f'(\alpha) = \cdots = f^{(k-1)}(\alpha) = 0,$$

ou seja, se α for raiz de f e das suas primeiras $k-1$ derivadas, mas $f^{(k)}(\alpha) \neq 0$, então podemos escrever $f(x) = (x - \alpha)^k F(x)$. O recíproco também é verdadeiro.[13]

Suponhamos que $x_1 \not\equiv x_2 \mod p$ (neste caso, usamos as fórmulas de (5.4)). Então, o denominador de $P_1 \bmod p + P_2 \bmod p$, que é $x_2 - x_1$, não é divisível por p. Suponhamos, então, que $x_1 \equiv x_2 \mod p$. Temos que $y_2 \equiv \pm y_1 \mod p$. Como, por hipótese,

$$P_1 \mod p + P_2 \mod p \neq O \mod p,$$

então $y_1 \equiv y_2 \not\equiv 0 \mod p$. Se $P_1 = P_2$, então, pelas fórmulas (5.5), verificamos que

$$P_1 \mod p + P_2 \mod p$$

tem denominadores primos com p. Suponhamos, agora, que $P_1 \neq P_2$, isto é, $x_1 \neq x_2$ e $y_1 = y_2$. Relembremos que $x_1 \equiv x_2 \mod p$ e que $y_1 \equiv y_2 \not\equiv 0 \mod p$. Podemos escrever que $x_2 = x_1 + p^r x$, com $r \geq 1$ e sendo x um número racional cujo numerador e denominador não são divisíveis por p. Usando (H.7), podemos escrever

$$y_2^2 - y_1^2 \equiv p^r x(3x_1^2 + a) \mod p$$

e, portanto,

$$\frac{y_2^2 - y_1^2}{x_2 - x_1} \equiv 3x_1^2 + a \mod p.$$

Como p não divide y_1 e $y_1 + y_2 = 2y_1$, então p não divide

$$\frac{y_2^2 - y_1^2}{(x_2 - x_1)(y_2 + y_1)} = \frac{y_2 - y_1}{x_2 - x_1} \equiv \frac{3x_1^2 + a}{y_2 + y_1} \mod p,$$

pelo que podemos concluir que p não divide os denominadores das coordenadas de $P + Q$. □

Apêndice I

Método curvas elípticas (algoritmo)

Exemplo. `restart;`

```
with(numtheory):
n     := 45457;
B     := 20;
C     := ceil(sqrt(n)+1+2*sqrt(sqrt(n)));
discm := 0:
mdc   := n:
prim  := 1:
k     := 1:

# Escolha aleatória de x, y e a; cálculo de b, discriminante e
  mdc(discriminante, n)
nrand := rand(0..n-1):
for i while mdc = n or discm = 0 do
  x     := nrand():
  y     := nrand():
  a     := nrand();
  b     := y^2 - x^3 - a*x;
  discm := 4*a^3 + 27*b^2;
  mdc   := igcd (discm, n):
  if mdc > 1 and mdc < n then
    printf ("A factorização está concluída: %d = %d * %d.", n, mdc, n/mdc):
  end if:
end do:
printf ("P = (%d, %d)", x, y):
print  ():
printf ("E(a, b): y^2 = x^3 + %d x + %d", a, b):
print  ():
printf ("Discriminante de E(a, b) = %d", discm):
print  ():
```

```
printf ("mdc(discriminante, n) = %d", mdc):
print  ():

if mdc = 1 and discm <> 0 then
  # Cálculo de k
  if mdc = 1 then
    for i from 1 to pi(B) do
      prim := nextprime(prim):
      k := k * prim^floor(log(C) / log(prim)):
    end do:
  end if:
# k := 1258336903824000;
  printf ("k = %d", k):
  print  ():
  kbin   := convert(k, binary):
  printf ("k = (%d)_{2}", kbin):
  print  ():

  # último bit
  if kbin mod 2 = 0 then
    x3        := NULL:
    Pxparcial := NULL:
    y3        := NULL:
    Pyparcial := NULL:
    kbin := convert(convert(kbin, decimal, binary) / 2, binary):
  else
    x3        := x:
    Pxparcial := x:
    y3        := y:
    Pyparcial := y:
    kbin := convert((convert(kbin, decimal, binary)-1) / 2, binary):
  end if:
  length(kbin);

  # bits seguintes
  x1 := x:
  x2 := x:
  y1 := y:
  y2 := y:
  Px3 := x3:
  Py3 := y3:
  for i from 1 to length(convert(k, binary))-1 do
```

```
# cálculo de 2^{i}P mod n
if igcd(2*y1, n) = 1 then
  x3 := ((3 * x1^2 + a) / (2 * y1))^2 - 2*x1 mod n:
  y3 := - y1 + ((3 * x1^2 + a) / (2 * y1)) * (x1 - x3) mod n:
  x1 := x3:
  x2 := x3:
  y1 := y3:
  y2 := y3:
  printf ("2^{%d}P mod n = (%d, %d)", i, x3, y3):
  print  ():
else
  if igcd(2*y1, n) > 1 and igcd(2*y1, n) < n then
    printf("A factorização está concluída: %d = %d * %d", n,
    igcd(2*y1, n), n/igcd(2*y1, n)):
    print ():
    break:
  else
    print("Sugiro que escolha uma outra curva elíptica E(a, b)
          / P(x,y) / k."):
    break:
  end if:
end if:

# actualização de kP mod n, sempre que o bit for 1
if kbin mod 2 = 1 then
  if Pxparcial = NULL and Pyparcial = NULL then
    Pxparcial := x3:
    Pyparcial := y3:
    Px3       := x3:
    Py3       := y3:
  else
    if x3 <> Px3 then
      if igcd (Px3 - x3, n) = 1 then
        Pxparcial := ((Py3 - y3) / (Px3 - x3))^2 - x3 - Px3 mod n:
        Pyparcial := ((Py3 - y3) / (Px3 - x3)) * (x3 - Pxparcial) - y3 mod n:
        Px3 := Pxparcial:
        Py3 := Pyparcial:
      else
        if igcd(Px3 - x3, n) > 1 and igcd(Px3 - x3, n) < n then
          printf("A factorização está concluída: %d = %d * %d", n,
                 igcd(Px3 - x3, n), n/igcd(Px3 - x3, n)):
          print ():
          break:
```

```
          else
            print("Sugiro que escolha uma outra curva elíptica E(a, b)
                   / P(x,y) / k."):
            break:
          end if:
        end if:
      else
        if x3 = Px3 and y3 = Py3 then
          if igcd (2*y3, n) = 1 then
            Pxparcial := ((3*x3^2 + a) / (2 * y3))^2 - 2 * x3 mod n:
            Pyparcial := - y3 + ((3*x3^2 + a) / (2 * y3)) *
                               (x3 - Pxparcial) mod n:
            Px3 := Pxparcial:
            Py3 := Pyparcial:
          else
            if igcd(2*y3, n) > 1 and igcd(2*y3, n) < n then
              printf("A factorização está concluída: %d = %d * %d", n,
                      igcd(2*y3, n), n/igcd(2*y3, n)):
              print  ():
              break:
            else
              print("Sugiro que escolha uma outra curva elíptica
                     E(a, b) / P(x,y) / k."):
              break:
            end if:
          end if:
        end if:
      end if:
    end if:
    kbin := convert((convert(kbin, decimal, binary)-1)/2, binary):
  else
    kbin := convert(convert(kbin, decimal, binary)/2, binary):
  end if:
  if i = length(convert(k, binary))-1 then
    printf ("kP mod n = %dP mod n = (%d, %d)", k, Pxparcial, Pyparcial):
    print  ():
    printf ("Sugiro que aumente o valor de k ou que escolha outra curva
             elíptica E(a,b)."):
    print  ():
  end if:
 end do:
end if:
```

Apêndice J

Expoente privado pequeno (algoritmo)

Exemplo. `restart;`

```
with(numtheory):
for i from 1 while type(n, integer) = false or n <= 0 do
  n := readstat("Insira o valor de n no sistema decimal, seguido de ';': ");
end do:
for i from 1 while type(e, integer) = false or e <= 0 do
  e := readstat("Insira o valor de e no sistema decimal, seguido de ';': ");
end do:
for i from 2 to nops(cfrac(e / n, 'quotients'))-1 do
  conv := nthconver(cfrac(e/n), i):
  printf("%d° convergente = %a", i, conv);
  print();
  k     := numer(conv):
  d     := denom(conv):
  if (e * d - 1) mod k = 0 then
    fi_n  := (e * d - 1) / k:
  if isolve(p^2 - (n - fi_n + 1) * p + n = 0) <> NULL then
      sols := [solve(p^2 - (n - fi_n + 1) * p + n = 0)];
      p := sols[1]:
      q := sols[2]:
      printf ("k = %d", k):
      print():
      printf ("d = %d", d):
      print():
      printf ("p = %d", p):
      print():
      printf ("q = %d", q):
```

```
      print():
      printf ("phi(n) = %d", fi_n):
      print():
      break:
    end if:
  end if:
end do:
```

Apêndice K

Ataque com parte da chave privada exposta (algoritmo)

Exemplo. `with(numtheory):`

```
restart;
n    := 177299;
ifactor(n);
e    := 11;
convert(n, binary);
beta := length(convert(n, binary));
b    := ceil(beta/4);
d    := n mod 2^b;
p := NULL:
for k from 1 to 2^b while p = NULL do
  for p0 from 1 to 2^b while p = NULL do
    if k*p0^2 + (e*d - k*n - k - 1)*p0 + k*n mod 2^b = 0 then
      for q0 from 1 to 2^b do
        if p0*q0 mod 2^b = n mod 2^b then
          printf("p0 = %d, q0 = %d", p0, q0);
          print ();
          for x from 1 to ceil(sqrt(n)) do
            if type(n / (2^b * x + p0), integer) = true then
              p := 2^b * x + p0;
              q := n / p;
          printf("x = %d, y = %d", x, (n / p - q0) / 2^b);
              print ();
              printf("p = %d, q = %d", p, q);
              print ();
              printf("phi(%d) = %d", n, phi(n));
              print ();
```

```
            printf("d = %d", e^(-1) mod phi(n));
              print ();
              break;
            end if;
          end do;
        end if;
      end do:
    end if;
  end do;
end do:
```

Bibliografia

[1] ALFORD, William Robert; GRANVILLE, Andrew James; POMERANCE, Carl. *On the difficulty of finding reliable witnesses.* ANTS-I: Proceedings of the First International Symposium on Algorithmic Number Theory, 1994, pág. 1–16, Springer-Verlag.

[2] ARNAULT, François. *Le test de primalité de Rabin-Miller: un nombre composé qui le "passe"* . Université de Poitiers, número 61, Novembro de 1991;

[3] BACH, Eric. *Analytic methods in the analysis and design of number - theoretical algorithms.* Uma dissertação distinguida pela ACM, em 1984. MIT Press, 1985.

[4] BARBEDO, Inês; MACHIAVELO, António (orientador) *O sistema criptográfico RSA: ataques e variantes.* Tese de mestrado em "Matemática - Fundamentos e Aplicações", Departamento de Matemática Pura, Faculdade de Ciências, Universidade do Porto do Porto, 2003.

[5] BONEH, Dan. *Twenty years of attacks on the RSA cryptosystem.* Notices of the AMS, 1999, Vol. 46, pág. 203–213;

[6] BONEH, Dan; DURFEE, Glenn; FRANKEL, Yair. *Exposing an RSA Private Key Given a Small Fraction of its Bits.* in Proceedings Asiacrypt '98, Lecture Notes in Computer Science, 1998, Vol. 1514, Springer, pág. 25–34;

[7] BRIGGS, Matthew Edward. *An Introduction to the General Number Field Sieve.* Master's thesis, Virginia Polytechnic Institute and State University, 1998. URL: http://scholar.lib.vt.edu/theses/available/etd-32298-93111/unrestricted/etd.pdf

[8] CASE, Michael. *A Beginner's Guide To The General Number Field Sieve.* Artigo não publicado, Oregon State University, 2003. URL: islab.oregonstate.edu/koc/ece575/03Project/Case/paper.pdf

[9] CHILDS, Lindsay N. *A concrete introduction to higher algebra*. Springer-Verlag, Berlin, 1979, 3ª edição.

[10] COPPERSMITH, Don. *Finding a small root of a univariate modular equation.* in Proc. of Eurocrypt '96, 1996, pág. 155–165;

[11] CRANDALL, Richard; POMERANCE, Carl. *Prime numbers: a computational perspective*. Springer-Verlag, New York, 2005, 2ª edição.

[12] DELAURENTIS, John M. *A further weakness in the common modulus protocol for the RSA cryptoalgorithm*. Cryptologia, 1984, Vol. 8, N. 3, pág. 253–259;

[13] EPPERSON, James F. *An introduction to numerical methods and analysis*. John Wiley and Sons (Hoboken, N.J), revisited edition, 2007, pág. 147;

[14] HARDY, Godfrey Harold; WRIGHT, Edward Maitland. *An introduction to the theory of numbers*. 5ª edição. Clarendon Press, Oxford, 1979.

[15] HASTAD, Johan. *Solving simultaneous modular equations of low degree*. SIAM Journal of Computing, 1988, Vol. 17, pág. 336–341;

[16] HOWGRAVE-GRAHAM, Nick. *Finding small roots of univariate modular equations revisited*. Proceedings of the 6th IMA International Conference on Cryptography and Coding, Springer-Verlag, London, UK, 1997, pág. 131–142;

[17] KINCHIN, Aleksandr Yakovlevich. *Continued Fractions* Dover Publications, 3ª edição, 1997, pág. 30.

[18] KOBLITZ, Neal. *A Course in Number Theory and Cryptography*. Graduate Texts in Mathematics, 1994, 2ª edição, Springer-Verlag, Berlin-Heidelberg, pág. 164–165.

[19] KOCHER, Paul Carl. *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*. CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, 1996, Springer-Verlag, Berlin-Heidelberg, Londres, pág. 104–113.

[20] LENSTRA, Arjen Klaas; LENSTRA Jr; Hendrik Willem; LOVÁSZ, Lászlo. *Factoring polynomials with rational coefficients*. Mathematische Annalen, 1982, Vol. 261, pág. 515–534, URL: https://openaccess.leidenuniv.nl/dspace/handle/1887/3810;

[21] LENSTRA Jr, Hendrik Willem. *Factoring integers with elliptic curves.* Annals of Mathematics, 1987, Vol. 126, pág. 649–673, URL: http://hdl.handle.net/1887/3826;

[22] LUCAS, Édouard. *Sur la recherche des grands nombres premiers.* AFAS, Congrès de Clermont Ferrand, 1876, Vol. 5, pág. 61–68.

[23] MILLER, Gary Lee. *Riemann's hypothesis and tests for primality.* in Proceedings of seventh annual ACM symposium on Theory of computing, 1975, pág. 234–239, URL: http://cr.yp.to/bib/entries.html#1975/miller;

[24] MOLLIN, Richard Anthony. *RSA and Public-Key Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 2002, 1ª edição, pág. 60–65.

[25] POLLARD, John M. *Theorems on factorization and primality testing.* Proceedings of the Cambridge Philosophical Society, 1974, Vol. 76, pág. 521–528, URL: http://cr.yp.to/bib/entries.html#1974/pollard;

[26] POMERANCE, Carl. *A tale of two sieves.* Notices of the American Mathematical Society, 1996, Vol. 43, pág. 1473–1485.

[27] RABIN, Michael Oser. *Probabilistic algorithm for testing primality.* Journal of Number Theory, 1980, Vol. 12, N. 1, pág. 128–138, URL: http://cr.yp.to/bib/entries.html#1980/rabin;

[28] RIVEST, Ronald Linn. *Remarks on a proposed cryptanalytic attack on the M.I.T. public-key cryptosystem.* Cryptologia, 1978, Vol. 2, N. 1, pág. 62–65.

[29] RIVEST, Ronald Linn; SHAMIR, Adi; ADLEMAN, Leonard Max *A method for obtaining digital signatures and public-key cryptosystems.* Communications of the ACM, 1978, Vol. 21, N. 2, pág. 120–126.

[30] SILVERMAN, Joseph Hillel. *The arithmetic of elliptic curves.* Springer-Verlag, Berlin, 1986;

[31] SIMMONS, Gustavus J. *A "weak" privacy protocol using the RSA crypto algorithm.* Cryptologia, 1983, Vol. 7, N. 2, pág. 180–182;

[32] SIMMONS, Gustavus J.; NORRIS, Michael J. *Preliminary Comments on the MIT Public-Key Cryptosystem.* Cryptologia, 1977, Vol. 1, N. 4, pág. 406–414.

[33] SINGH, Simon. *O Livro dos Códigos.* Temas e Debates, 1999, 1ª edição.

[34] YAN, Song Y. *Cryptanalytic Attacks on RSA*. Springer-Verlag New York, Inc., 2008.

[35] YAN, Song Y. *Number Theory for Computing*. Springer-Verlag New York, Inc., 2002, 2ª edição, pág. 266–267.