Jorge Manuel Lopes Santos

O uso de cifragem para protecção de canais abertos

Departamento de Matemática Pura

Faculdade de Ciências da Universidade do Porto

Setembro de 2002

Jorge Manuel Lopes Santos

O uso de cifragem para protecção de canais abertos

Dissertação submetida para a satisfação parcial dos requisitos
do grau de mestre em Ensino da Matemática.

Departamento de Matemática Pura
Faculdade de Ciências da Universidade do Porto

Setembro de 2002

Dissertação realizada sob a supervisão do

Prof. Doutor Vítor Araújo,

Professor Auxiliar do

Departamento de Matemática Pura da

Faculdade de Ciências da Universidade do Porto

Agradecimentos

Muito em especial, agradeço:

Ao Professor Vítor Araújo. Ao Rui Duarte e ao Fernando Almeida. À escola básica e secundária de São João da Pesqueira e de Mêda, por possibilitarem a realização deste trabalho. À minha família, pela paciência e pelo apoio moral. Aos meus amigos. Por último, (os últimos são sempre os primeiros), para quem eu dedico este trabalho, à *Clara*.

Conteúdo

Lista de Tabelas

Lista de Figuras

Introdução

" O segredo melhor guardado é o que a ninguém é revelado"

Criptologia: *A ciência do segredo*[1]. A arte de criar criptogramas, mensagens compreendidas unicamente pelo seu destinatário.

Esta arte serviu apenas interesses governamentais e acima de tudo militares. Mas, nos dias de hoje, ocorre precisamente o oposto. A ciência dos códigos secretos serve propósitos bem mais abertos e amplos. Através da criptografia, noções como confidencialidade, autenticidade, confiança e segurança estão ao alcance de todos nós.

A criptografia clássica era uma criptografia simétrica, de chave secreta. A mesma chave secreta que cifra os dados, é a que os decifra.

Contudo, surgem dois problemas de grande importância: a chave que cifra os dados é a mesma que os decifra, daí, a necessidade de uma *distribuição de chaves secretas* e a necessidade de grandes quantidades de números aleatórios para a geração da chave secreta.

A segurança dos dados só era conseguida quando existia um canal fisicamente seguro para a entrega da chave secreta.

A solução surge com *Whitfield Diffie* e *Martin Hellman* ao inventarem a *criptografia de chave pública*, que permite comunicar em segurança na presença de terceiros, sem previamente necessitar de uma troca de chaves secretas. O canal de comunicação pode ser mesmo público, admitindo que esse mesmo canal não pode ser modificado nem impedido.

Os criptossistemas de chave pública usados actualmente recorrem essencialmente a operações de exponenciação e operações modulares em relação a primos gigantescos em corpos ou anéis finitos. Isto torna-os bem mais lentos do que os criptossistemas simétricos. Para comunicar em segurança num canal aberto, é comum usar criptossistemas simétricos para a tarefa de cifragem de grandes quantidades de dados, e atribuir a tarefa auxiliar

[1]Citação de *Jacques Stern* que intitula o seu livro consagrado à criptologia.

aos "lentos" criptossistemas de chave pública – a distribuição periódica de chaves secretas (*chaves de sessão*).

O facto de não existir uma quantidade (chave secreta) comum para o emissário e o destinatário, possibilita que a criptografia de chave publica responda a outras necessidades importantes, além da cifragem de dados, que somente as suas características únicas o permitem.

Listamos a seguir algumas tarefas que a criptografia de chave pública pode facultar:

Cifra de chave pública Podemos emitir uma chave pública para que todos possam cifrar mensagens mas somente uma única pessoa, o proprietário da chave pública e privada, possa ler essa mensagem com uma chave privada correspondente.

Estabelecimento de chaves secretas Podemos estabelecer uma quantidade secreta comum, uma chave secreta de sessão, para que duas pessoas possam comunicar num canal público que não pode ser impedido nem modificado.

Identificação e autenticação de pessoas Permite certificar a identidade de uma determinada pessoa durante uma troca de mensagem.

Assinaturas digitais Um modo simples e elegante de provar que uma mensagem é de quem se diz ser, suprimindo qualquer possibilidade de forjar a nossa identidade. No entanto, as assinaturas digitais, como as convencionais, podem ser forjadas. A diferença é que a assinatura digital pode ser matematicamente verificada atestando a sua integridade e autenticidade.

A assinatura digital deve mesmo constituir prova inegável, mesmo perante a justiça. Os seus parâmetros devem ser suficientemente seguros para que esta seja válida durante uma boa dezena de anos. Isto era impossível antes do aparecimento da criptografia de chave pública. Ver 1.4.

Acreditação de entidades certificadoras Permite convencer uma pessoa interagindo com uma entidade acreditada. Permite também assinaturas digitais utilizando funções

hash que substituem as questões da entidade verificadora. (Heurística de *Fiat-Shamir*, [25]).

Problemas repartidos Permite o voto electrónico. Para realizar uma votação segura, o sistema deve verificar cinco propriedades: só podem votar eleitores autorizados; cada eleitor pode votar uma só vez; ninguém deve poder determinar o voto dos outros; ninguém pode alterar os votos sem ser descoberto; e todos os eleitores podem verificar o processo de contagem dos votos e certificar que os seus votos foram apurados correctamente (ver [37]).

Dinheiro digital Consiste num sistema de transacções bancárias, em que se pode controlar a legitimidade das transacções efectuadas, mas sem invadir o sigilo de cada transacção. Por exemplo, o banco não sabe para onde o cliente transfere o seu dinheiro; o banco somente pode determinar se o cliente tem a quantia que pretende transferir (ver [37]).

O que isto pode permitir no futuro: o fim de vários postos de atendimento público, para tudo o que diz respeito a documentação, reclamações, ofícios, requerimentos, pagamentos etc, (Livro Verde para a Sociedade de Informação), aumentando a celeridade de todos os processos burocráticos. Juntamente com uma boa infra-estrutura de telecomunicações, permitirá a implantação de novos postos de teletrabalho com toda a segurança e privacidade. Muitas outras aplicações surgirão, sem dúvida, no futuro.

Neste trabalho, vamos considerar dois sistemas de cifragem mais pormenorizadamente, um deles baseado no problema da *factorização de números inteiros*, o RSA 1.2, e outro baseado no *problema da soma de subconjuntos*, ou *problema da mochila* (*"knapsack"*) 1.3.

Antes disso (§ 1.1.2) definiremos alguns conceitos de Complexidade Computacional que nos permitirão comparar problemas computacionais e dizer que uns são "mais difíceis" ou "mais fáceis" do que outros. No final do primeiro capítulo (§ 1.4), apresentamos um esquema de Assinatura Digital baseado no sistema de cifragem de chave pública RSA.

Há diversos problemas técnicos envolvidos na implantação de um sistema de comunicação efectivo com criptografia de chave pública.

A geração de chaves pública e privada para cada interlocutor, afim de serem usadas por algum método de criptografia de chave pública, exige a geração automática de

números "aleatórios", de maneira que se possa garantir com razoável fiabilidade que não há diferentes utilizadores com idênticas chaves públicas e/ou privadas. Isto é conseguido com algoritmos de geração de "Números Pseudoaleatórios", assunto tratado no segundo capítulo: estes métodos permitem a qualquer computador gerar sequências de números que, não sendo "verdadeiramente aleatórios", se assemelham, de muitos pontos de vista, a sequências de números obtidos ao acaso. Afloraremos outras utilizações mais antigas deste geradores na Análise Numérica.

Como ficará claro logo no primeiro capítulo, o método RSA baseia-se no uso engenhoso de pares de números primos, que vão essencialmente formar o par chave pública – chave privada de cada utilizador. É necessário então gerar aleatoriamente números primos! Na verdade, o que se faz é gerar inteiros aleatórios (capítulo 2) e *testá-los para verificar a sua primalidade.* Os Testes de Primalidade são um tema fascinante da Teoria de Números há séculos, e veremos no terceiro capítulo como ganharam actualidade ao se transformarem num dos suportes do método RSA.

No último capítulo desta dissertação mostramos como se pode instalar num computador, configurar e usar uma concretização dos resultados teóricos apresentados: o software "Pretty Good Privacy" (PGP) que permite cifrar todo o nosso correio electrónico usando criptografia de chave pública, em particular o sistema de cifragem RSA.

Na realidade, esta dissertação teve como motivação e guia a apresentação cuidada das ideias, métodos e principais resultados que permite software como o PGP e GnuPG (versão em software livre para o sistema operativo GNU/Linux) realizarem a cifragem e decifragem de mensagens de correio electrónico. De facto, como memorado em muitas passagens do texto, várias secções desta dissertação foram motivadas pelo código fonte o PGP e do GnuPG (ambos disponíveis na Internet), para compreender o funcionamento interno destes programas.

Este é um bom exemplo da aplicação muito bem sucedida da Matemática, e logo de um dos seus ramos mais antigos e tradicionalmente mais afastado das "questões práticas" a problemas centrais da sociedade contemporânea, com implicações sociais, económicas e políticas de grande alcance: a comunicação à distância com rapidez, fiabilidade, segurança e privacidade.

Capítulo 1

Métodos de protecção de comunicações

Por volta dos anos 70, duas descobertas revolucionaram o curso da história da criptologia: a *Criptografia de Chave Pública* e a *Autenticação.*

A criptografia clássica era uma criptografia simétrica, de chave secreta. A mesma chave secreta que cifra os dados, é a que os decifra. Um bom exemplo é a cifra de *Vernam*, a única cifra que foi provada ser inquebrável, inventada nos laboratórios AT&T em 1917. Se duas pessoas decidem comunicar secretamente num canal aberto, devem acordar uma "frase" de bits $x_1x_2x_3\ldots$ – a chave secreta – obtida por um gerador de bits aleatórios criptograficamente seguro, ver 2.2.4. A chave secreta deverá ter o mesmo tamanho em bits do que a mensagem a enviar e será usada uma só vez. Para cifrar uma mensagem representada pelos bits $m_1m_2\ldots m_k$, calculamos os bits $c_i = m_i \oplus x_i,\ 1 \leq i \leq k$, em que $x \oplus y$ representa uma "disjunção exclusiva" dos bits x e y, ou seja, 1 se $x \neq y$ e 0 se $x = y$. Para recuperar a mensagem original calculamos

$$c_j \oplus x_j = (m_j \oplus x_j) \oplus x_j = m_j = m_j \oplus (x_j \oplus x_j) = m_j \oplus 0 = m_j,\ 1 \leq j \leq k.$$

Exemplo 1.0.1 (Cifra de Vernam) Pretendemos cifrar a mensagem "*Segredo*". A representação numérica dos caracteres da palavra *Segredo*, recorrendo à tabela ASCII, é dada na base hexadecimal por 53 65 67 72 65 64 6F e na base binária por

$$m = \texttt{01010011 01100101 01100111 01110010 01100101 01100100 01101111}.$$

Consideremos a chave secreta, 8A 46 03 3C 45 74 08 acordada entre duas entidades para uso na cifra de *Vernam*, cuja representação binária é

$$x = \texttt{10001010 01000110 00000011 00111100 01000101 01110100 00001000}.$$

Calculamos a mensagem cifrada

$$c = \texttt{11011001 00100011 01100100 01001110 00100000 00010000 11010111},$$

onde $c_i = m_i \oplus x_i$, $1 \leq i \leq 56$. Na base hexadecimal, a mensagem cifrada é representada por $c =$ D9 23 64 4E 20 10 67. Para voltar a recuperar a mensagem m, basta calcular $m_i = c_i \oplus x_i$, $1 \leq i \leq 56$.

Consideremos que a chave secreta (x_j) é aleatória e não é usada mais do que uma só vez – ou seja, para responder à primeira mensagem com k bits usamos os bits $x_{k+1}, x_{k+2}, \ldots$ da chave secreta – nestas condições, a *cifra de Vernam* (*one time pad*) é inquebrável. Vejamos que, de facto, esta cifra é inquebrável. Suponha que interceptou uma pequena parte da mensagem cifrada, digamos 8 bits. Imagine que tem conhecimento que os 8 bits representam uma letra ASCII 'M' ou uma letra ASCII 'A'. Sabemos que o inimigo ataca pelo mar se a mensagem representar um 'M' ou que o inimigo ataca pelo ar se representar um 'A'. Note-se que sabemos muito acerca da mensagem cifrada! Tudo o que nós procuramos é a chave secreta com vista a recuperar a mensagem original. Podemos efectuar essa procura de forma intensiva e testar todas as 256 ($\#\{0,1\}^8$) hipotéticas chaves de 8 bits. Os resultados da procura são duas chaves de 8 bits, uma chave que decifra a mensagem obtendo 'M' e a outra decifra a mensagem obtendo 'A'. Como a chave secreta é aleatória, continuamos sem saber qual dos dois casos é a mensagem original.

Contudo, surgem dois problemas de grande importância: a chave que cifra os dados é a mesma que os decifra, daí, a necessidade de uma *distribuição de chaves secretas* e a necessidade de grandes quantidades de números aleatórios para a geração da chave secreta.

Existem criptossistemas convencionais que não necessitam grandes quantidades de números aleatórios, cujos níveis de segurança são plenamente satisfatórios. Por exemplo: o DES (Data Encryption Standard), desenvolvido pela IBM, usa apenas uma chave de 64 bits (Para mais detalhes veja §7.4.2 em [25]). Com o criptossistema DES, as chaves secretas são de tamanho aceitável, ficando eliminado o principal inconveniente da *cifra de Vernam*. Mas ainda persiste o *problema da distribuição de chaves* secretas!

A segurança dos dados só era conseguida quando existia um canal fisicamente seguro para a entrega da chave secreta. Por exemplo: um carro blindado de uma empresa de segurança que efectua a entrega da chave secreta de mão em mão, sem quaisquer testemunhas. Podemos imaginar os milhões que se gastariam se tal fosse necessário cada vez que uma entidade pública, privada ou governamental trocasse informação confidencial, ou até mesmo quando efectuássemos uma simples compra na Internet. Um outro problema reside na quantidade de chaves a distribuir. Se um sistema tiver n utilizadores, então são necessárias distribuir $\frac{n(n-1)}{2}$ chaves e cada utilizador deverá guardar $n-1$ chaves. Uma vez que existem cerca de 20 milhões de cibernautas, para disponibizar um sistema de comunicações seguro, teria de se distribuir cerca de 200 triliões de chaves e cada utilizador teria de guardar 20 milhões de chaves, uma para cada utilizador!

A solução surge com *Whitfield Diffie* e *Martin Hellman* ao inventarem a *criptografia de chave pública.* Permite comunicar em segurança na presença de terceiros, sem previamente necessitar de uma troca de chaves secretas. Aliás, o canal de comunicação pode ser mesmo público, admitindo que esse mesmo canal não pode ser modificado nem impedido.

1.1 Método de chave pública

Considere $\mathcal{M}$ o conjunto de todas as possíveis unidades de mensagens definidas num determinado alfabeto de letras $\mathcal{A}$, e $\mathcal{C}$ o conjunto de todas as possíveis unidades de mensagens cifradas. Um *criptossistema* é uma família de funções injectivas $f_C : \mathcal{M} \to \mathcal{C}$, cada função correspondendo a um parâmetro C a que chamamos *chave.* O conjunto $\mathcal{K}$ designa o conjunto de todas as chaves possíveis C.

O conceito de criptografia de chave pública foi inventado e publicado em 1976 por *Diffie* e *Hellman*, ver [11]. Vejamos qual a ideia de *Diffie* e *Hellman.*

Nos criptossitemas simétricos, tal como a cifra de *Vernam*, a função de cifragem era um processo de dois sentidos, ou seja, conhecendo a "chave", podíamos cifrar e decifrar mensagens.

$$\underset{\text{João}}{f_C(M) = m} \longrightarrow \text{Canal Público} \longrightarrow \underset{\text{Pedro}}{f_C^{-1}(m) = M}$$

A sugestão de *Diffie* e *Hellman* foi: construir uma "*função de um só sentido*" caracterizada por um parâmetro, a *chave pública* C, de tal forma que $f_C(M) = m$ seja fácil de calcular,

mas que a função inversa de f_C fosse muito difícil de calcular. Contudo, inverter a função f_C deve ser muito fácil se tivermos acesso a uma informação adicional – a *chave privada* D. Uma função que verifique estas propriedades diz-se uma "*one way function trapdoor*". *Diffie* e *Hellman* não conseguiram pôr completamente esta ideia a funcionar, mas o esforço de *Ron Rivest*, *Adi Shamir* e *Leonard Adleman* seria recompensado em 1977, publicando o sistema de cifragem RSA (§1.2), o primeiro criptossistema de chave pública viável.

Em 1997, segundo um artigo de *James Ellis* [1] datado de 1977, artigo até então confidencial da entidade governamental britânica CESG (Communications Electronics Security Group), a criptografia de chave pública teria sido inventada em 1970 pelo próprio *James Ellis* [12]. Mais ainda, em 1973, em pouco menos de trinta minutos e sem a possibilidade de registo em suporte de papel, como afirma *Clifford Cocks* numa entrevista com *Simon Singh* no documentário da BBC *"The Science of Secrecy – Going Public"*, inventara uma variante do actual criptossistema de chave pública RSA 1.2. E em 1974, *Malcolm Williamson*, encontra uma solução para o problema de troca de chaves, variante da solução de o actual *Diffie* e *Hellman*.

Foram precisos quase 30 anos para que, a 17 de Dezembro de 1997, numa sessão surpresa que decorreu na 6º conferência IMA (Institute for Mathematics and its Applications) em Cirencester, se apresentasse esta versão dos factos pelo próprio *Clifford Cocks*.

1.1.1 Como funciona o método de chave pública

Imaginemos que o Pedro e o João pretendem comunicar em segurança num determinado canal público, na presença de terceiros. Antes de mais, cada indivíduo tem de gerar um par de chaves.

O Pedro gera o par (C_P, D_P), enquanto que o João gera o par (C_J, D_J). Com a quantidade C_P, denominada a *chave pública*, o João ou *qualquer outra pessoa* pode cifrar uma mensagem M, ou seja, pode-se efectuar com facilidade o cálculo $m = f_{C_P}(M)$ a enviar ao Pedro. Após receber a mensagem, o Pedro vai utilizar a quantidade D_P que *só ele* pode possuir, denominada a *chave privada*, para decifrar a mensagem m e reaver a mensagem original M. Ou seja, o Pedro inverte o processo do João efectuando o cálculo $f_{D_P}(m) = f_{D_P}(f_{C_P}(M)) = M$. Note-se que se não conhecermos a quantidade D_P, então

[1] http://www.cesg.gov.uk/

a função inversa de $f_{C_P}(M)$ é extremamente difícil de calcular!

Emerge agora uma grande vantagem deste método: a chave que permitiu ao Pedro decifrar a mensagem m (a *chave privada* do Pedro correspondente à sua *chave pública*) não é comprometida, pois esta nunca saiu da posse do seu legítimo proprietário. Contrariamente, a *chave pública* deve ser divulgada e disponibilizada num servidor público acessível para todos, como exemplifica a tabela 1.1, ou como uma lista telefónica. Por exemplo, idap://keyserver.pgp.com e http://pgpkeys.mit.edu:11371 são servidores usados pelo *PGP v7.0.3*, onde é possível disponibilizar chaves públicas. Fica assim eliminado o *problema de distribuição de chaves.* O número de chaves a guardar reduz drasticamente, basta guardar a nossa chave pública e privada, pois todas as restantes chaves públicas necessárias para comunicar são de domínio público! Esta assimetria entre a chave pública e

Nome	Email	Chave pública
João	joao@email.com	C_J
Pedro	pedro@email.com	C_P
...	...	...

Tabela 1.1: *Servidor de chaves públicas.*

privada, nos criptossistemas de chave pública, deu origem ao que chamamos de *criptografia assimétrica.*

Nos criptossistemas clássicos, o conhecimento que permite cifrar dados é equivalente ao conhecimento que permite decifrar. O mesmo não acontece nos criptossitemas de chave pública, no qual o universo das chaves é tão grande que impossibilita uma busca exaustiva, assim como é impraticável obter a chave privada D_P do Pedro, a partir da chave pública C_P, sem uma demorosa e proibitiva série de cálculos. Também deve ser impraticável calcular a mensagem original M a partir da mensagem cifrada m e da chave pública C_P, sem o conhecimento da chave privada D_P. Por outras palavras, a função de cifragem deve ser uma "*função de um só sentido*".

Existem vários criptossistemas de chave pública, classificados quanto às variáveis:

(i) Variável única:

(a) Em $\mathbb{Z}_n$ com n grande: RSA (ver §1.2), Rabin [21].

(b) Em grupos mais complexos: Curvas elípticas [21].

(c) Em pequenos corpos finitos: HFE (Hidden Fields Equations) [8].

(ii) Várias variáveis lineares:

(a) Em $\mathbb{Z}$: a "mochila" de *Merkle-Hellman* (ver §1.3).

(b) Em pequenos corpos finitos: a "mochila" de *Chor-Rivest* [21].

(iii) Várias variáveis quadráticas:

(a) Em $\mathbb{Z}_n$ com n grande.

(b) Em pequenos corpos finitos.

Note-se que os esquemas de assinatura podem ser construídos a partir de qualquer um dos criptossistemas acima.

As funções de cifragem da maior parte dos criptossistemas de chave pública surgem de problemas computacionais com um grau de complexidade alto, problemas de cálculo com reputação de serem difíceis, embora, por vezes, a totalidade da complexidade do problema seja desconhecida. Por exemplo: a segurança do sistema de cifragem RSA assenta no problema da factorização de números inteiros. Pode acontecer que uma função, hoje, suposta ser de um só sentido possa perder este *estatuto* daqui a algum anos devido aos avanços tecnológicos

Neste trabalho, vamos considerar dois sistemas de cifragem mais pormenorizadamente, um deles baseado no *problema da factorização de números inteiros*, o RSA (§1.2), e outro baseado no *problema da soma de subconjuntos*, ou *problema da mochila* ("*knapsack*") 1.3.

1.1.2 Um pouco de Teoria da Complexidade

O principal objectivo da teoria da complexidade é fornecer mecanismos de *classificação de problemas computacionais*, tendo em conta os recursos necessários para os resolver. Essa classificação não deve depender de um modelo computacional em particular, mas sim da dificuldade intrínseca do problema. Os recursos medidos podem incluir tempo, espaço de armazenamento, número de processadores etc., mas, de um modo geral, o recurso mais focado é o tempo.

1.1.2.1 Notação assimptótica

Um *algoritmo* é um procedimento computacional *bem definido* para resolver um problema que assume variáveis de entrada (os dados) e pára com uma variável de saída (a resposta). O *tamanho* de uma variável de entrada n, inteiro, é aproximadamente $\lfloor \log_2(n) \rfloor + 1$ (número de bits em representação binária). Aqui, para não envolver demasiados termos técnicos para os nossos propósitos, podemos pensar num *algoritmo* como um programa de computador, escrito numa linguagem de programação específica, para um computador específico, que assume uma variável de entrada e pára com uma variável de saída.

É geralmente difícil calcular exactamente o tempo gasto na realização de operações por um certo algoritmo ou, de modo mais simplista, o seu "*custo*". Assim, somos levados a procurar aproximações que normalmente derivam do estudo *assimptótico* do custo de um algoritmo. Isto é, o estudo do crescimento do tempo de realização das operações de um algoritmo em função do crescimento (sem limite) dos valores de entrada. É de notar que não estamos a ter em conta outros factores que influenciam o custo de um algoritmo como, por exemplo, o tempo de acesso à memória, que pode depender da arquitectura física específica da máquina utilizada.

Definição 1.1.1 (Notação de ordem) *Sejam $f, g : \mathbb{N} \to \mathbb{R}$ com $g(n) \geq 0$ para todo o $n \in \mathbb{N}$. Diremos que:*

(i) (Majoração assimptótica) $f(n) = O(g(n))$ se existem constantes $c > 0$ e $n_0 \in \mathbb{N}$ tais que $0 \leq f(n) \leq cg(n)$ para todo $n \geq n_0$.

(ii) (Minoração assimptótica) $f(n) = \Omega(g(n))$ se existem constantes $c > 0$ e $n_0 \in \mathbb{N}$ tais que $0 \leq cg(n) \leq f(n)$ para todo $n \geq n_0$.

(iii) (Limitação assimptótica) $f(n) = \Theta(g(n))$ se existem constantes $c_1 > 0$ e $c_2 > 0$ tais que $c_1 g(n) \leq f(n) \leq c_2 g(n)$ para todo $n \geq n_0$.

Intuitivamente, $f(n) = O(g(n))$ significa que f não cresce assimptoticamente mais rápido do que um múltiplo de $g(n)$, enquanto que $f(n) = \Omega(g(n))$ significa que f cresce pelo menos tão rápido assimptoticamente que um múltiplo de $g(n)$.

Definição 1.1.2 *Diz-se que um algoritmo tem um* custo polinomial *se, no pior dos casos, o seu custo é da forma $O(n^k)$, onde n é o tamanho dos dados de entrada e k é constante. Um algoritmo que não pode ser assim majorado diz-se de* custo exponencial.

Os algoritmos de custo polinomial são considerados *bons* ou *eficientes*, enquanto que os algoritmos de custo exponencial são ditos *ineficientes*. Analogamente, se um problema computacional pode ser resolvido em tempo polinomial, este diz-se *fácil* ou *tratável*.

Por exemplo: existem algoritmos eficientes que implementam um corrector ortográfico. Por isso, programas comerciais de processamento de texto podem implementar estes algoritmos.

Para exemplificar uma majoração do "custo" de um algoritmo vamos estudar uma implementação eficiente do algoritmo de *Euclides* e do algoritmo extendido de *Euclides*[2].

1.1.2.2 Algoritmo de Euclides

O *algoritmo de Euclides* aparece em *"Elementos" de Euclides* nas proposições 1 e 2 do Livro VII usado para calcular o máximo divisor comum de dois inteiros. Também aparece nas proposições 2 e 3 do Livro X da mesma obra para encontrar a maior medida comum de duas quantidades, bem como para determinar a sua incomensurabilidade. Vamos começar por definir o máximo divisor comum de dois números.

Definição 1.1.3 *Dizemos que d é o máximo divisor comum de dois números inteiros a e b, se*

(i) $d \mid a$ *e* $d \mid b$;

(ii) se existir c tal que $c \mid a$ *e* $c \mid b$, *então* $c \mid d$.

Se d for o máximo divisor comum de a e de b escrevemos $d = (a, b)$.

O próximo Teorema prova a existência e unicidade do máximo divisor comum de dois números.

Teorema 1.1.1 *Sejam a e b inteiros não nulos. O máximo divisor comum de a e de b existe e é único.*

[2]Podemos encontrar várias implementações eficientes do algoritmo extendido de *Euclides* em [7, 9].

Demonstração. Seja

$$S = \{ax + by \mid x, y \in \mathbb{Z}\}.$$

É fácil ver que S é não vazio e contém inteiros. Seja d o mais pequeno inteiro positivo em S. Então $d = ax + by$, para algum x e y. Queremos provar que $d = (a, b)$. Se $a = dq + r$, $0 \leq r < d$, então

$$r = a - dq = a - axq - byq = a(1 - xq) + b(-yq) \in S$$

Como d é o mais pequeno inteiro em S, $r < d$ e $r \in S$, então $r = 0$. Isto implica que $d \mid a$. Analogamente se vê que $d \mid b$.

Se $c \mid a$ e $c \mid b$, para algum c, então $c \mid ax + by = d$. Logo $d = (a, b)$. Se existir outro máximo divisor comum de a e b, ou seja, $d' = (a, b)$, obtemos $d \mid d'$ e $d' \mid d$, logo $d = d'$. □

Da prova do Teorema 1.1.1 segue que o seguinte resultado.

Corolário 1.1.2 *Sejam a e b inteiros não nulos. Existem inteiros x e y tais que*

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

O *algoritmo de Euclides* permite calcular eficientemente o máximo divisor comum de dois inteiros e baseia-se no seguinte resultado.

Teorema 1.1.3 *Se a e b são dois inteiros positivos, então $(a, b) = (b, a \pmod b)$.*

Demonstração. Se $d = (a, b)$, então $d \mid a$ e $d \mid b$. Existem inteiros k_1 e k_2 tais que $a = dk_1$ e $b = dk_2$. Pelo algoritmo da divisão existe um inteiro q para o qual $a = qb + (a \pmod b)$. Logo, $a \pmod b = a - bq = dk_1 - dk_2q = d(k_1 - k_2q)$, ou seja, d divide $a \pmod b$. Como $d \mid b$, então $d \mid (b, a \pmod b)$.

Suponhamos que $d' = (b, a \pmod b)$, isto é, $d' \mid b$ e $d' \mid a \pmod b$. Por um lado $d' \mid (a + k'b)$, para $k' \in \mathbb{Z}$. Por outro lado, existem inteiros k_1 e k_2 tais que $d'k_1 = b$ e $d'k_2 = a + bk'$. Logo $a = d'k_2 - k'b = d'k_2 - d'k_1k' = d'(k_2 - k_1k')$. Portanto $d' \mid a$. Como $d' \mid b$, então $d' \mid (a, b)$. Concluímos que $d \mid d'$ e $d' \mid d$, logo, $d = d'$. □

Exemplo 1.1.1 Vamos calcular $(30, 18)$. Pelo Teorema 1.1.3, temos

$$(30, 18) = (18, 30 \pmod{18}) = (18, 12) = (12, 6) = (6, 0) = 6.$$

Em cada iteração, se $b \neq 0$, o *algoritmo de Euclides* substitui a por b e b por $a \pmod{b}$. Assim que $b = 0$, o algoritmo pára e retorna o inteiro a. Vamos provar que o *algoritmo de Euclides* termina sempre ($b = 0$) e que o valor de a na última iteração é máximo divisor comum de a e b.

Vamos traduzir o *algoritmo de Euclides* por uma notação recursiva. Seja r_n *sequência dos restos*: $r_0 = a$, $r_1 = b$ e, para cada $k \geq 2$, se $r_{k-1} \neq 0$ definimos

$$r_k = r_{k-2} \pmod{r_{k-1}}.$$

A sequência $r_2, r_3, \ldots$ é a sequência dos restos calculados no *algoritmo de Euclides*. Ao aplicar o Teorema 1.1.3 k vezes, não alteramos o máximo divisor comum de a e b, e obtemos $d = r_k$ se $r_{k+1} = 0$. Só falta provar que existe um k tal que $r_{k+1} = 0$.

Temos que $r_k \leq 0$ para um dado k, e que $0 \leq r_{k+1} \leq r_k - 1$. Logo, como o resto r_k decresce pelo menos uma unidade em cada iteração, ao fim de $b - 1 = r_1 - 1$ passos o processo pára.

Vamos estender o *algoritmo de Euclides* de maneira que o algoritmo também permita escrever $d = (a, b)$ como combinação linear de a e de b, ou seja, forneça inteiros x e y tais que $d = ax + by$.

Tal como no *algoritmo de Euclides*, definimos a sequência dos restos $r_0, r_1, r_2, \ldots, r_{k+1}$. Vamos supor que $r_{k+1} = 0$ e, portanto, que $d = (a, b) = r_k$. Seja q_n o quociente de r_{n-1} por r_n, ou seja, $r_{n-1} = q_n r_n + r_{n+1}$ para n entre 1 e k. Recursivamente, a sequência dos quocientes é definida por

$$q_n = \left\lfloor \frac{r_{n-1}}{r_n} \right\rfloor, 1 \leq n \leq k.$$

Vamos explicar como podemos construir duas sequências x_k e y_k, de maneira que $x = (-1)^k x_k$ e $y = (-1)^{k+1} y_k$ sejam os coeficientes procurados. Começamos por fazer

$$x_0 = 1, \quad x_1 = 0, \quad y_0 = 0, \quad y_1 = 1,$$

e definimos recursivamente

$$x_n = q_{n-1} \cdot x_{n-1} + x_{n-2}, \quad y_n = q_{n-1} \cdot y_{n-1} + y_{n-2}, \quad 2 \leq n \leq k + 1.$$

Podemos dispor as 4 sequências na tabela 1.2 com 4 linhas e $k + 1$ colunas. Para facilitar a nossa discussão, supomos que $a > b$ e tomamos $r_0 = a$ e $r_1 = b$.

O próximo resultado garante a construção de d como combinação linear de a e b.

	0	1	2	...	k	$k+1$
r_n	$r_0 = a$	$r_1 = b$	r_2	...	r_k	0
q_n	×	q_1	q_2	...	q_k	×
x_n	1	0	x_2	...	x_k	x_{k+1}
y_n	0	1	y_2	...	y_k	y_{k+1}

Tabela 1.2: *Implementação do algoritmo extendido de Euclides.*

Teorema 1.1.4 *Temos que* $r_n = (-1)^n\, x_n a + \ (-1)^{n+1} y_n b$, *para* n *entre* 0 *e* $k+1$.

Demonstração. É fácil ver que

$$a = r_0 = 1 \cdot a - 0 \cdot b = x_0 \cdot a - y_0 \cdot b$$
$$b = r_1 = -0 \cdot a - 1 \cdot b = -x_1 \cdot a + y_1 \cdot b$$

mostrando que a relação é verdadeira para $n = 0$ e para $n = 1$.

Concluímos a prova do Teorema por indução sobre n. Seja $n \geq 2$ e suponha-se que a relação é válida para todos os valores de 0 a $n-1$. Então, pela definição dos quocientes q_n e dos restos r_n,

$$\begin{aligned} r_n &= r_{n-2} - q_{n-1} r_{n-1} \\ &= (-1)^{n-2}\, x_{n-2} a + \ (-1)^{n-1} y_{n-2} b - q_{n-1} \left((-1)^{n-1}\, x_{n-1} a + \ (-1)^n y_{n-1} b \right) \\ &= (-1)^n a \left(q_{n-1} x_{n-1} + x_{n-2} \right) \ + \ (-1)^{n+1} b \left(q_{n-1} y_{n-1} + y_{n-2} \right) \\ &= (-1)^n a x_n \ + \ (-1)^{n+1} b y_n. \end{aligned}$$

Ficou provado que $r_n = (-1)^n\, x_n a + \ (-1)^{n+1} y_n b$, $0 \leq n \leq k+1$. □

Em particular, o Teorema 1.1.4 dá-nos uma representação do máximo divisor comum $d = r_k$ como combinação linear de a e de b, obtendo

$$r_k = (-1)^k x_k\, a \ + \ (-1)^{k+1} y_k\, b.$$

Exemplo 1.1.2 Vamos calcular máximo divisor comum entre 752 e 193.

	0	1	2	3	4	5	6	7	8
r_n	752	193	173	20	13	7	6	1	0
q_n	×	3	1	8	1	1	1	6	×
x_n	1	0	1	1	9	10	19	29	×
y_n	0	1	3	4	35	39	74	113	×

Observando a tabela, é fácil ver que $(752, 193) = 1$ e que

$$\begin{aligned} 1 &= (-1)^7 \cdot 29 \cdot 752 + (-1)^8 \cdot 113 \cdot 193 \\ &= -29 \cdot 752 + 113 \cdot 193. \end{aligned}$$

Custo do algoritmo Extendido de *Euclides* Vamos agora responder a pergunta: quantas iterações são necessárias efectuar até que o algoritmo páre, ou seja, qual o valor de k?

Um vez que $r_i = r_{i-2} \pmod{r_{i-1}}$, temos forçosamente $r_0 > r_1 > \ldots > r_k$. Por outro lado, temos que $r_i = q_{i+1} \cdot r_{i+1} + r_{i+2} \geq r_{i+1} + r_{i+2} > 2 \cdot r_{i+2}$. Portanto, $1 \leq r_k < \frac{r_0}{2^{\lfloor \frac{k}{2} \rfloor}}$, ou seja, $\log_2 r_0 > \lfloor \frac{k}{2} \rfloor$. O número de iterações necessárias para que o algoritmo pare é igual a $O(\log r_0)$. Finalmente, podemos enunciar o resultado que estabelece o custo de esta implementação do *algoritmo extendido de Euclides.*

Teorema 1.1.5 *Sejam a e b inteiros e $m = \max(a, b)$. O custo do algoritmo extendido de* Euclides *aplicado a a e b é, no máximo, igual a $O(\log^3 m)$.*

Demonstração. Neste algoritmo realizamos, no máximo, $O(\log m)$ iterações. Em cada iteração são feitas no máximo 7 operações com números menores ou iguais a m. Destas operações[3], as que são de maior custo são as divisões e as multiplicações cujo custo é igual a $O(\log^2 m)$. □

1.1.2.3 Exponenciação modular rápida

Veremos neste trabalho que muitos sistema de cifragem dependem do cálculo de $a^e \pmod{m}$, com módulo m e expoente e bastante grandes. É por isso, importante dispor de um método rápido para efectuar a exponenciação modular. Descrevemos agora um método rápido de efectuar este cálculo.

Seja

$$e = \sum_{i=0}^{k} e_i 2^i$$

a representação binária de e. Os coeficientes e_i são 0 ou 1. Logo,

$$a^e \equiv a^{\sum_{i=0}^{k} e_i 2^i} \equiv \prod_{i=0}^{k} \left(a^{2^i}\right)^{e_i} \equiv \prod_{0 \leq i \leq k,\ e_i = 1} a^{2^i} \pmod{m}$$

[3]Em [7, 22], estabelece-se e prova-se o custo de operações aritméticas básicas tais como a adição, a multiplicação e operações modulares.

A partir de esta formula podemos obter o seguinte algoritmo:

(i) Calcular os sucessivos quadrados a^{2^i}, $0 \leq i \leq k$. Note-se que $a^{2^{i+1}} = \left(a^{2^i}\right)^2$, logo podemos calcular os sucessivos quadrados a^{2^i} recursivamente.

(ii) Determinar a^e como o produto de a^{2^i} quando $e_i = 1$.

Exemplo 1.1.3 Vamos calcular $6^{73} \pmod{100}$. A representação binária do expoente é $73 = 1 + 2^3 + 2^6$. Determinamos os sucessivos quadrados de 6 módulo 100: 6, $6^2 = 36$, $6^{2^2} = -4 \pmod{100}$, $6^{2^3} = 16 \pmod{100}$, $6^{2^4} = 56 \pmod{100}$, $6^{2^5} = 36 \pmod{100}$, $6^{2^6} = 16 \pmod{100}$. Calculamos $6^{73} \equiv 6 \cdot 6^{2^3} \cdot 6^{2^6} \equiv 6 \cdot 16 \cdot (-4) \equiv 16 \pmod{100}$. Neste cálculo, só efectuamos 6 quadrados e duas multiplicações modulares. É muito melhor do que as 72 multiplicações modulares do método tradicional!

Com este método, apenas são necessários, no máximo, $\lfloor \log_2 e \rfloor$ quadrados modulares e $\lfloor \log_2 e \rfloor$ multiplicações modulares, em vez de e multiplicações modulares! Cada multiplicação modular tem custo $O(\log^2 m)$ (ver [7, 22]). Portanto, se e é um inteiro e $a \in \{0, \ldots, m-1\}$, então o cálculo de $a^e \pmod{m}$ tem custo $O(\log e \cdot \log^2 m)$.

1.1.2.4 Classes de complexidade

Quando enfrentamos um problema computacional coloca-se uma questão importante: existe algum algoritmo rápido para resolver o problema?

Na teoria da complexidade computacional concentramos a nossa atenção nos problemas de decisão, isto é, problemas que têm como resposta um simples SIM ou NÃO. Na prática, isto não restringe demasiado o âmbito do estudo, uma vez que os problemas que serão abordados podem ser enunciados como problemas de decisão.

Definição 1.1.4 *A classe de complexidade $\mathcal{P}$ é o conjunto de todos os problemas de decisão que se podem resolver em tempo polinomial. Caso contrário diz-se não-$\mathcal{P}$.*

Os problemas não-$\mathcal{P}$ são problemas difíceis. Para mostrar que um problema é não-$\mathcal{P}$, seria preciso mostrar que qualquer algoritmo que o resolva é não polinomial, tarefa que deve ser muito difícil! Talvez por isso, não se conhece ainda nenhum problema que se tenha demonstrado não-$\mathcal{P}$. No entanto, podemos estudar uma classe de complexidade intermédia: a classe $\mathcal{NP}$.

Definição 1.1.5 *A classe de complexidade $\mathcal{NP}$ é o conjunto de todos os problemas de decisão para os quais se pode verificar uma resposta SIM em tempo polinomial, com a ajuda de alguma informação extra: um certificado.*

Definição 1.1.6 *A classe de complexidade co-$\mathcal{NP}$ é o conjunto de todos os problemas de decisão para os quais se pode verificar uma resposta NÃO em tempo polinomial, usando um certificado apropriado.*

Note-se que o facto de um problema estar em $\mathcal{NP}$ (ou co-$\mathcal{NP}$) não implica que seja fácil obter um certificado; o que acontece é que, se tal certificado existir, então pode ser usado eficientemente para verificar a resposta SIM (respectivamente NÃO).

O problema fundamental da teoria da complexidade é o célebre $\mathcal{P} = \mathcal{NP}$. O problema consiste em determinar se as duas classes de problemas coincidem. Este é um dos maiores problemas em aberto da matemática e faz parte de um conjunto de sete problemas designados pelo instituto Clay como "os problemas do milénio". Quem solucionar o problema $\mathcal{P} = \mathcal{NP}$ será recompensado pelo instituto com um prémio de um milhão de dólares!

Este problema é de uma importância extrema para a criptografia, porque a maioria dos criptossistema de chave pública são baseados em problemas $\mathcal{NP}$. Quem provar que $\mathcal{P} = \mathcal{NP}$ anula a segurança de todas comunicações levando rapidamente o nosso mundo ao caos (a não ser que a segurança das comunicações seja assegurada por outros meios).

Exemplo 1.1.4 (Problema COMPOSIÇÃO) Dado um inteiro n, n é composto? Ou seja, existem inteiros $a, b > 1$ tal que $n = ab$?
Este problema pertence a $\mathcal{NP}$, porque é possível verificar em tempo polinomial se n é composto, dado um divisor a de n, onde $1 < a < n$. Aqui o certificado é o divisor a. Ainda não é conhecido se este problema está em $\mathcal{P}$, ou seja, se podemos factorizar qualquer número inteiro em tempo polinomial.

Definição 1.1.7 *Sejam A e B dois problemas de decisão. Diz-se que $A \leq_P B$, se existe um algoritmo de custo polinomial que resolve A, usando um hipotético algoritmo de custo polinomial que resolva B.*

Dizer que $A \leq_P B$, significa que B é pelo menos tão difícil quanto A. Assim, se A é um problema computacional considerado intratável, então mostrando que $A \leq_P B$, obtemos

uma forte evidência de que também B é intratável. Se $A \leq_P B$ e $B \leq_P A$, então A e B dizem-se *computacionalmente equivalentes*, simbolicamente $A \equiv_P B$. Numa linguagem mais simples, $A \equiv_P B$ significa que A e B são ambos tratáveis ou são ambos intratáveis.

Estamos em condições de definir uma classe de problemas muito importantes, a classe de problemas $\mathcal{NP}$–completos.

Definição 1.1.8 *Um problema de decisão L é $\mathcal{NP}$–completo se:*

(i) $L \in \mathcal{NP}$, *e*

(ii) $A \leq_P L$ *para todo* $A \in \mathcal{NP}$

$\mathcal{NPC}$ é a classe de todos os problemas $\mathcal{NP}$–completos.

Os problemas $\mathcal{NP}$–completos são os mais difíceis em $\mathcal{NP}$, no sentido que são pelo menos tão difíceis quanto todos os problemas de $\mathcal{NP}$. Um outro aspecto importante é que cada um destes problemas é totalmente representativo da sua classes, isto é, pode mostrar-se que a solução de qualquer problema $\mathcal{NP}$ se pode construir a partir dele em tempo polinomial.

Exemplo 1.1.5 (Problema da soma de subconjuntos) Dado um conjunto de inteiros $\{a_1, a_2, \ldots, a_n\}$ e um inteiro s, determinar se existe ou não um subconjunto $\{a_{i_1}, \ldots, a_{i_j}\}$ de $\{a_1, a_2, \ldots, a_n\}$ tal que $s = \sum_{j=1}^{k} a_{i_j}$. Este problema é $\mathcal{NP}$–completo.

Em Março de 2000, *Richard Kaye*[4] prova que a consistência do jogo de computador "*Minesweeper*" é $\mathcal{NP}$–completo, ou seja, dada uma configuração arbitrária de minas e bandeiras do "*Minesweeper*", decidir se essa configuração é possível é um problema $\mathcal{NP}$–completo. Este resultado é bastante importante, pois se alguém conseguir construir um algoritmo polinomial para resolver o problema da consistência do "*Minesweeper*", então esse algoritmo poderá resolver indirectamente qualquer problema $\mathcal{NP}$. Isto significa que se prova que $\mathcal{P} = \mathcal{NP}$ e se ganhou um milhão de dólares!

Vamos, por último, ver uma classificação de problemas que não se restringe aos problemas de decisão. Esta classificação engloba os problemas $\mathcal{NP}$–completos.

[4] http://mat.bham.ac.uk/R.W.Kaye/minesw/minesw.htm

Definição 1.1.9 *Diz-se que um problema L é $\mathcal{NP}$–difícil se existe um problema de decisão A, $\mathcal{NP}$–completo, tal que $A \leq_P L$.*

Exemplo 1.1.6 (Problema $\mathcal{NP}$–difícil) Seja A o problema da soma de subconjuntos (exemplo 1.1.5). Considere o problema L: "Dado um conjunto de inteiros $\{a_1, a_2, \ldots, a_n\}$ e um inteiro s, *encontre* um subconjunto $\{a_{i_1}, \ldots, a_{i_j}\}$ de $\{a_i\}_i$ tal que $s = \sum_{j=1}^{k} a_{i_j}$". O problema L é $\mathcal{NP}$–difícil, uma vez que A é um problema de decisão $\mathcal{NP}$–completo e $A \leq_P L$. Note-se que no problema $\mathcal{NP}$–difícil L, não se pretende somente verificar se existe mas *encontrar* um subconjunto $\{a_{i_1}, \ldots, a_{i_j}\}$ de $\{a_i\}_i$ tal que $s = \sum_{j=1}^{k} a_{i_j}$.

1.2 Sistema de cifragem RSA

> *"Em 1977, três pessoas fizeram a mais singular e espectacular contribuição para a criptografia de chave pública:* ***Ronald Rivest, Adi Shamir*** *e* ***Leonard Adleman****... tomaram o desafio de produzir um criptossistema de chave pública totalmente desenvolvido... Em Maio de 1977, eles foram recompensados com o sucesso... Eles haviam descoberto como um pouco de Teoria de Números poderia ser utilizada para resolver o problema."* [10]

O RSA é o criptossistema de chave pública mais usado para a cifragem de dados e assinatura. É baseado no problema intratável da *factorização de inteiros.* O problema consiste em encontrar a factorização em números primos de um certo inteiro n, isto é, encontrar primos distintos p_i e inteiros $e_i > 0$ tais que, $n = p_1^{e_1} p_2^{e_2} \cdots p_1^{e_1}$.

De um modo geral, o problema de verificar se um dado número inteiro é primo, é um problema extremamente mais fácil do que o problema da factorização de um inteiro como veremos mais adiante no capítulo 3 (ver também o exemplo 1.1.4).

Vamos descrever como funciona o sistema de cifragem RSA.

Tal como o sistema de Chave Pública, para cifrar e decifrar mensagem com o sistema de cifragem RSA é necessário possuir um par de chaves.

Geração das chaves *pública* e *privada* Cada entidade cria um par de chaves: uma chave pública para cifrar e a uma chave privada para decifrar.

(i) Geramos dois números aleatórios primos p e q, suficientemente grandes (cerca de 300 dígitos decimais), distintos e de comprimento "aproximadamente" igual, mas a diferença $p - q$ não deve ser muito pequena, ver 3.3.1. Por outro lado, se $p \approx q$,

então $p \approx \sqrt{n}$. Isto permite factorizar n efectuando divisões sucessivas por todos os primos próximos de $\sqrt{n}$.

(ii) Calculamos $n = p \times q$ e $\varphi(n) = \varphi(pq) = (p-1)(q-1)$ (φ é a *função de Euler*, ver definição 1.2.1).

(iii) Geramos um número aleatório inteiro c (c de codificador), de tal forma que c e $\varphi(n)$ sejam primos entre si,

$$(c, \varphi(n)) = 1 \quad (\Leftrightarrow (c, p-1) = (c, q-1) = 1), \quad 1 \leq c \leq \varphi(n)$$

(iv) Calculamos, usando o algoritmo extendido de Euclides, o único inteiro d (d de descodificador) tal que,

$$c \times d \equiv 1 \pmod{\varphi(n)}, \qquad 1 < d < \varphi(n)$$

isto é, d é o inverso de c módulo $\varphi(n)$. Na secção 1.1.2.2 estudamos uma implementação eficiente do algoritmo extendido de *Euclides*.

Os números aleatórios p, q e c, são escolhidos com ajuda de um *gerador de números pseudoaleatórios*, um programa de computador que produz uma sequência de dígitos de tal forma que o número gerado não é previsível, não pode ser duplicado e possui todas as propriedades estatísticas de números aleatórios verdadeiros. Estes geradores são analisados com mais detalhe no capítulo 2.

Os números c e d são chamados *expoente de cifragem* e *expoente de decifragem*, respectivamente, enquanto que n é chamado de *módulo*. A chave pública é constituída pelo par (n, c) e a chave privada por $(n, p, q, \varphi(n), d)$; os primos p e q e $\varphi(n)$ não são necessários para decifrar mensagens, mas podem ser usados para acelerar os cálculos de exponenciação (ver §1.2.3).

Cifragem: Suponhamos que o Pedro (P) pretende enviar uma mensagem cifrada ao João (J). Seja agora M a mensagem numérica, ou parte da mensagem a ser cifrada.

(i) Gerar a chave pública e privada (n_J, c_J) do João.

(ii) M deve ser de tal forma que $(M, n) = 1$ e $M < n$. Se $M \geq n$, basta partir M em blocos M_i onde $M_i < p$ e $M_i < q$.

(iii) Calcular:

$$m \equiv M^{c_J} \pmod{n_J}.$$

Na secção 1.1.2.3 analisamos um método que permite calcular $M^{c_J} \pmod{n_J}$ muito rapidamente.

Agora o João pode receber a mensagem m cifrada com a chave pública (n_J, c_J). A mensagem m pode ser enviada por canal aberto de comunicações, pois o João é o único a possuir o *expoente de decifragem* d_J. Assim, mesmo que a mensagem cifrada seja interceptada por uma terceira pessoa, essa não conseguirá descobrir o conteúdo da mensagem cifrada.

Decifragem: Para que o João recupere a mensagem original M a partir de m, basta calcular $M \equiv m^{d_J} \pmod{n_J}$. Podemos colocar uma questão natural: será que a mensagem decifrada é sempre igual à original?

O seguinte Teorema permite-nos garantir o funcionamento biunívoco do sistema de cifragem RSA.

Teorema 1.2.1 (RSA) *Sejam (n, c) a chave pública de cifragem e (n, d) a chave privada de cifragem. Seja M um inteiro menor que n. Se $m \equiv M^c \pmod{n}$, então $M \equiv m^d \pmod{n}$.*

Para que possamos compreender a demonstração deste resultado, apresentamos na secção seguinte resultados indispensáveis da Teoria de Números.

1.2.1 Um pouco de Teoria de Números

Vamos começar com a definição de *grupo*. Um grupo G é um conjunto de elementos munido de uma operação binária $\oplus$ que satisfaz as seguintes propriedades: a operação binária é associativa, tem elemento neutro e e para todo o elemento $a \in G$ existe um elemento inverso denotado a^{-1}.

Exemplo 1.2.1 Sendo p é um inteiro, considere o grupo $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$, munido da operação binária $a \oplus b \stackrel{\text{def}}{=} a + b \pmod{p}$.

Definimos a *ordem* do grupo G, denotado por $|G|$, como o número de elementos que existem em G. Usamos a notação h^m para representar a operação $\oplus$ repetida m vezes com o elemento h: $h \oplus \cdots \oplus h$. A ordem de um elemento h num grupo G é o mais pequeno inteiro m tal que $h^m = 1$.

Um *grupo cíclico* é um grupo que contém um elemento g, chamado *gerador* do grupo, tal que $G = \{g, g^2, \ldots, g^m\}$, onde $m = |G|$, e g^m é identidade do grupo.

Exemplo 1.2.2 O grupo $\mathbb{Z}_p^* = \{1 \leq x \leq p : (x,p) = 1\}$ munida da operação multiplicação módulo p, ou seja, $a \oplus b \stackrel{\text{def}}{=} a \cdot b \pmod{p}$, é um grupo cíclico.

1.2.1.1 Funções aritméticas multiplicativas

Um conjunto de inteiros $\{a_1, a_2, \ldots, a_n\}$ diz-se um *sistema reduzido de resíduos módulo* m, se satisfaz as seguinte condições:

(i) Cada a_i é primo com m, $i = 1, \ldots, n$.

(ii) Se $i \neq j$, então $a_i \not\equiv a_j \pmod{m}$.

(iii) Se $(a, m) = 1$, então $a \equiv a_i \pmod{m}$, para algum $i \in \{1, 2, \ldots, n\}$.

Exemplo 1.2.3 $\{1, 3, 5, 9, 11, 13, 15, 17, 19, 23, 25, 27\}$ é um sistema reduzido de resíduos módulo 28.

Teorema 1.2.2 *Se* $\{a_1, a_2, \ldots, a_n\}$ *é um sistema reduzido de resíduos módulo* m *e se* $(a, m) = 1$, *então* $\{aa_1, aa_2, \ldots, aa_n\}$ *é também um sistema reduzido de resíduos módulo* m.

Demonstração. Vamos ver que cada uma das condições (i), (ii) e (iii) são satisfeitas pelo conjunto $\{aa_1, aa_2, \ldots, aa_n\}$. Suponhamos que $(a, m) = 1$.

(i) Cada aa_i é primo com m, porque se não fosse, então existiria um primo p tal que $p \mid m$ e $p \mid aa_i$. Mas se $p \mid aa_i$, então $p \mid a$ ou $p \mid a_i$, ou seja, $(a, m) \neq 1$ ou $(a_i, m) \neq 1$ o que contraria as hipóteses.

(ii) Se $aa_i \equiv aa_j \pmod{m}$ e $i \neq j$, então $a_i \equiv a_j \pmod{m}$ visto que $(a, m) = 1$, ou seja, $\{a_1, a_2, \ldots, a_n\}$ não seria um sistema reduzido de resíduos módulo m.

(iii) Se b é um inteiro tal que $(b, m) = 1$, como $(a, m) = 1$ a equação Diofantina $ax + my = b$ tem soluções inteiras x e y. Pode ver-se facilmente que $(x, m) = 1$, senão existiria algum primo $p \mid x$ e $p \mid m$, logo p dividiria b e, consequentemente, não teríamos $(b, m) = 1$. Visto que $(x, m) = 1$ e que $\{a_1, a_2, \ldots, a_n\}$ é um sistema reduzido de resíduos módulo m, resulta que $x \equiv a_i \pmod{m}$, para algum i entre 1 e n, ou seja, $x = a_i + km$. Logo $a(a_i + km) + my = b$, ou equivalentemente, $aa_i + akm + ym = b$, o que significa que $b \equiv aa_i \pmod{m}$, para algum i entre 1 e n.

□

Todo o sistema reduzido de resíduos módulo m tem o mesmo número de elementos que a ordem do grupo cíclico $\mathbb{Z}_m^*$, tantos quantos os elementos que são primos com m e não excedem m.

Definição 1.2.1 (Função de Euler) *Para cada número natural m definimos $\varphi(m)$, a* função de Euler, *como sendo o número de inteiros positivos que não excedem m e são primos com m.*

Exemplo 1.2.4 Tendo em conta o exemplo 1.2.3, $\varphi(28) = 12$. Se p é primo, então $\varphi(p) = p - 1$, porque $\{1, 2, \ldots, p - 1\}$ é um sistema reduzido de resíduos módulo p.

O Teorema 1.2.4 é uma ferramenta muito importante para efectuar o cálculo de $\varphi(m)$. Mas primeiro vamos estabelecer dois resultados necessários para a prova de vários resultados.

Proposição 1.2.1 *Se $(a, b) = 1$ e $a \mid bc$, então $a \mid c$.*

Demonstração. Se $(a, b) = 1$, então existem inteiros x e y tal que $ax + by = 1$. Ao multiplicar a igualdade anterior por c obtemos $axc + byc = c$. Mas $a \mid axc$ e como $a \mid bc$, então $a \mid bcy$. Logo $a \mid c$ □

Corolário 1.2.3 *Se $(a, m) = 1$ e $ax \equiv ay \pmod{m}$, então $x \equiv y \pmod{m}$.*

Demonstração. Se $ax \equiv ay \pmod{m}$, então $m \mid (ax - ay)$ ou $m \mid a(x - y)$. Como $(a, m) = 1$, pela proposição 1.2.1 temos $m \mid (x - y)$, ou seja, $x \equiv y \pmod{m}$. □

Teorema 1.2.4 *Se m e n são inteiros positivos tais que $(m, n) = 1$, então $\varphi(mn) = \varphi(m)\varphi(n)$.*

Nota: Dizemos que um função aritmética f é multiplicativa se para todos inteiros m, n, se $(m,n) = 1$, $f(mn) = f(m)f(n)$. Pelo Teorema 1.2.4 φ é uma função multiplicativa.

Demonstração. Suponhamos que $(m,n) = 1$. Sejam $M = \{a_1, a_2, \ldots, a_{\varphi(m)}\}$ e $N = \{b_1, b_2, \ldots, b_{\varphi(N)}\}$ sistemas de reduzidos de resíduos módulo m e n, respectivamente. Seja T um sistema reduzido de resíduos módulo mn.

Nestas condições podemos estabelecer o seguinte resultado,

Lema 1.2.5 *$(x, mn) = 1$ se, e só se, $(x,m) = 1$ e $(x,n) = 1$.*

Demonstração do lema 1.2.5. Se $(x, mn) = 1$, então $(x,m) = 1$ porque se $(x,m) \neq 1$ existiria um número primo p tal que $p \mid x$ e $p \mid m$. Mas se $p \mid m$ também $p \mid mn$. Isto é absurdo, pois $p \mid x$ e $p \mid mn$ contradiz $(x, mn) = 1$. Analogamente se prova que $(x,n) = 1$.

Reciprocamente, se $(x,m) = 1$ e $(x,n) = 1$, então $(x, nm) = 1$. Suponhamos por absurdo que $(x, nm) \neq 1$. Logo existe um número primo p tal que $p \mid x$ e $p \mid nm$. Mas $p \mid mn$ implica que $p \mid m$ ou $p \mid n$. Por um lado, se $p \mid m$, como p também divide x, obtemos que $(x,m) \neq 1$. Por outro lado, se $p \mid n$, então $(x,n) \neq 1$. Uma contradição em ambos casos pois inicialmente supomos que $(x,m) = 1$ e $(x,n) = 1$. □

Seja x um elemento de T. Como T é um sistema reduzido de resíduos temos que $(x, mn) = 1$, logo $(x,m) = 1$ e $(x,n) = 1$. Por definição, existe um, e um só, $a_i \in M$ e existe um, e um só, $b_j \in N$ tais que $x \equiv a_i \pmod{m}$ e $x \equiv b_j \pmod{n}$.

Podemos considerar a função,

$$f: \begin{array}{ccc} T & \longrightarrow & M \times N \\ x_{ij} & \longmapsto & (a_i, b_j) \end{array}, \quad \text{com } 1 \leq i \leq \varphi(m) \text{ e } 1 \leq j \leq \varphi(n).$$

Vamos provar que f é bijectiva. Seja $(a_i, b_j) \in M \times N$ e mostramos que existe algum $x \in T$ para o qual se tem $x \equiv a_i \pmod{m}$ e $x \equiv b_j \pmod{n}$, ou seja, vamos mostrar que existe algum $x \in T$ para o qual se tem $x = a_i + hm$ e $x = b_j + kn$, com h e k inteiros.

Como $(m,n) = 1$, a equação Diofantina

$$hn - km = a_i - b_j$$

tem soluções em h e k, quaisquer que sejam a_i e b_j, uma vez que $(m,n) \mid (a_i - b_j)$. Logo f é sobrejectiva.

Por outro lado, seja $y = a_i + hm = b_j + kn$, com h e k inteiros, que existem pelo parágrafo anterior. Resulta que $(y, m) = 1$ e $(y, n) = 1$. Se $(y, m) \neq 1$, existiria algum número primo p tal que $p \mid y$ e $p \mid m$, o que implica que $p \mid y - hm = a_i$. Um absurdo, pois $(a_i, m) = 1$. De forma análoga se vê que $(y, n) = 1$.

Pelo lema 1.2.5 temos que $(y, mn) = 1$. Como T é sistema reduzido de resíduos módulo mn, existe um, e um só, $x \in T$ tal que $x \equiv y \pmod{mn}$.

Focou provado que f é bijectiva, portanto, $\varphi(mn) = \varphi(m)\varphi(n)$. □

Do Teorema 1.2.4 resulta, por indução, que se $m_1, m_2, \ldots, m_k$ são k inteiros primos entre si dois a dois, então $\varphi(m_1 m_2 \cdots m_k) = \varphi(m_1)\varphi(m_2) \cdots \varphi(m_k)$. Se $m = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$, onde os inteiros p_i são primos distintos, então tem-se

$$\varphi(m) = \varphi(p_1^{\alpha_1})\varphi(p_2^{\alpha_2}) \cdots \varphi(p_k^{\alpha_k}). \tag{1.1}$$

Naturalmente, este tipo de decomposição vale para qualquer função aritmética multiplicativa.

Como $\varphi(p_i^{\alpha_i})$ é o número de inteiros positivos que não excedem $p_i^{\alpha_i}$ e são primos com $p_i^{\alpha_i}$, isto é, que não são divisíveis por p_i, então $\varphi(p_i^{\alpha_i}) = p_i^{\alpha_i} - \frac{p_i^{\alpha_i}}{p_i} = p_i^{\alpha_i - 1}(p_i - 1)$. Logo a *função de Euler* pode ser calculada mediante a decomposição em factores primos do número m, substituindo cada $\varphi(p_i^{\alpha_i})$, na equação 1.1, por $p_i^{\alpha_i - 1}(p_i - 1)$,

$$\varphi(m) = p_1^{\alpha_1 - 1}(p_1 - 1) p_2^{\alpha_2 - 1}(p_2 - 1) \cdots p_k^{\alpha_k - 1}(p_k - 1). \tag{1.2}$$

1.2.1.2 O Pequeno Teorema de Fermat

O *Pequeno Teorema de Fermat* está contido numa carta dirigida por *Pierre Fermat* a *Bernard Frénicle de Bessy*, datada de 18 de Outubro de 1640. Nessa carta, *Fermat* não enviou a demonstração do referido Teorema, e os seus correspondentes não mostraram interesse em conhecer a demonstração. Talvez por isso, não conhecemos como é que *Fermat* chegou à formulação do seu Teorema.

A primeira demonstração do *Pequeno Teorema de Fermat* foi publicada, por *Euler*, nos *Proceedings* da academia de S. Petersburgo em 1736. Posteriormente, em 1760, *Euler* generaliza o *Pequeno Teorema de Fermat* demonstrando o Teorema 1.2.6 (*Teorema de Euler*). Este Teorema traduz uma importante propriedade da *função de Euler*. É uma

propriedade importante para o sistema de cifragem RSA, visto que a prova do Teorema 1.2.1 é uma aplicação directa do *Teorema de Euler.*

Teorema 1.2.6 (Teorema de Euler) *Se a e m são inteiros, com m positivo, tais que $(a, m) = 1$, então $a^{\varphi(m)} \equiv 1 \pmod{m}$.*

Demonstração. Seja $\{a_1, a_2, \ldots, a_{\varphi(m)}\}$ um sistema reduzido de resíduos módulo m. Como $(a, m) = 1$, pelo Teorema 1.2.2 sabemos que $\{aa_1, aa_2, \ldots, aa_{\varphi(m)}\}$ é também um sistema reduzido de resíduos módulo m. Logo,

$$\begin{aligned} & aa_1aa_2 \cdots aa_{\varphi(m)} \equiv a_1a_2 \cdots a_{\varphi(m)} \pmod{m}, \\ \Leftrightarrow \quad & a^{\varphi(m)}a_1a_2, \cdots a_{\varphi(m)} \equiv a_1a_2 \cdots a_{\varphi(m)} \pmod{m}. \end{aligned}$$

Como $(a_1, m) = 1$ e $(a_2, m) = 1$, ou seja, $a_1x_1 + my_1 = 1$ e $a_2x_2 + my_2 = 1$, para inteiros x_1, x_2, y_1 e y_2, então

$$(a_1x_1 + my_1) \cdot (a_2x_2 + my_2) = 1 \Leftrightarrow a_1a_2(x_1x_2) + m(a_1x_1y_2 + a_2x_2y_1 + my_1y_2) = 1,$$

ou seja, $(a_1a_2, m) = 1$. De novo, como $(a_1a_2, m) = 1$ e $(a_3, m) = 1$, então $(a_1a_2a_3, m) = 1$. Repetindo o processo sucessivamente para cada elemento do sistema reduzido de resíduos $\{a_1, a_2, \ldots, a_{\varphi(m)}\}$, obtemos $(a_1a_2a_3 \ldots a_{\varphi(m)}, m) = 1$.

Temos $a^{\varphi(m)}a_1a_2 \cdots a_{\varphi(m)} \equiv a_1a_2 \cdots a_{\varphi(m)} \pmod{m}$, então $m \mid a_1a_2 \cdots a_{\varphi(m)}(a^{\varphi(m)} - 1)$. Como $(a_1a_2a_3 \ldots a_{\varphi(m)}, m) = 1$, pelo lema 1.2.1 temos $m \mid (a^{\varphi(m)} - 1)$, ou seja, $a^{\varphi(m)} \equiv 1 \pmod{m}$. □

Teorema 1.2.7 (Pequeno Teorema de Fermat) *Se p é um número primo e a não é divisível por p, então $a^{p-1} \equiv 1 \pmod{p}$.*

Demonstração. Como p é um número primo, então $\varphi(p) = p - 1$. Também temos que $(a, p) = 1$ é equivalente a $p \nmid a$, logo pelo Teorema 1.2.6, $a^{p-1} \equiv 1 \pmod{p}$. □

Exemplo 1.2.5 O número 1223 é primo e 5 não é divisível por 1223, então pelo Teorema 1.2.7 resulta que $5^{1222} \equiv 1 \pmod{1223}$.

1.2.1.3 Raízes primitivas

Consideremos m um inteiro maior do que 1 e a um inteiro primo com m. Definimos $r = \text{ord}_m(a)$, a *ordem de a módulo m*, como o menor inteiro r tal que $a^r \equiv 1 \pmod{m}$.

Pelo Teorema de *Euler*, temos que $a^{\varphi(m)} \equiv 1 \pmod{m}$. Daqui resulta que a ordem de a é menor ou igual a $\varphi(m)$, para todo a primo com m.

Definição 1.2.2 *Diz-se que g é uma raiz primitiva de m se* $\text{ord}_m(a) = \varphi(m)$.

Notemos que existe uma raiz primitiva módulo m se, e só se, $\mathbb{Z}_m^*$ é um grupo cíclico.

Exemplo 1.2.6 Os números 2 e 7 são raízes primitivas de 11, porque $\text{ord}_{11}(2) = \text{ord}_{11}(7) = \varphi(11) = 10$. O número 12 não tem raízes primitivas.

Algumas propriedades de $\text{ord}_m(a)$:

(i) Se $\text{ord}_m(a) = r$, então $r \mid \varphi(m)$.

Pelo algoritmo da divisão, temos $\varphi(m) = rq + s$, com $0 \leq s < r$. Se $s > 0$, então $1 \equiv a^{\varphi(m)} \equiv a^{rq+s} \equiv (a^r)^q \cdot a^s \equiv a^s \pmod{m}$, com $s < r$. Isto contradiz a hipótese de $\text{ord}_m(a) = r$.

(ii) Se $\text{ord}_m(a) \mid k$ se, e só se, $a^k \equiv 1 \pmod{m}$.

Suponhamos que $k = \text{ord}_m(a) \cdot q + r$, com $0 \leq r < \text{ord}_m(a)$. Como $a^k = a^{\text{ord}_m(a) \cdot q + r} \equiv a^r \pmod{m}$, com $r < \text{ord}_m(a)$, vem que $a^k \equiv 1 \pmod{m}$ equivale a $r = 0$, ou seja, $\text{ord}_m(a) \mid r$.

(iii) Se $\text{ord}_m(a) = r$ e i, j são inteiros não negativos, então tem-se $a^i \equiv a^j \pmod{m}$ se, e só se, $i \equiv j \pmod{r}$.

Se $a^i \equiv a^j \pmod{m}$ e $i > j$, então tem-se $a^{i-j} \equiv 1 \pmod{m}$, logo $r \mid i - j$, ou seja, $i \equiv j \pmod{r}$. Inversamente, se $i \equiv j \pmod{r}$, então $a^{i-j} \equiv 1 \pmod{m}$, portanto, $a^i \equiv a^j \pmod{m}$.

(iv) $\text{ord}_m(a) = \varphi(m)$, ou seja, a é um raiz primitiva de m se, e só se, $\{a, a^2, \ldots, a^{\varphi(m)}\}$ é um sistema reduzido de resíduos módulo m.

(v) Se $\text{ord}_m(a) = r$ e k é um inteiro positivo, então $\text{ord}_m(a^k) = \frac{r}{(k,r)}$.

Seja $d = (r,k)$, $k = dq_1$ e $r = dq_2$, com $(q_1, q_2) = 1$. Queremos mostrar que $\text{ord}_m(a^k) = q_2$. Seja $\text{ord}_m(a^k) = s$.

Por um lado, $a^{ks} \equiv (a^k)^s \equiv 1 \pmod{m}$, logo $r \mid ks$ ou $dq_2 \mid dq_1 s$, ou seja, $q_2 \mid q_1 s$. Como $(q_1, q_2) = 1$ vem que $q_2 \mid s$.

Por outro lado, $(a^k)^{q_2} \equiv a^{kq_2} \equiv a^{dq_1 q_2} \equiv (a^r)^{q_1} \equiv 1 \pmod{m}$. Daqui resulta que $s \mid q_2$. Ficou provado que $s = q_2$.

Nota: a^k tem a mesma ordem módulo m que a se, e só se, $(k,r) = 1$.

Os próximos resultados são essenciais para enunciar Teorema das raízes primitivas que nos dá informações acerca da existência de raízes primitivas para um inteiro m.

Inteiros que têm raiz primitiva Dizer que g é uma raiz primitiva de m, é dizer que $\text{ord}_m(g) = \varphi(m)$, ou, por outras palavras, que o conjunto

$$\left\{g, g^2, g^3, \ldots, g^{\varphi(m)}\right\}$$

é um sistema reduzido de resíduos módulo m.

Para os inteiros que não têm raiz primitiva não é possível escrever os elementos de um sistema reduzido de resíduos como potência de um elemento.

Lema 1.2.8 *Se $k \geq 3$, então não existe nenhuma raiz primitiva módulo 2^k.*

Demonstração. Basta mostrar, por indução sobre n, que para todo o inteiro ímpar a, $a^{2^{n-2}} \equiv 1 \pmod{2^n}$, porque $\varphi(n) = 2^{n-1}$.

Claro que é verdade para $n = 3$. Suponhamos que é verdade para $n = k$, ou seja, para todo o inteiro ímpar a, $a^{2^{k-2}} = 2^k b + 1$ para algum inteiro b. Elevando ambos membros da equação ao quadrado, obtemos $a^{2^{k-1}} = 2^{k+1}\left(2^{k-1}b^2 + b\right) + 1 \equiv 1 \pmod{2^{k+1}}$, o que completa a prova. □

Lema 1.2.9 *Se $m \geq 3$ e $n \geq 3$ são inteiros primos entre si, $(m,n) = 1$, então não existem raízes primitivas módulo mn.*

Demonstração. Seja $d = (\varphi(m), \varphi(m))$ e $k = [\varphi(m), \varphi(m)]$. Por um lado, para todo o inteiro a primo com mn, $(a, mn) = 1$, pelo Teorema de *Euler* temos,

$$a^k \equiv 1 \pmod{m} \quad \text{e} \quad a^k \equiv 1 \pmod{n}.$$

Por outro lado, como $(m, n) = 1$, o Teorema Chinês dos Restos implica que

$$a^k \equiv 1 \pmod{mn}. \tag{1.3}$$

Notando que $\varphi(m)$ e que $\varphi(n)$ são inteiros pares, temos $k = \frac{\varphi(m)\varphi(n)}{d} = \frac{\varphi(mn)}{d} \leq \frac{\varphi(mn)}{2}$. Logo, a congruência 1.3 implica que não existem raízes primitivas módulo mn. □

Dos dois Lemas anteriores resulta que, se $m > 1$ é um inteiro que tem alguma raiz primitiva, então m não é o produto de dois factores, primos entre si e maiores que 2, e nem é uma potência de 2 maior que 4.

Os inteiros que podem ter alguma raiz primitiva são precisamente 2, 4, p^α e $2p^\alpha$, com p primo e ímpar e α inteiro positivo.

É imediato que 2 tem 1 como raiz primitiva e que 4 tem 3 como raiz primitiva. Resta ver os casos p^α e $2p^\alpha$.

Primeiro, vamos provar o resultado auxiliar seguinte.

Teorema 1.2.10 (Teorema de Gauss) *Se n é um inteiro positivo, então*

$$\sum_{d|n} \varphi(d) = n$$

Demonstração. Vejamos primeiro que se $d \mid n$, então há $\varphi(\frac{n}{d})$ elementos do conjunto $S = \{1, 2, \ldots, n\}$ cujo máximo divisor comum com n é d.

O subconjunto de S cujos elementos são divisíveis por d é $S' = \{d, 2d, \ldots, \frac{n}{d}d\}$. Seja $kd \in S'$. Então $(kd, n) = (kd, \frac{n}{d}d) = d \cdot (k, \frac{n}{d})$. Logo, $(kd, n) = d$ se, e só se, $(k, \frac{n}{d}) = 1$. Ou seja, o número de inteiros existentes em S cujo máximo divisor comum com n é igual a d, é igual ao número de inteiros k que não excedem $\frac{n}{d}$ e são primos com $\frac{n}{d}$. Tal número é $\varphi(\frac{n}{d})$.

Como cada um dos inteiros de S tem, para máximo divisor comum com n, um divisor d de n, então

$$n = \sum_{d|n} \varphi\left(\frac{n}{d}\right).$$

Mas, temos $d \mid n$ se, e só se, $\frac{n}{d} \mid n$, isto é, se os divisores positivos de n são $d_1, \ldots, d_t$, então os divisores positivos de n são $\frac{n}{d_t}, \ldots, \frac{n}{d_1}$.

Logo,

$$n = \sum_{d|n} \varphi(d).$$

□

Vejamos que, se p é primo, então existem raízes primitivas módulo p.

Lema 1.2.11 *Seja p um número primo ímpar e $d \mid p-1$. Então há $\varphi(d)$ inteiros incongruentes módulo p, cuja ordem é d.*

Demonstração. Seja a é um número inteiro. Se $\mathrm{ord}_p(a) = d$, então pela propriedade (i) $d \mid p-1$. Consideremos o conjunto $S = \{a, a^2, \ldots, a^d\}$, os seus elementos têm ordem $\mathrm{ord}_p(a^k) = \frac{d}{(k,d)}$, para $1 \leq k \leq d$, pela propriedade (vi). Como $\mathrm{ord}_p(a^k) = d$ se, e só se, $(k,d) = 1$, se existir um elemento com ordem d, então o número de elementos de S cuja ordem é igual a d é $\varphi(d)$.

Falta provar que se $d \mid p-1$, então há pelo menos um inteiro a tal que $\mathrm{ord}_p(a) = d$.

Sejam $d_1, d_2, \ldots, d_t$ todos os divisores de $p-1$ e, para cada d_i, seja $\psi(d_i)$ o número de elementos do conjunto $R = \{1, 2, \ldots, p-1\}$ cuja ordem módulo p é d_i. (Todo o elemento de R é primo com p e tem ordem módulo p que divide $p-1$). Portanto,

$$\sum_{i=1}^{t} \psi(d_i) = p-1.$$

Mas pelo Teorema de Gauss 1.2.10, temos

$$\sum_{i=1}^{t} \varphi(d_i) = p-1,$$

isto é,

$$\sum_{i=1}^{t} \psi(d_i) = \sum_{i=1}^{t} \varphi(d_i). \tag{1.4}$$

Por outro lado, sabemos que para cada d_i se tem $\psi(d_i) = 0$ ou $\psi(d_i) = \varphi(d_i)$. Pela equação 1.4, resulta que se tem necessariamente $\psi(d_i) = \varphi(d_i)$, para cada d_i. Caso contrário, teríamos

$$\sum_{i=1}^{t} \psi(d_i) < \sum_{i=1}^{t} \varphi(d_i).$$

Logo, há $\varphi(d)$ elementos cuja ordem é d. □

Em particular, se p é primo, o Lema anterior diz que há $\varphi(p-1)$ raízes primitivas de p.

Lema 1.2.12 *Seja p é um número primo ímpar. Então existem raízes primitivas g módulo p tais que*

$$g^{p-1} \not\equiv 1 \pmod{p^2}. \tag{1.5}$$

Demonstração. Seja g uma raiz primitiva de p tal que $g^{p-1} \equiv 1 \pmod{p^2}$ e consideremos o inteiro $g+p$.

É claro que o inteiro $g+p$ é raiz primitiva de p. Temos

$$\begin{aligned}(g+p)^{p-1} &= g^{p-1} + pg^{p-2} + \binom{p-1}{2}p^2g^{p-3} + \cdots + p^{p-1} \\ &\equiv 1 - pg^{p-2} \pmod{p^2}.\end{aligned}$$

Como $(g,p) = 1$, tem-se que $pg^{p-2} \not\equiv 0 \pmod{p^2}$ e, consequentemente, $(g+p)^{p-1} \not\equiv 1 \pmod{p^2}$. □

Lema 1.2.13 *Se g é uma raiz primitiva módulo p^k e*

$$g^{\varphi(p^k)} \not\equiv 1 \pmod{p^{k+1}}, \tag{1.6}$$

então g é uma raiz primitiva módulo p^{k+1}, e

$$g^{\varphi(p^{k+1})} \not\equiv 1 \pmod{p^{k+2}}.$$

Demonstração. Seja n a ordem de g módulo p^{k+1}. Visto que $g^{\varphi(p^{k+1})} \equiv 1 \pmod{p^{k+1}}$, vem que $n \mid \varphi(p^{k+1}) = p^k(p-1)$. Por outro lado, $g^n \equiv 1 \pmod{p^{k+1}}$ implica que $g^n \equiv 1 \pmod{p^k}$. Como g é uma raiz primitiva módulo p^k, vemos que $\varphi(p^k) = p^{k-1}(p-1) \mid n$. Portanto, n é igual a $\varphi(p^k)$ ou $\varphi(p^{k+1})$.

Mas, pela equação 1.6 segue que a ordem de g é $\varphi(p^{k+1})$, ou seja, g é uma raiz primitiva módulo p^{k+1}. □

Corolário 1.2.14 *Se p é um número primo ímpar, então para todo $k \geq 1$ existem raízes primitivas módulo p^k.*

Demonstração. O corolário resulta dos lemas anteriores por indução sobre k. □

Corolário 1.2.15 *Se p é um primo ímpar, então para todo $k \geq 1$ existem raízes primitivas módulo $2p^k$.*

Demonstração. Seja g uma raiz primitiva módulo p^k. Consideremos que g é ímpar. Seja $n = \text{ord}_{2p^k}(g)$. Então $n \mid \varphi(2p^k) = \varphi(p^k)$.

Por outro lado, $g^n \equiv 1 \pmod{2p^k}$, o que implica que $g^n \equiv 1 \pmod{p^k}$. Como g é uma raiz primitiva módulo p^k, $\varphi(p^k) \mid n$. Isto implica que $n = \varphi(2p^k)$, ou seja, que g é uma raiz primitiva módulo $2p^k$. □

Finalmente, os resultados anteriores permitem enunciar o Teorema das raízes primitivas.

Teorema 1.2.16 (Teorema das raizes primitivas) *Seja m um inteiro positivo. Então uma raiz primitiva módulo m existe se, e só se, m é igual a 2, 4, p^k ou $2p^k$, em que p é um número primo maior que 2 e k é um inteiro positivo.*

Seja m um inteiro que tem alguma raiz primitiva g. Pela propriedade (iv), $\{g, g^2, \ldots, g^{\varphi(m)}\}$ é um sistema reduzido de resíduos módulo m. Se a é um inteiro primo com m, então existe um, e um só, inteiro i tal que

$$a \equiv g^i \pmod{m}, \quad 0 \leq i \leq \varphi(m) - 1.$$

Definição 1.2.3 *Definimos o índice de a com respeito à raiz primitiva g, denotado* $\text{ind}_g\, a$, *ao menor inteiro não negativo i tal que $a \equiv g^i \pmod{m}$.*

1.2.1.4 Lei da reciprocidade quadrática

Seja a e m inteiros tais que $(a, m) = 1$.

Definição 1.2.4 *Se a congruência $x^2 \equiv a \pmod{m}$ tem solução, diz-se que a é* resíduo quadrático módulo m. *Se a congruência não tem solução, diz-se que a é um* não resíduo quadrático módulo m.

Se a não é primo com m, então diz-se que a nem é resíduo quadrático, nem é não resíduo quadrático.

Símbolo de *Legendre* *Adrien-Marie Legendre* (1752-1833) introduziu um símbolo para exprimir o carácter quadrático de um inteiro com respeito a um primo.

Definição 1.2.5 *Seja p um inteiro primo ímpar. Para um inteiro a, definimos o símbolo de Legendre por*

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{se } p \mid a, \\ 1 & \text{se } a \text{ é resíduo quadrático módulo } p, \\ -1 & \text{se } a \text{ é não resíduo quadrático módulo } p. \end{cases}$$

O próximo resultado, descoberto por *Euler*, permite calcular o *símbolo de Legendre.*

Teorema 1.2.17 (Critério de Euler) *Sejam a um inteiro e p um primo, tais que $(a,p) = 1$. Então*

$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod{p}.$$

Demonstração. Note-se que se p é primo, p admite uma raiz primitiva g. A congruência

$$x^n \equiv a \pmod{p} \tag{1.7}$$

é equivalente à congruência $n \operatorname{ind}_g x \equiv \operatorname{ind}_g a \pmod{\varphi(p)}$, ou seja, a congruência 1.7 tem solução se, e só se, $(n, \varphi(p)) \mid \operatorname{ind}_g a$.

Seja $d = (n, \varphi(p))$, então $n = dq_1$, $\varphi(p) = dq_2$ e $(q_1, q_2) = 1$. A congruência

$$a^{\frac{\varphi(p)}{(n,\varphi(p))}} = a^{q_2} \equiv 1 \pmod{p}$$

tem solução se, e só se, $q_2 \operatorname{ind}_g a \equiv 0 \pmod{\varphi(p)}$. Isto significa que $\varphi(p) = dq_2 \mid q_2 \operatorname{ind}_g a$, ou seja, $d \mid \operatorname{ind}_g a$. Como vimos, uma condição necessária e suficiente para que a congruência 1.7 tenha solução é que $(n, \varphi(p)) \mid \operatorname{ind}_g a$. Logo, concluímos que $x^n \equiv a \pmod{n}$ tem solução se, e só se $a^{\frac{\varphi(p)}{(n,\varphi(p))}} \equiv 1 \pmod{p}$.

Para o *critério de Euler* consideramos $n = 2$ e $\varphi(p) = p - 1$. Se a é *resíduo quadrático*, ou seja, $\left(\frac{a}{p}\right) = 1$, temos $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$.

Suponhamos que a é não resíduo quadrático. Como $p-1$ é par temos $(2, p-1) = 2$. Se $(a,p) = 1$, pelo *Teorema de Fermat* temos $a^{p-1} \equiv 1 \pmod{p}$ ou, equivalentemente, $a^{p-1} - 1 \equiv \left(a^{\frac{p-1}{2}} - 1\right)\left(a^{\frac{p-1}{2}} + 1\right) \equiv 0 \pmod{p}$.

Podemos garantir que $\left(a^{\frac{p-1}{2}}-1\right) \equiv 0$ ou $\left(a^{\frac{p-1}{2}}+1\right) \equiv 0$. Além disto, estas congruências não são simultaneamente satisfeitas porque, se o fossem, $2 \equiv 0 \pmod{p}$, o que é falso pois p é um primo ímpar.

Portanto, a é *não resíduo quadrático*, temos forçosamente $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$. □

Vamos introduzir algumas propriedades que permitem auxiliar o cálculo do símbolo de *Legendre*.

Teorema 1.2.18 *Seja p um primo. Pelo critério de* Euler *e pela definição do símbolo de* Legendre, *seguem as seguintes propriedades:*

(1) Se $a \equiv b \pmod{p}$, então $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$.

(2) $\left(\frac{a^2}{p}\right) = 1$.

(3) $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$.

(4) $$\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} = \begin{cases} +1 & \text{se } p \equiv 1 \pmod{4}, \\ -1 & \text{se } p \equiv 3 \pmod{4}. \end{cases}$$

Demonstração. Para provar 1) basta observar que $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \equiv b^{\frac{p-1}{2}} \equiv \left(\frac{b}{p}\right) \pmod{p}$. Na propriedade 2, é óbvio que a^2 é resíduo quadrático módulo p porque, pelo Teorema de *Fermat*, $(a^2)^{\frac{p-1}{2}} \equiv a^{p-1} \equiv 1 \pmod{p}$. As propriedades 3 e 4 seguem do critério de *Euler*, pois

$$\left(\frac{ab}{p}\right) \equiv (ab)^{\frac{p-1}{2}} = a^{\frac{p-1}{2}} \cdot b^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right)\left(\frac{b}{p}\right) \pmod{p}$$

e $\left(\frac{-1}{p}\right) \equiv (-1)^{\frac{p-1}{2}} \pmod{p}$, pelo critério de *Euler*.

Como $\left(\frac{ab}{p}\right), \left(\frac{a}{p}\right), \left(\frac{b}{p}\right)$ e $\left(\frac{-1}{p}\right)$ só tomam valores ± 1, podemos tomar a igualdade nas congruências anteriores, porque $p \geq 3$.

Notemos que $\left(\frac{-1}{p}\right) = 1$ se $\frac{p-1}{2}$ é par. Isto acontece se existe um inteiro n tal que $\frac{p-1}{2} = 2n$, ou seja, se $p = 1 + 4n$. Analogamente se vê que $\left(\frac{-1}{p}\right) = -1$ se $\frac{p-1}{2}$ é ímpar, ou seja, se p é da forma $3 + 4n$. □

Outras propriedades muito importantes:

(5) Se p é primo, então $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$.

(6) Lei da reciprocidade quadrática (Legendre-Gauss):

Se p e q são primos ímpares, então

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}} = \begin{cases} +1 & \text{se } p \equiv 1 \pmod 4 \text{ ou } q \equiv 1 \pmod 4, \\ -1 & \text{se } p \equiv q \equiv 3 \pmod 4. \end{cases}$$

Para demonstrar a lei da reciprocidade quadrática vamos começar por demonstrar alguns resultados essenciais e o Teorema de *Eisenstein.*

Teorema 1.2.19 (Lema de Gauss) *Se q é um inteiro não divisível pelo primo ímpar p, então $\left(\frac{q}{p}\right) = (-1)^k$, onde k é o número de resíduos positivos módulo p existentes no conjunto*

$$\left\{q, 2q, 3q, \ldots, \tfrac{p-1}{2}q\right\}, \tag{1.8}$$

que são maiores que $\frac{p}{2}$.

Por exemplo, consideremos $\left(\frac{4}{11}\right)$. Como $\frac{p-1}{2} = 5$, o conjunto 1.8 é $\{4, 8, 12, 16, 20\}$ ou $\{4, 8, 1, 5, 9\}$. Temos dois elementos maiores que 5.5 (8 e 9, portanto $\left(\frac{4}{11}\right) = (-1)^2 = 1$.

Demonstração do Lema de Gauss. Comecemos por observar que os elementos do conjunto 1.8 são todos não nulos. Se $aq \equiv 0 \pmod p$, para $a \in \{1, 2, \ldots, \frac{p-1}{2}\}$, então $p \mid aq$. Uma contradição porque $a < p$ e $p \nmid q$.

Os elementos do conjunto 1.8 são incongruentes dois a dois módulo p. Se $aq \equiv bq \pmod p$, para $a, b \in \left\{1, 2, \ldots, \frac{p-1}{2}\right\}$, então $p \mid (a-b)q$. Como $p \nmid q$, vem que $p \mid (a-b)$, ou seja, $a \equiv b \pmod p$, o que é impossível.

Consideremos a seguinte reordenação do conjunto 1.8,

$$\{r_1, r_2, \ldots, r_h, s_1, s_2, \ldots, s_k\}, \quad \text{com } h + k = \tfrac{p-1}{2}, \tag{1.9}$$

formada por resíduos positivos módulo p, em que os inteiros r_i são menores que $\frac{p}{2}$ e os inteiros s_j são maiores que $\frac{p}{2}$.

Considerando

$$\{r_1, r_2, \ldots, r_h, p - s_1, p - s_2, \ldots, p - s_k\}, \tag{1.10}$$

obtemos um conjunto de elementos positivos menores que $\frac{p}{2}$. Quaisquer dois elementos do conjunto 1.10 são incongruentes módulo p. Por um lado, quando $i \neq j$, temos $p - s_i \not\equiv p - s_j$

$(\text{mod } p)$ e $r_i \not\equiv r_j \pmod{p}$, porque os elementos do conjunto 1.8 são incongruentes módulo p. Por outro lado, temos $p - s_i \not\equiv r_j \pmod{p}$. Caso contrário, $s_j + r_i \equiv 0 \pmod{p}$ e como cada elemento do conjunto 1.9 é congruente módulo p com algum elemento do conjunto 1.8, obtemos

$$s_j + r_i \equiv aq + bq \equiv (a+b)q \equiv 0 \pmod{p}, \quad a, b \leq \frac{p-1}{2}.$$

Por hipótese, $p \nmid q$, logo

$$p \mid (a+b) \leq \frac{p-1}{2} + \frac{p-1}{2} = p - 1.$$

Isto é impossível! Portanto, os elementos do conjunto 1.10 são incongruentes módulo p dois a dois.

Note-se que o conjunto 1.10 tem $\frac{p-1}{2}$ elementos positivos e menores que $\frac{p}{2}$. Isto implica que os elementos do conjunto 1.10 são forçosamente os elementos $1, 2, \ldots, \frac{p-1}{2}$.

Multiplicando todos os elementos do conjunto 1.10 obtemos

$$\prod_{i=1}^{k}(p - s_i) \cdot \prod_{j=1}^{h} r_j \equiv \left(\frac{p-1}{2}\right)! \pmod{p}.$$

Como $p - s_i \equiv -s_i \pmod{p}$, vem que

$$(-1)^k \prod_{i=1}^{k}(s_i) \cdot \prod_{j=1}^{h} r_j \equiv \left(\frac{p-1}{2}\right)! \pmod{p},$$

e como cada elemento do conjunto 1.9 é congruente módulo p com algum elemento do conjunto 1.8, a congruência anterior é equivalente a

$$(-1)^k \left(\frac{p-1}{2}\right)! q^{\frac{p-1}{2}} \equiv \left(\frac{p-1}{2}\right)! \pmod{p}.$$

Da congruência anterior resulta que $(-1)^k q^{\frac{p-1}{2}} \equiv 1 \pmod{p}$.

Portanto, como $\left(\frac{q}{p}\right) \equiv q^{\frac{p-1}{2}} \pmod{p}$ (critério de *Euler*), ficou provado que $\left(\frac{q}{p}\right) = (-1)^k$. □

Consideremos

$$A = \sum_{i=1}^{h} r_i, \quad B = \sum_{j=1}^{k} s_j \quad \text{e} \quad M = \left[\frac{q}{p}\right] + \left[\frac{2q}{p}\right] + \cdots + \left[\frac{\frac{p-1}{2}q}{p}\right] = \sum_{i=1}^{\frac{p-1}{2}} \left[\frac{iq}{p}\right],$$

onde $\left[\frac{iq}{p}\right]$ designa o maior inteiro que não excede $\frac{iq}{p}$, ou seja, $iq = \left[\frac{iq}{p}\right] p + \varepsilon_i$, em que $0 \leq \varepsilon_i < p$, para $i \in \{1, 2, \ldots, \frac{p-1}{2}\}$.

Lema 1.2.20 *Se p e q são números ímpares diferentes, em que p é primo e não divide q, então*

$$\left(\frac{q}{p}\right) = (-1)^M.$$

Demonstração. A soma de todos os elementos do conjunto $\left\{q, 2q, \ldots, \frac{p-1}{2}q\right\}$ é igual a

$$\sum_{i=1}^{\frac{p-1}{2}} iq = \sum_{i=1}^{\frac{p-1}{2}} \left[\frac{iq}{p}\right] p + \sum_{i=1}^{\frac{p-1}{2}} \varepsilon_i, \quad 0 \leq \varepsilon_i < p.$$

Como $\sum_{i=1}^{\frac{p-1}{2}} i = \frac{1+\frac{p-1}{2}}{2} \cdot \frac{p-1}{2} = \frac{p^2-1}{8}$ e os números ε_i são os números $r_1, \ldots, r_h, s_1, \ldots, s_k$, então

$$\frac{p^2-1}{8}q = Mp + A + B. \tag{1.11}$$

Como os elementos do conjunto 1.10 são precisamente os números $1, 2, \ldots, \frac{p-1}{2}$, temos que

$$\sum_{i=1}^{h} r_i + \sum_{j=1}^{k} (p - s_j) = \sum_{l=1}^{\frac{p-1}{2}} i,$$

ou seja

$$A + kp - B = \frac{p^2-1}{8}. \tag{1.12}$$

Das equações 1.11 e 1.12, resulta que

$$\frac{p^2-1}{8}(q-1) = (M-k)p + 2B. \tag{1.13}$$

Por hipótese, q é um número ímpar, logo $\frac{p^2-1}{8}(q-1)$ é par. Portanto, $(M-k)p \equiv 0 \pmod 2$, ou seja, $M \equiv k \pmod 2$. Isto significa que k é par ou ímpar consoante M é par ou ímpar, respectivamente.

Como p não divide q, pelo Lema de *Gauss* concluímos que $\left(\frac{q}{p}\right) = (-1)^k = (-1)^M$. □

Teorema 1.2.21 (Propriedade 5) *Se p é um número primo ímpar, então*

$$\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}.$$

Demonstração. Usando as notações e a demonstração do Lema anterior, tomando $q = 2$ temos $M = \left[\frac{2}{p}\right] + \left[\frac{4}{p}\right] + \cdots + \left[\frac{p-1}{p}\right] = 0$. Resulta da equação 1.13 que

$$\frac{p^2-1}{8} \equiv -kp \pmod 2.$$

Isto significa que k é par ou ímpar consoante $\frac{p^2-1}{8}$ é par ou ímpar, respectivamente. Portanto, pelo Lema de *Gauss* $\left(\frac{2}{p}\right) = (-1)^k = (-1)^{\frac{p^2-1}{8}}$ □

Fazendo $N = \left[\frac{p}{q}\right] + \left[\frac{2p}{q}\right] + \cdots + \left[\frac{\frac{q-1}{2}p}{q}\right]$, vamos demonstrar o Teorema de *Eisenstein*, o último resultado essencial para a prova da *Lei da Reciprocidade Quadrática.*

Teorema 1.2.22 (Teorema de Eisenstein) *Se p e q são primos ímpares diferentes, então*

$$M + N = \frac{p-1}{2} \cdot \frac{q-1}{2}.$$

Demonstração. Consideremos a recta r, no plano cartesiano, definida pela equação $y = \frac{q}{p}x$. A recta r é representada na figura 1.1. Na porção da recta r contida dentro do

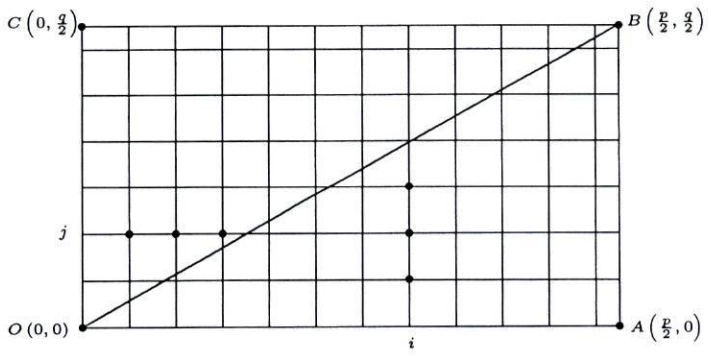


Figura 1.1: $y = \frac{q}{p}x$.

rectângulo $[OABC]$, não tem pontos cujas coordenadas sejam ambas números inteiros, pois q e p são primos entre si e a maior abcissa inteira contida no interior do rectângulo é menor que $\frac{p}{2}$. O número $\left[\frac{q}{p}i\right]$ é o maior inteiro contido na ordenada do ponto de abcissa i para a recta r. Logo $\left[\frac{q}{p}i\right]$ corresponde ao número de pontos de coordenadas inteiras que pertencem a recta $x = i$ e que ficam "abaixo" da recta r. Portanto, M é o número de pontos de coordenadas inteiras que estão contidos no interior do rectângulo $[OABC]$ e que estão "abaixo" da recta r.

Analogamente, N é o número de pontos de coordenadas inteiras que estão contidos no interior do rectângulo $[OABC]$ e que estão "acima" da recta r.

Visto que, o número total de pontos de coordenadas inteiras que estão contidos no interior do rectângulo $[OABC]$ é $\frac{p-1}{2} \cdot \frac{q-1}{2}$. Isto prova que $M + N = \frac{p-1}{2} \cdot \frac{q-1}{2}$. □

Agora, a *Lei da Reciprocidade Quadrática* é de demonstração imediata.

Demonstração da propriedade 6. Do Lema 1.2.20, resulta que

$$\left(\frac{q}{p}\right) = (-1)^M \quad \text{e} \quad \left(\frac{p}{q}\right) = (-1)^N,$$

logo, pelo Teorema 1.2.22,

$$\left(\frac{p}{q}\right) \cdot \left(\frac{q}{p}\right) = (-1)^{M+N} = (-1)^{\frac{p-1}{2} \cdot \frac{q-1}{2}}.$$

Notemos que $\frac{p-1}{2} \cdot \frac{q-1}{2}$ é par se, e só se, $\frac{p-1}{2} \cdot \frac{q-1}{2} \equiv 0 \pmod 2$. Isto acontece se $\frac{p-1}{2} \equiv 0 \pmod 2$ ou $\frac{q-1}{2} \equiv 0 \pmod 2$. Estas congruências são equivalentes a $\frac{p-1}{2} = 2n$ ou $\frac{q-1}{2} = 2m$, para algum inteiro n e m. Logo, $\left(\frac{p}{q}\right) \cdot \left(\frac{q}{p}\right) = 1$ se $p \equiv 1 \pmod 4$ ou $q \equiv 1 \pmod 4$ e $\left(\frac{p}{q}\right) \cdot \left(\frac{q}{p}\right) = -1$ se $p \equiv q \equiv 3 \pmod 4$. □

Símbolo de *Jacobi* Vamos agora definir um utensílio que nos permite rapidamente calcular $\left(\frac{n}{m}\right)$. No entanto, este símbolo é mais usado com números muito grandes, quando a factorização põe muitas dificuldades. Este obstáculo desaparece graças a *Carl Gustav Jacob Jacobi* (1804–1851).

Definição 1.2.6 *Seja n um inteiro e m um inteiro positivo ímpar, $m = p_1 \cdots p_r$, em que cada p_i é um primo ímpar que pode aparecer repetido. O símbolo de* Jacobi *é definido por*

$$\left(\frac{n}{m}\right) = \prod_{i=1}^{r} \left(\frac{n}{p_i}\right),$$

onde $\left(\frac{n}{p_i}\right)$ é o símbolo de Legendre.

Se $(m, n) = 1$ e $x^2 \equiv n \pmod m$ tem solução, então é natural que $\left(\frac{n}{m}\right) = 1$. No entanto, apesar de $\left(\frac{n}{m}\right) = 1$, n pode não ser um resíduo quadrático módulo m. Trata-se de uma conveniência para tratar factorizações. Notemos que se m é primo, o símbolo de *Jacobi* é idêntico ao símbolo de *Legendre*. E o aspecto mais interessante é que o símbolo de *Jacobi* satisfaz as mesmas regras computacionais que o símbolo de *Legendre*.

Teorema 1.2.23 *Sejam n e m inteiros positivos ímpares e primos entre si. Pelas propriedades do símbolo de* Legendre *e pela definição do símbolo de* Jacobi, *seguem as seguintes propriedades:*

(1) Se $n \equiv n' \pmod m$, então $\left(\frac{n}{m}\right) = \left(\frac{n'}{m}\right)$.

(2) $\left(\frac{n^2}{m}\right) = 1$.

(3) $\left(\frac{n}{m}\right)\left(\frac{n'}{m}\right) = \left(\frac{nn'}{m}\right)$.

(4) $\left(\frac{n}{m}\right)\left(\frac{n}{m'}\right) = \left(\frac{n}{mm'}\right)$.

(5) $\left(\frac{-1}{m}\right) = (-1)^{\frac{m-1}{2}}$.

(6) $\left(\frac{2}{m}\right) = (-1)^{\frac{p^2-1}{8}}$.

(7) Lei da reciprocidade quadrática: $\left(\frac{n}{m}\right)\left(\frac{m}{n}\right) = (-1)^{\frac{n-1}{2}\cdot\frac{m-1}{2}}$.

Demonstração. As propriedades 1 a 3 são consequências imediatas da definição do símbolo de *Jacobi* e do facto destas propriedades se verificarem para o símbolo de *Legendre*.

A propriedade 4 segue da definição do símbolo de *Jacobi*. Se $m = p_1, \ldots, p_k$ e $m' = q_1, \ldots, q_l$ são as decomposições em factores primos de m e m', em os primos p_i e q_j podem aparecer repetidos, então

$$\left(\frac{n}{m}\right)\left(\frac{n}{m'}\right) = \left(\frac{n}{p_1}\right)\cdots\left(\frac{n}{p_k}\right)\left(\frac{n}{q_1}\right)\cdots\left(\frac{n}{q_l}\right) = \left(\frac{n}{p_1\cdots p_k q_1\cdots q_l}\right) = \left(\frac{n}{mm'}\right).$$

Para provar as propriedades 5, 6 e 7, necessitamos do seguinte Lema.

Lema 1.2.24 *Se x e y são números inteiros ímpares, então*

$$\frac{x-1}{2} + \frac{y-1}{2} \equiv \frac{xy-1}{2} \pmod 2, \quad e \tag{1.14}$$

$$\frac{x^2-1}{8} + \frac{y^2-1}{8} \equiv \frac{x^2y^2-1}{8} \pmod 2. \tag{1.15}$$

Demonstração do lema 1.2.24. Como x e y são números ímpares, os números $(x-1)$ e $(y-1)$ são pares. Logo $\frac{(x-1)(y-1)}{2}$ ainda é par e

$$0 \equiv \frac{(x-1)(y-1)}{2} \equiv \frac{xy-x-y+1}{2} \equiv \frac{xy-1-x+1-y+1}{2} \pmod 2.$$

Ficou provado que $\frac{x-1}{2} + \frac{y-1}{2} \equiv \frac{xy-1}{2} \pmod 2$.

Por outro lado, como $(x-1)$ e $(x+1)$ são pares, vem que $x^2-1 = (x-1)(x+1)$ tem pelo menos dois factores 2. Analogamente para y^2-1. Logo, $(x^2-1)(y^2-1)$ tem pelo menos 4 factores 2. Consequentemente, $\frac{(x^2-1)(y^2-1)}{8}$ é par e

$$0 \equiv \frac{(x^2-1)(y^2-1)}{8} \equiv \frac{x^2y^2-x^2-y^2+1}{8} \equiv \frac{x^2y^2-1-x^2+1-y^2+1}{8} \pmod 2.$$

Isto prova que $\frac{x^2-1}{8} + \frac{y^2-1}{8} \equiv \frac{x^2y^2-1}{8} \pmod 2$. □

Suponhamos que $m = p_1, \ldots, p_k$ e $n = q_1, \ldots, q_l$ são as decomposições em factores primos de m e n, em que os primos p_i e q_j podem aparecer repetidos. Logo,

$$\left(\frac{-1}{m}\right) = \left(\frac{-1}{p_1}\right) \cdots \left(\frac{-1}{p_k}\right) = (-1)^{\sum_{i=1}^{k} \frac{p_i-1}{2}} = (-1)^{\frac{m-1}{2}},$$

porque, pela congruência 1.14 do Lema 1.2.24 temos $\sum_{i=1}^{k} \frac{p_i-1}{2} \equiv \frac{(\prod_{i=1}^{k} p_i)-1}{2} \pmod 2$.

Analogamente se prova que $\left(\frac{2}{m}\right) = (-1)^{\frac{m^2-1}{8}}$, usando a congruência 1.15 do Lema 1.2.24.

Por fim, basta observar que, para i fixo, a congruência 1.14 do Lema 1.2.24 implica que $\sum_{j=1}^{l} \frac{p_i-1}{2}\frac{q_j-1}{2} \equiv \frac{p_i-1}{2} \sum_{j=1}^{l} \frac{q_j-1}{2} \equiv \frac{p_i-1}{2}\frac{(\prod_{j=1}^{l} q_j)-1}{2} \pmod 2$. Logo, pelas propriedades 3) e 4) temos

$$\begin{aligned}\left(\frac{n}{m}\right)\left(\frac{m}{n}\right) &= \left(\prod_{i=1}^{k}\prod_{j=1}^{l}\left(\frac{q_j}{p_i}\right)\right)\left(\prod_{i=1}^{k}\prod_{j=1}^{l}\left(\frac{p_i}{q_j}\right)\right) = \prod_{i=1}^{k}\prod_{j=1}^{l}\left(\frac{q_j}{p_i}\right)\left(\frac{p_i}{q_j}\right) \\ &= \prod_{i=1}^{k}\prod_{j=1}^{l}(-1)^{\frac{p_i-1}{2}\frac{q_j-1}{2}} = (-1)^{\sum_{i=1}^{k}\sum_{j=1}^{l}\frac{p_i-1}{2}\frac{q_j-1}{2}} \\ &= (-1)^{\sum_{i=1}^{k}\frac{p_i-1}{2}\frac{n-1}{2}} = (-1)^{\frac{m-1}{2}\frac{n-1}{2}}.\end{aligned}$$

Isto completa a prova da propriedade 7 e do Teorema. □

Com estas propriedades podemos "retirar" os factores 2 e seguidamente aplicar a Lei da reciprocidade quadrática sem a preocupação de verificar se o denominador é primo. Se $a = 2^e a_1$, em que a_1 é ímpar, então

$$\left(\frac{a}{n}\right) = \left(\frac{2^e}{n}\right)\left(\frac{a_1}{n}\right) = \left(\frac{2}{n}\right)^e \left(\frac{n \pmod{a_1}}{a_1}\right)(-1)^{\frac{a_1-1}{2}\frac{n-1}{2}}.$$

Isto permite construir um algoritmo rápido para calcular o símbolo de *Jacobi*, por exemplo, a relação 3.2 da secção 3.2.3.

Exemplo 1.2.7 Vamos calcular $\left(\frac{1003}{1151}\right)$.

$$\begin{aligned}\left(\frac{1003}{1151}\right) &= -\left(\frac{1151}{1003}\right) = -\left(\frac{148}{1003}\right) = -\left(\frac{2^2 \cdot 37}{1003}\right) = -\left(\frac{2^2}{1003}\right)\left(\frac{37}{1003}\right) \\ &= -\left(\frac{1003}{37}\right) = -\left(\frac{2^2}{37}\right) = -1.\end{aligned}$$

1.2.1.5 Solução geral da congruência $ax \equiv b \pmod{m}$

Dados os inteiros $a \not\equiv 0 \pmod{m}$ e b, pretendemos encontrar os inteiros x tais que $ax \equiv b \pmod{m}$ ou, de forma equivalente, encontrar os inteiros x tais que a equação Diofantina

$$ax - my = b \tag{1.16}$$

tenha soluções inteiras.

Podemos ver facilmente que a equação 1.16 tem soluções inteiras em x e y se, e só se, $(a, m) \mid b$.

Suponhamos que existem inteiros x e y que verificam $ax - my = b$. Considere $d = (a, m)$, ou seja, $d \mid a$ e $d \mid m$. Isto implica que existem inteiros k_1 e k_2 tal que $a = dk_1$ e $m = dk_2$. Logo $b = (ax - my) = dk_1x - dk_2y = d(k_1x - k_2y)$. Concluímos que b é divisível por (a, m).

Reciprocamente, suponha-se que $d \mid b$. O algoritmo extendido de *Euclides* 1.1.2.2 permite escrever d como combinação linear de a e m, ou seja

$$d = au + mv, \quad u \text{ e } v \text{ inteiros.}$$

Como $d \mid b$, podemos multiplicar cada membro da equação anterior por $\frac{b}{d}$ obtendo

$$b = a\frac{ub}{d} + m\frac{vb}{d}.$$

As soluções da equação 1.16 são $x = \frac{ub}{d}$ e $y = -\frac{vb}{d}$.

Ficou provado que a equação 1.16 tem soluções inteiras em x e y se, e só se, $(a, m) \mid b$.

Seja $(a, m) = d$ e suponhamos que $d \mid b$. Se (x_0, y_0) são soluções da equação 1.16, então a solução geral é

$$\begin{cases} x = x_0 + \frac{m}{d}t \\ y = y_0 + \frac{a}{d}t \end{cases}, \quad t \in \mathbb{Z}. \tag{1.17}$$

Como nos interessa somente resolver a congruência $ax \equiv b \pmod{n}$, só nos interessam as soluções $x = x_0 + \frac{m}{d}t$, com $t \in \mathbb{Z}$, que sejam distintas módulo m.

Teorema 1.2.25 *Seja $a \not\equiv 0 \pmod{m}$ e x_0 uma solução da congruência*

$$ax \equiv b \pmod{m}.$$

Seja $d = (a, m)$ e suponha-se que $d \mid b$, então a solução geral da congruência é dada por

$$x \equiv x_0 + t\frac{m}{d} \pmod{m},$$

onde $t = 0, \ldots, d-1$.

1.2.1.6 Teorema Chinês dos Restos

Consideremos a congruência

$$ax \equiv b \pmod{m} \tag{1.18}$$

e m um número inteiro grande. O cálculo de uma solução pode ser muito difícil dependendo do "tamanho" de m. Podemos simplificar o cálculo de soluções da congruência 1.18 do seguinte modo.

Suponhamos que m se factoriza como $m = \prod_{i=1}^{n} p_i^{\alpha_i}$, em que cada p_i é primo e $m_i = p_i^{\alpha_i}$, então $m \mid (ax - b)$ se e só se $m_i \mid (ax - b)$, para cada i. A congruência 1.18 pode ser escrita de forma equivalente como um sistema de congruências:

$$ax \equiv b \pmod{m_i}, \quad i = 1, \ldots, n. \tag{1.19}$$

Suponhamos que obtemos uma solução de cada uma das congruências do sistema 1.19. Por exemplo:

$$x \equiv a_i \pmod{m_i}, \quad i = 1, \cdots, n. \tag{1.20}$$

Isto significa que se tem 1.18 se, e só se, são satisfeitas simultaneamente as congruências 1.20, ou seja, encontrar soluções de 1.18 equivale a encontrar soluções do sistema 1.20. O próximo resultado, chamado Teorema Chinês dos Restos[5], dá-nos uma possível construção da solução.

Teorema 1.2.26 (Teorema Chinês dos Restos) *Considere inteiros $m_1, m_2, \ldots, m_n$ positivos e primos entre si dois a dois. O sistema*

$$x \equiv a_i \pmod{m_i}, \quad \textit{para } i = 1, \cdots, n$$

[5]Os primeiros registos deste Teorema aparecem em trabalhos de matemáticos Chineses, daí o nome *Teorema Chinês dos Restos.* O mais antigo de tais registos que contém o referido Teorema é o *"Sun Tzu Suan Ching"* (conhecido também como *"Sunzi Suanjing"*) escrito aproximadamente no terceiro século por *Sun Zi.*

tem exactamente uma solução módulo $m = m_1 \cdot m_2 \cdots m_n$. *Essa solução pode ser dada pelo algoritmo de Gauss,*

$$\alpha = \sum_{i=1}^{n} \frac{m}{m_i} \cdot b_i \cdot a_i$$

em que b_i *é tal que* $\frac{m}{m_i} \cdot b_i \equiv 1 \pmod{m_i}$.

Demonstração. Vejamos primeiro que existe uma solução. Temos que $\left(\frac{m}{m_i}, m_i\right) = 1$, para todo i. Daqui resulta que existe b_i tal que $\frac{m}{m_i} \cdot b_i \equiv 1 \pmod{m_i}$. Como $m_j \mid \frac{m}{m_i}$, quando $j \neq i$; então $\frac{m}{m_i} \cdot b_i \equiv 0 \pmod{m_j}$, para $j \neq i$. Assim o número inteiro:

$$\alpha = \sum_{i=1}^{n} \frac{m}{m_i} \cdot b_i \cdot a_i$$

é solução de cada uma das congruências consideradas, uma vez que:

$$\begin{aligned} \alpha &= \tfrac{m}{m_1} \cdot b_1 \cdot a_1 + \cdots + \tfrac{m}{m_n} \cdot b_n \cdot a_n \\ &\equiv \tfrac{m}{m_i} \cdot b_i \cdot a_i \pmod{m_i}, \quad \left(\tfrac{m}{m_j} \equiv 0 \pmod{m_i},\ i \neq j\right) \\ &\equiv a_i \pmod{m_i}, \quad \left(\tfrac{m}{m_i} \cdot b_i \equiv 1 \pmod{m_i}\right). \end{aligned}$$

Vejamos agora a unicidade da solução módulo m. Suponhamos que α e β são soluções de todas as congruências $x \equiv a_i \pmod{m_i}$, para todo i. Temos então $\alpha \equiv \beta \equiv a_i \pmod{m_i}$, para todo i. Logo $\alpha - \beta \equiv 0 \pmod{m_i}$ ou de forma equivalente $m_i \mid (\alpha - \beta)$, para todo i. Como os m_i são primos entre si dois a dois, temos que $m = m_1 \cdots m_n \mid (a - b)$, ou seja $a \equiv b \pmod{m}$. □

A demonstração faculta um método expedito para construir a solução.

Exemplo 1.2.8 Vamos ver um problema enunciado por *Sun Zi.*

> "Temos um número de objectos, mas não sabemos exactamente quantos são. Se os contarmos três a três sobram dois. Se os contarmos cinco a cinco sobram três. Se os contarmos sete a sete sobram dois. Quantos objectos serão?" *(Sun Tze Suan Ching)*

Na linguagem das congruências temos

$$x \equiv 2 \pmod{3}, \quad x \equiv 3 \pmod{5} \quad \text{e} \quad x \equiv 2 \pmod{7}.$$

A solução destas congruências é dada pelo algoritmo de *Gauss*:

$$\begin{aligned} \alpha &\equiv \textstyle\sum_{i=1}^{n} \frac{m}{m_i} \cdot b_i \cdot a_i \equiv \frac{3\cdot5\cdot7}{3} \cdot 2 \cdot 2 + \frac{3\cdot5\cdot7}{5} \cdot 1 \cdot 3 + \frac{3\cdot5\cdot7}{7} \cdot 1 \cdot 2 \\ &\equiv 70 \cdot 2 + 21 \cdot 3 + 15 \cdot 2 \equiv 23 \pmod{3 \cdot 5 \cdot 7} \end{aligned}$$

em que $b_i \cdot \frac{m}{m_i} \equiv 1 \pmod{m_i}$.

Exemplo 1.2.9 Ambas as congruências

$$x \equiv 0 \pmod 4 \quad \text{e} \quad x \equiv 1 \pmod 6$$

têm solução, mas como $(4,6) = 2 \neq 1$, o sistema não tem solução.

1.2.2 Prova do Teorema RSA 1.2.1

Estamos prontos para usar os resultados apresentados de Teoria de Números para provar o Teorema 1.2.1.

Demonstração. Pretende-se provar que:

$$M \equiv m^d \pmod n$$

i) Como $m \equiv M^c \pmod n$, temos que

$$m^d \equiv \underbrace{m \cdot m \cdots m}_{d\times} \equiv \underbrace{M^c \cdot M^c \cdots M^c}_{d\times} \equiv M^{cd} \pmod n,$$

assim fica provado que $m^d \equiv M^{cd} \pmod n$, bastando agora provar que $M^{cd} \equiv M \pmod n$.

ii) Uma vez que $cd \equiv 1 \pmod{\phi(n)}$, então existe $k \in \mathbb{Z}$ tal que

$$cd = 1 + k\phi(n) = 1 + k(q-1)(p-1),$$

pois $\phi(n) = \phi(pq) = (p-1)(q-1)$.

Pretende-se aplicar o Teorema de Fermat, para isso, M e p têm que ser primos entre si, conduzindo assim à análise de dois casos distintos tendo em conta o valor de (M,p).

Caso 1: Se $(M,p) = 1$, então pelo Teorema de Fermat,

$$M^{p-1} \equiv 1 \pmod p$$

logo

$$\begin{aligned} M^{cd} &\equiv M^{1+k\varphi(n)} \pmod p, \quad k \in \mathbb{Z} \\ &\equiv M^{1+k(q-1)(p-1)} \pmod p \\ &\equiv M \times (M^{p-1})^{k(q-1)} \pmod p \\ &\equiv M \times 1^{k(q-1)} \pmod p \\ &\equiv M \pmod p \end{aligned}$$

Caso 2: Se $(M, p) \neq 1$, então p divide M (p é primo) e assim

$$M \equiv 0 \pmod{p},$$

como p divide M, p divide M^{cd}, obtendo

$$M^{cd} \equiv 0 \pmod{p}$$

das duas congruência obtemos,

$$M^{cd} \equiv M \pmod{p}.$$

Logo em todos os casos $M^{cd} \equiv M \pmod{p}$.

iii) Usando o mesmo argumento de *ii)* obtemos:

$$M^{cd} \equiv M \pmod{q}$$

iv) Por fim, como $M^{cd} \equiv M \pmod{p}$ e $M^{cd} \equiv M \pmod{q}$, ou de forma equivalente,

$$p \mid \left(M^{cd} - M\right) \quad \text{e} \quad q \mid \left(M^{cd} - M\right)$$

os primos distintos p e q dividem o mesmo número $\left(M^{cd} - M\right)$, então o produto pq também é um divisor, ou seja,

$$n = pq \mid \left(M^{cd} - M\right) \quad (\Leftrightarrow M^{cd} \equiv M \pmod{n}).$$

Fica assim provado que $m^d \equiv M^{cd} \equiv M \pmod{n}$. □

Pensa-se que este algoritmo é uma "*one way function trapdoor*". De facto, inverter a equação $m \equiv M^c \pmod{n}$ é muito difícil, mas se conheceremos o expoente de decifragem d, o "*trapdoor*", é fácil inverter a função.

Vamos ver um exemplo de aplicação do sistema de cifragem RSA:

Exemplo 1.2.10 Escolhemos dois números inteiros primos $p = 983$ e $q = 1021$. Calculamos

$$n = p \cdot q = 983 \times 1021 = 1003643$$

e $\varphi(n) = \varphi(p \cdot q) = 982 \times 1020 = 1001640$. O primeiro c que verifica $(c, \varphi(n)) = 1$ é $c = 7$. Os números p, q e $\varphi(n)$ serão necessários mais tarde, mas, até então, devem ser mantidos secretos. Podemos divulgar a nossa *chave pública* $(1003643, 7)$, que consiste no *módulo* $n = 1003643$ e no *expoente de cifragem* $c = 7$.

Suponhamos que alguém pretende cifrar a mensagem:

"Segredo meio encoberto, é sempre descoberto!"

O computador converte todas as letras em números. Recorrendo ao padrão ASCII, a mensagem é dada na base decimal pela seguinte sequência de números inteiros:

M = 083 101, 103 114, 101 100, 111 032, 109 101, 105 111, 032 101, 110 099, 111 098, 101 114, 116 111, 044 032, 233 032, 115 101, 109 112, 114 101, 010 100, 101 115, 099 111, 098 101, 114 116, 111 033.

Visto que $n = 1003643$ tem 7 dígitos decimais, vamos cifrar a mensagem em 22 blocos m_i de 6 dígitos decimais cada (correspondentes a blocos de 2 letras), para garantir que cada bloco $M_i < n$, com $i = 1, \ldots, 22$. Para cifrar o bloco $M_1 = 83101$, calculamos:

$$m_1 \equiv 83101^7 \equiv 528189 \pmod{1003643}.$$

Repetindo o cálculo anterior para os 21 blocos restantes, obtemos a mensagem cifrada:

m = 528189, 266885, 490718, 283421, 353405, 306043, 744010, 322645, 364059, 297805, 544459, 286978, 565454, 387839, 370590, 919704, 324124, 535456, 460952, 140150, 267578, 8952.

Como a chave pública é conhecida por todos, qualquer pessoa nos pode cifrar e enviar mensagens com privacidade, pois é muito difícil inverter a *função de cifragem* sem conhecer *expoente de decifragem*!

Mas o legítimo proprietário do par de chaves RSA, deve conseguir inverter facilmente a *função de cifragem*, pois ele conhece os parâmetros p e q que nos permitem calcular o *expoente de decifragem* d.

Devemos encontrar o único inteiro d tal que $c \cdot d \equiv 1 \pmod{\varphi(n)}$ e $(d, \varphi(n)) = 1$, ou seja, queremos calcular d tal que $7 \cdot d \equiv 1 \pmod{1001640}$. Usando o *algoritmo extendido de Euclides*, obtemos $d = 286183$. No exemplo 1.1.2, o inverso $c \pmod{\varphi(n)}$ seria o coeficiente y_k. A *chave privada* consiste no *módulo* $n = 1003643$ e no *expoente de decifragem* $d = 286183$. Para decifrar o bloco m_1, como vimos na secção 1.1.2.3, podemos calcular rapidamente:

$$M_1 \equiv 528189^{286183} \equiv 83101 \pmod{1003643}.$$

Podemos recuperar a mensagem inteira aplicando o cálculo anterior aos restantes blocos!

Notamos que o que torna o sistema de cifragem RSA possível de implementar nos computadores actuais é o facto de o algoritmo extendido de *Euclides* ter um custo muito baixo, bem como a existência de um método rápido de exponenciação modular. Se não existissem estes algoritmos seria impraticável utilizar o sistema de cifragem RSA.

Em contraste, "ainda" não existe algum algoritmo rápido para factorizar números inteiros. Isto assegura a segurança do sistema de cifragem RSA. (ver 1.2.4)

1.2.3 Método RSA acelerado

1.2.3.1 Cálculo do expoente de decifragem d

Quando geramos um novo par de chaves, o cálculo do *expoente de decifragem* d pode optimizado recorrendo ao Teorema 1.2.27. Para isso, vamos primeiro definir o *expoente universal.*

Definição 1.2.7 *Seja n o produto de dois números primos p e q. Definimos o número $\lambda(n)$, o "mínimo múltiplo comum" $[p-1, q-1]$, como o* ***expoente universal*** *de n. Para diminuir o custo do cálculo de $\lambda(n)$, podemos calcular:*

$$\lambda(n) = \frac{\phi(n)}{(p-1, q-1)}$$

uma vez que:

$$(p-1, q-1) \times [p-1, q-1] = (p-1)(q-1) = \phi(n).$$

O *expoente universal* $\lambda(n)$ pode ser usado no lugar de $\phi(n)$ para calcular o expoente de decifragem d da chave privada (n, d). Podemos calcular, usando o algoritmo extendido de Euclides 1.1.2.2, o único inteiro d tal que,

$$c \times d \equiv 1 \pmod{\lambda(n)}$$

isto é, d é o inverso de c módulo $\lambda(n)$. Desta forma, reformulamos o passo 4 da Geração das chaves *pública* e *privada*, de modo a optimizar a obtenção do *expoente de decifração.*

Teorema 1.2.27 *Seja (n, c) a chave pública de cifragem e M um inteiro menor que n.*

$$\text{Se} \quad \begin{cases} m \equiv M^c \pmod{n} \\ c \times d \equiv 1 \pmod{\lambda(n)} \end{cases}, \quad \text{então} \quad M \equiv m^d \pmod{n}$$

Demonstração. Pretende-se provar que:

$$M \equiv m^d \pmod{n}$$

onde agora d é tal que $c \times d \equiv 1 \pmod{\lambda(n)}$. Esta prova é de todo idêntica à do Teorema 1.2.1, diferindo apenas na prova do segundo caso:

$$M^{cd} \equiv M \pmod{p}, \quad \text{onde} \quad (M,p) = 1.$$

i) Uma vez que $cd \equiv 1 \pmod{\lambda(n)}$, então existe $k \in \mathbb{Z}$ tal que:

$$cd = 1 + k\lambda(n) = 1 + k\frac{\phi(n)}{(p-1,q-1)} = 1 + k\frac{(q-1)(p-1)}{(p-1,q-1)}, \tag{1.21}$$

como $(p-1,q-1) \mid (q-1)$, fazendo $h = \frac{(q-1)}{(p-1,q-1)}$ a equação 1.21 é equivalente a $h(p-1)$.

Pretende-se agora aplicar de novo o Teorema de Fermat. Para isso, vamos considerar que M e p são primos entre si. O outro caso, $(M,p) = p$ é óbvio. Se $(M,p) = 1$,

$$\begin{aligned}
M^{cd} &\equiv M^{1+k\lambda(n)} \pmod{p}, \quad k \in \mathbb{Z} \\
&\equiv M^{1+k\frac{\phi(n)}{(p-1,q-1)}} \pmod{p} \\
&\equiv M^{1+k\frac{(q-1)(p-1)}{(p-1,q-1)}} \pmod{p} \\
&\equiv M \times M^{kh(p-1)} \pmod{p}, \quad l \in \mathbb{Z} \\
&\equiv M \times \left(M^{p-1}\right)^{kh} \pmod{p} \\
&\equiv M \times 1^{kh} \pmod{p} \\
&\equiv M \pmod{p}
\end{aligned}$$

Logo $M^{cd} \equiv M \pmod{p}$.

ii) Usando o mesmo argumento obtemos $M^{cd} \equiv M \pmod{q}$

iii) Como $M^{cd} \equiv M \pmod{p}$ e $M^{cd} \equiv M \pmod{q}$, os primos distintos p e q dividem o mesmo número $\left(M^{cd} - M\right)$, logo o produto pq também é um divisor de $\left(M^{cd} - M\right)$, ou seja, provou-se que

$$m^d \equiv M^{cd} \equiv M \pmod{n}.$$

□

1.2.3.2 Optimização da decifração

Sejam (n,c) e (n,d) as chaves pública e privada de cifragem, respectivamente, e m uma mensagem cifrada com a chave pública (n,c). Pelo Teorema 1.2.1, para decifrar a mensagem cifrada m, aplicamos a chave privada (n,d) para calcular $M \equiv m^d \pmod{n}$.

Na prática, para cifrar e decifrar mensagens mais rapidamente, escolhemos c como o primeiro inteiro maior do que 1 que é primo com $\varphi(n)$. Com esta modificação, obtemos um expoente de cifragem c pequeno e, consequentemente, uma codificação de mensagens mais rápida. Para a descodificação de mensagens, o *GnuPG v1.06* usa o processo descrito pelo Teorema 1.2.28 para realizar operações modulares, com módulos mais pequenos que n.

Definição 1.2.8 *Sejam p e q dois primos distintos, de modo que $p < q$. Definimos $\mu(p,q) = p^{-1} \pmod{q}$, ou seja, $\mu = \mu(p,q)$ é um número inteiro para o qual existe um inteiro η tal que $\mu p + \eta q = 1$.*

Teorema 1.2.28 *Seja (n,p,q,d,μ) a chave privada de cifragem* ***estendida****, M um inteiro menor que n e seja $m \equiv M^c \pmod{n}$.*

$$\textit{Sejam} \begin{cases} m_1 \equiv m^{d \pmod{p-1}} \pmod{p} \\ m_2 \equiv m^{d \pmod{q-1}} \pmod{q} \\ h \equiv \mu(m_2 - m_1) \pmod{q} \end{cases}, \quad \begin{array}{l} \textit{então } M = m_1 + hp \textit{ é solução da equação} \\ M^c \equiv m \pmod{n} \end{array}$$

Demonstração. Como d é o inverso de c módulo $\phi(n)$, obtemos:

$$\begin{aligned} & cd \equiv 1 \pmod{\phi(n)} \\ \Leftrightarrow\ & cd + k'\phi(n) = 1, \quad \text{para algum } k' \in \mathbb{Z} \\ \Leftrightarrow\ & cd + [k'(q-1)](p-1) = 1 \\ \Rightarrow\ & cd \equiv 1 \pmod{p-1} \end{aligned}$$

Por definição de m_1,

$$\begin{aligned} m_1 &\equiv m^{d \pmod{p-1}} \pmod{p} \\ &\equiv m^{d+k(p-1)} \pmod{p}, \quad \text{para algum } k \in \mathbb{Z} \\ &\equiv (M^c)^{d+k(p-1)} \pmod{p} \\ &\equiv M^{cd+ck(p-1)} \pmod{p} \\ &\equiv M^{cd \pmod{p-1}} \pmod{p} \\ &\equiv M^{1+l(p-1)} \pmod{p}, \quad \text{para algum } l \in \mathbb{Z} \\ &\equiv M \times \left(M^{(p-1)}\right)^l \pmod{p} \\ &\equiv M \times 1^l \pmod{p} \\ &\equiv M \pmod{p} \end{aligned}$$

De forma análoga podemos ver que

$$m_2 \equiv M \pmod{q}.$$

Por um lado sabemos que:

$$\begin{cases} m_1 \equiv M \pmod{p} \\ m_2 \equiv M \pmod{q} \end{cases}$$

por outro, pelo Teorema Chinês dos Restos o sistema

$$\begin{cases} X \equiv M \pmod{p} \\ X \equiv M \pmod{q} \end{cases}$$

tem uma única solução $\pmod{pq}$

$$X \equiv M \pmod{pq}.$$

A solução dada pela demonstração do Teorema Chinês dos Restos é:

$$M = m_1 \eta q + m_2 \mu p,$$

onde η e μ são tais que $\mu p + \eta q = 1$. Substituindo $\eta q = 1 - \mu p$ em M obtemos:

$$M = m_1 + hp, \quad \text{onde } h = (m_2 - m_1)\mu.$$

□

1.2.4 Segurança no RSA

Não existe nenhuma técnica que permita provar que um criptossistema é seguro. Podemos unicamente procurar um meio para o quebrar.

Suponhamos que "perdemos" os números primos secretos p e q, mas que conhecemos o produto $n = pq$. Como recuperar os primos p e q? Será possível decifrar mensagens sem conhecer p e q?

Pretende-se mostrar que a segurança do sistema de cifragem RSA reside apenas na dificuldade do problema da factorização de números inteiros. Todos os caminhos possíveis para quebrar o RSA são, pelo menos, tão difíceis como factorizar n.

1.2.4.1 Calcular $\varphi(n)$ sem factorizar n

Se conseguirmos descobrir $\varphi(n)$, podemos facilmente obter d calculando o inverso de c módulo $\varphi(n)$. Este processo pode não ser mais fácil do que factorizar n, visto que $\varphi(n)$ permite factorizar n facilmente.

Note-se que conhecemos o produto $P = pq = n$ e que $\varphi(n) = (p-1)(q-1) = n - (p+q) + 1$. É fácil obter a soma $S = p + q$ calculando $S = p + q = n - \varphi(n) + 1$. Logo p e q são raízes do polinómio $x^2 - (n - \varphi(n) + 1)x + n = 0$. O cálculo das raízes do polinómio anterior é rápido ($O(\log^3(n))$).

Existe outra maneira, semelhante, de calcular p e q. Primeiro calculamos $p + q = n - \varphi(n) + 1$. Seguidamente calculamos $p - q = \sqrt{(p+q)^2 - 4n}$. Por fim $q = \frac{(p+q)-(p-q)}{2}$.

Este dois processos permitem obter facilmente uma factorização de n conhecendo $\varphi(n)$. Logo conhecer $\varphi(n)$ é equivalente a conhecer a factorização de n.

1.2.4.2 Calcular d sem conhecer $\varphi(n)$

Calcular d sem conhecer $\varphi(n)$ não deve ser mais fácil do que factorizar n, uma vez que o conhecimento de d permite factorizar n facilmente. Miller [26] provou que n pode ser factorizado usando um múltiplo de $\varphi(n)$. Uma vez que a congruência $cd \equiv 1 \pmod{\varphi(n)}$ é equivalente a $cd = 1 + k\varphi(n)$, para algum $k \in \mathbb{Z}$, conhecer d é equivalente a conhecer a factorização de n.

É claro que o expoente de decifragem d deve ser escolhido a partir de um conjunto suficientemente grande. Se n é grande, uma procura extensa de d não é uma tarefa mais fácil do que factorizar n.

Podemos tentar encontrar um inteiro d' que seja equivalente a d, mas, com vimos em 1.2.3.1, o expoente de decifragem é congruente com d módulo $\lambda(n) = \text{mmc}\,(p-1, q-1)$. Assim, encontrar algum $d' \pmod{\lambda(n)}$ não é mais fácil do que factorizar n.

A última possibilidade de quebrar o sistema, sem conhecer $\varphi(n)$, consiste em encontrar $M \pmod{n}$ a partir de $M^c \pmod{n}$, ou seja, calcular as c–ésimas raízes módulo n sem factorizar n. Este problema não é tão bem conhecido como a factorização de números inteiros, mas parece ser um problema muito difícil.

1.2.4.3 A factorização e o futuro

Rivest, *Shamir* e *Adleman* conjecturaram que a segurança do sistema de cifragem RSA reside na dificuldade em factorizar o módulo n, ver [36]. Qualquer método usado para tentar quebrar o RSA permite factorizar eficientemente o módulo n.

Se os primos p e q são muito grandes, digamos 200 dígitos decimais cada, a factorização de n pode demorar uma eternidade como veremos no capítulo 3. Mas os rápidos avanços da tecnologia fazem surgir computadores com capacidades de processamento em paralelo e com processadores cada vez mais velozes, bem como novos algoritmos sofisticados para factorização de números inteiros, como o *"Quadratic Sieve"* e o *"Number Field Sieve"*, ver 3.3.

Será que, de futuro, devido à constante evolução dos computadores, as comunicações seguras serão impossíveis? O tempo necessário para multiplicar dois números, cada vez maiores, cresce mais devagar (polinomialmente) do que o da sua factorização (exponencial). O perigo em si não parece vir dos computadores, mas da Matemática! Apesar de estes algoritmos de factorização não serem polinomiais (pensa-se não haver nenhum polinomial) o tempo necessário para factorizar números grandes tem diminuído drasticamente, devido a novos métodos matemáticos.

Apesar da extrema dificuldade do problema, sucessivos desafios de factorização de números inteiros "grandes" são vencidos muito mais rapidamente do que se poderia prever inicialmente.

Em 1976, estimou-se que o número inteiro RSA-129 (412 bits), número com 129 dígitos decimais usado pelos autores do RSA para cifrar a primeira mensagem com método de chave pública, demoraria milhões de anos a factorizar. Mas tal não se verificou! Em 1994, após um esforço de 8 meses coordenados por Leylad, Atkins e Graff, uma rede de computadores do mundo inteiro, cerca de 1600 máquinas, factorizou o número RSA-129 em dois inteiros primos com 64 e 65 dígitos decimais, usando o algoritmo de factorização *"Quadratic Sieve"* de *Carl Pomerance*. A matriz final quadrada tinha dimensão $188\,346^2$.

O texto da mensagem cifrada era "*The magic words are squeamish ossifrage*[6]".

Em 22 Agosto de 1999, um esforço conjunto de vários investigadores liderado por *Riele*, permitiu factorizar um número inteiro com 155 dígitos decimais (512 bits) em dois

[6] *Ossifrage* é uma espécie de abutre.

inteiros primos de 78 dígitos decimais. O algoritmo de factorização usado foi o "*Number Field Sieve*", numa vasta rede de computadores em computação paralela. Seriam precisos cerca de 8 000 Anos-MIPS, ou seja, cerca de 2^{58} operações de CPU! A filtragem de dados demorou 1 mês inteiro e a matriz final tridimensional tem 6 699 191 linhas, 6 711 336 colunas e 417 131 631 entradas em altura.

Actualmente, um módulo com menos de 1024 bits, cerca de 300 dígitos decimais, deve ser considerado inseguro. À medida que a tecnologia avança, aumenta o tamanho do módulo n e o RSA torna-se cada vez mais fácil de utilizar e cada vez mais seguro!

No entanto, se alguém descobrir um método matemático de factorização de inteiros rápido, terá em sua posse um arma poderosa que poderá desencadear o caos no mundo inteiro. Uma evolução mais natural é o aparecimento de criptossistemas que generalizem o RSA utilizando grupos mais complexos: as curvas elípticas ou polinómios mais complexos, ver [8].

1.3 Sistema de cifragem Knapsack

Desde o nascimento da criptografia de chave pública apareceram e floresceram vários criptossistemas de múltiplas variáveis de natureza combinatória baseados no problema da soma de subconjuntos ou *problema da mochila* (knapsack). Infelizmente todos eles quebrados com sucesso!

O sistema de cifragem *Chor-Rivest* foi publicado em 1984 e revisto em 1988. É o único sistema baseado na soma de subconjuntos sem recorrer à "usual" multiplicação modular. *Schnorr* e *Horner* desenvolveram um ataque que reduz para algumas horas o tempo necessário para quebrar o esquema.

O criptossistema de chave pública que vamos ver em detalhe é o de *Merkle-Hellman* (§1.3.2), proposto em 1978 e conhecido como o *criptossistema knapsack*. Este criptossistema foi "quebrado" por *Adi Shamir* na conferência Crypto'82, veja 1.3.4 e [40]. Seguidamente, *Merkle* publicou uma versão do sistema de cifragem de *Merkle-Hellman* com várias multiplicações modulares que foi também quebrado, por *Brickell*.

1.3.1 Problema da soma de subconjuntos

O problema *geral* da *soma de subconjuntos* consiste em determinar, dados vários inteiros (ou pesos) $c_1, c_2, \ldots, c_n$ e m, se existe um subconjunto $c_{i_1}, \ldots, c_{i_k}$ cuja soma é m. Isto é equivalente a determinar se existem variáveis $x_1, x_2, \ldots, x_n$ tais que

$$\sum_{j=1}^{n} x_j c_j = m, \quad x_j \in \{0,1\} \text{ para todo o } j. \tag{1.22}$$

Trata-se de um problema de *decisão*, pois só perguntamos se é possível encontrar uma solução. Note-se que este problema pode ter várias soluções, ter uma única solução ou não ter nenhuma solução.

O problema da *factorização de inteiros* é aparentemente difícil, mas ainda não foi provado que é difícil. Em contraste, como vimos no exemplo 1.1.5, o *problema da soma de subconjuntos* é um problema $\mathcal{NP}$–completo, ou seja, se encontrarmos algum algoritmo determinístico polinomial que resolva o problema da *soma de subconjuntos*, então a conjectura $\mathcal{NP} \neq \mathcal{P}$ será falsa.

Não é conhecido nenhum algoritmo polinomial que resolva o problema geral da soma de subconjuntos. Podemos determinar se a equação 1.22 tem solução e até determinar qual a solução, calculando todas as possíveis somas $\sum_{j=1}^{n} x_j c_j$ em que $x_j \in \{0,1\}$. Sabemos que existem 2^n subconjuntos de $\{c_i\}_{i=1,\ldots,n}$, logo seriam necessários 2^n passos!

Outro método consiste em calcular

$$S_1 = \left\{ \sum_{j=1}^{\lfloor n/2 \rfloor} x_j c_j : x_j = 0 \text{ ou } 1 \right\} \text{ e } \quad S_2 = \left\{ s - \sum_{j>\lfloor n/2 \rfloor} x_j c_j : x_j = 0 \text{ ou } 1 \right\},$$

(com custo de $2^{\frac{n}{2}}$ passos) e procurar um elemento comum aos conjuntos S_1 e S_2. A existência de um elemento comum para os conjuntos S_1 e S_2 acontece precisamente quando existe uma solução para a equação 1.22: se

$$y = \sum_{j=1}^{\lfloor n/2 \rfloor} x_j c_j = s - \sum_{j>\lfloor n/2 \rfloor} x_j c_j, \quad \text{então} \quad s = \sum_{j=1}^{n} x_j c_j.$$

Para o processo completo são necessários pelo menos $2^{\frac{n}{2}}$ passos, bem como espaço para guardar $2^{\frac{n}{2}}$ elementos, o que por vezes pode ser impeditivo. Contudo, este algoritmo continua a ser o mais rápido para resolver o problema geral da *soma de subconjuntos*.

1.3.2 Sistema de cifragem de Merkle-Hellman

Consideremos uma mensagem M representada na forma binária por $M = x_1x_2 \ldots x_n$, em que $x_j \in \{0,1\}$ e $1 \leq j \leq n$. A ideia consiste em usar um conjunto público de *"pesos"* $C_P = (c_1, c_2, \ldots, c_n)$ gerados pelo Pedro. Com este conjunto podemos cifrar a mensagem M calculando

$$m = \sum_{j_1}^{n} x_j c_j.$$

A mensagem cifrada m é a mensagem a transmitir ao Pedro.

Se a mensagem m for intersectada pelo João, para recuperar a mensagem original M, este terá que resolver o problema geral da soma de subconjuntos, um problema aparentemente intratável!

Porém, se o João tem de resolver o problema da soma de subconjuntos, também o Pedro o terá de resolver ao receber a mensagem m! Isto torna o esquema impraticável a menos que o Pedro possua um poder de cálculo muito superior ao normal, o que não é aceitável pois este esquema deveria permitir ao Pedro, o legítimo destinatário, decifrar facilmente a mensagem m. Existem alguns tipos de problemas de soma de subconjuntos que são fáceis de resolver e que vão servir de "*alçapão*" para o nosso esquema.

Conjuntos *supercrescentes*: Se $c_j = 2^{j-1}$ em que $1 \leq j \leq n$, então

$$m = \sum_{j=1}^{n} x_j 2^{j-1},$$

em que os x_j representam os dígitos binários de m. De uma forma mais geral, um conjunto $(c_i)_{i=1,\ldots,n}$ diz-se *supercrescente* se

$$c_j > \sum_{i=1}^{j-1} c_i, \quad 2 \leq j \leq n. \tag{1.23}$$

Exemplo 1.3.1 O conjunto $(2, 3, 7, 15, 31, 63, 127, 235)$ é um conjunto *supercrescente*.

Neste caso, o problema da soma de subconjuntos é fácil de resolver. Observe que $x_n = 1$ se, e só se

$$m > \sum_{j=1}^{n-1} c_j,$$

e que $x_n = 0$ caso contrário. Uma vez determinado $x_n \in \{0,1\}$, o problema fica reduzido a determinar $x_1, x_2, \ldots, x_{n-1}$ tal que

$$m - x_n c_n = \sum_{j=1}^{n-1} x_j c_j, \quad \text{em que } x_j \in \{0,1\}, \text{ e } 1 \leq j \leq n-1.$$

Isto permite calcular recursivamente $M = x_1 x_2 \ldots x_n$. O uso de um *n–uplo ordenado supercrescente* $(c_1, c_2, \ldots, c_n)$ permitiria ao Pedro recuperar a mensagem original a partir de m, mas também o possibilitaria igualmente a terceiros.

O sistema de cifragem *Merkle-Hellman* combina estes dois tipos de conjuntos, *geral* e *supercrescente*, para obter um criptossistema de chave pública. Começamos por gerar um conjunto supercrescente $D = (d_1, d_2, \ldots, d_n)$ que é fácil de resolver. Dissimulamos o conjunto D num conjunto geral $C = (c_1, c_2, \ldots, c_n)$, preservando a estrutura de D, de modo que os "pesos" públicos $c_1, \ldots, c_n$ escondam essa estrutura. Contudo, devemos conseguir inverter o processo para resolver apenas o problema de subconjuntos supercrescentes. A transformação usada por *Merkle* para dissimular o conjunto D é uma multiplicação modular.

Geração das chaves *pública* e *privada*: Suponhamos que o Pedro pretende receber mensagens cifradas com o sistema de cifragem de *Merkle-Hellman*. Vamos primeiro gerar as chaves *pública* e *privada*.

- Começamos por gerar um *n–uplo ordenado supercrescente* $D_P = (d_1, d_2, \ldots, d_n)$, em que se verifica:

$$d_1 \approx 2^{n+1}; \quad d_j > \sum_{i=1}^{j-1} d_j, \text{ para } 2 \leq j \leq n; \quad d_n \approx 2^{2n}. \tag{1.24}$$

  (Discutiremos adiante a escolha do conjunto supercrescente)

- Escolhemos aleatoriamente dois inteiros positivos N e W tais que:

$$2 \cdot \sum_{j=1}^{n} d_j > N > \sum_{j=1}^{n} d_j, \quad \text{e } (N, W) = 1.$$

- Calculamos

$$c'_j \equiv d_j \cdot W \pmod{N}. \tag{1.25}$$

Visto que $(N, W) = 1$ e $N > d_j$, os valores c'_j nunca se anulam, para todo o j entre 1 e n. De facto, suponhamos que existe um j para o qual $c'_j = 0$. Então $N \mid d_j \cdot W$. Como $(N, W) = 1$, pela proposição 1.2.1 obtemos que $N \mid d_j$. Isto contradiz o facto de $N > d_j$. Podemos concluir que cada c'_j está estritamente entre 0 e N.

Talvez alguma informação seja facultada caso seja conhecido que o "peso" público c'_j foi calculado a partir do "peso" privado d_j. Isto pode ser remediado permutando os elementos do conjunto C.

- Escolhemos uma permutação π no conjunto de índices {1,...,n} e definimos

$$c_j = c'_{\pi(j)}, \quad 1 \leq j \leq n. \tag{1.26}$$

O conjunto $C_P = (c_1, c_2, \ldots, c_n)$ é a *chave pública* do Pedro, enquanto que a *chave privada* consiste do conjunto $D_P = (d_1, \ldots, d_n)$, dos parâmetros N e W e da permutação π. Os parâmetros (D_P, N, W, π) devem ser mantidos secretos (a chave privada).

Cifragem: Seja agora M a mensagem numérica, ou parte da mensagem numérica a ser cifrada para enviar ao Pedro.

(i) Consideremos $M = x_1 x_2 \ldots x_n$ representado na base binária cujo comprimento deve ser igual ao número de elementos do conjunto D_P.

(ii) Calculamos:

$$m = \sum_{j=1}^{n} x_j c_j.$$

Podemos enviar ao Pedro a mensagem cifrada m através de um canal público. Mesmo que a mensagem cifrada m seja interceptada, aparentemente não seria possível decifra-la sem conhecer (D_P, N, W, π), a chave *privada* do Pedro.

Decifragem: Para recuperar a mensagem original M a partir da mensagem cifrada m, o Pedro vai começar por calcular

$$y \equiv m \cdot W^{-1} \pmod{N},$$

onde W^{-1} denota o inverso de W módulo N.

Observe que pelas equações 1.25 e 1.26, temos que

$$y \equiv m \cdot W^{-1} \equiv \sum_{j=1}^{n} x_j \cdot c_j \cdot W^{-1} \equiv \sum_{j=1}^{n} x_j \cdot c'_{\pi(j)} \cdot W^{-1} \equiv \sum_{j=1}^{n} x_j \cdot d_{\pi(j)} \pmod{N}.$$

Uma vez que N foi escolhido satisfazendo a condição $N > \sum_{j=1}^{n} d_j$, temos forçosamente

$$y = \sum_{j=1}^{n} x_j \cdot d_{\pi(j)}. \tag{1.27}$$

Para finalizar o processo de decifragem:

- Resolvemos o problema da soma de subconjuntos 1.27 cuja solução $(x'_1, \ldots, x'_n)$ é fácil calcular visto que $D_P = (d_1, d_2, \ldots, d_n)$ é um conjunto supercrescente.

- Como na equação 1.27 o dígito x_j é multiplicado pelo "peso" $d_{\pi(j)}$, aplicamos a permutação π a $(x'_1, \ldots, x'_n)$ para restaurar a posição original dos dígitos binários de M.

O Pedro é a única pessoa que conhece a sua chave *privada* (D_P, N, W, π), portanto só ele consegue restaurar a mensagem M.

Exemplo 1.3.2 Vamos primeiramente gerar a chave privada e pública de cifragem. Escolhemos um conjunto *supercrescente* de 8 elementos que verifica as condições 1.24:

$$D = (2, 3, 7, 15, 31, 63, 127, 235).$$

A soma de todos os elementos de D é 483. Escolhemos $N = 500 > 483$ e o número inteiro $W = 23$ primo com N.

Calculamos $c'_1 = 46 \equiv 2 \cdot 23 \pmod{500}$. Efectuando o mesmo cálculo para os outros elementos do conjunto D obtemos

$$C' = (46, 69, 161, 213, 345, 405, 421, 449).$$

Construímos o conjunto C aplicando aos elementos de C' a permutação $\pi = (2\,1\,5\,4\,6\,8\,7\,3)$,

$$C = (69, 161, 421, 345, 46, 213, 449, 405).$$

A chave pública consiste do conjunto $C = (69, 161, 421, 345, 46, 213, 449, 405)$ e a chave privada consiste de $D = (2, 3, 7, 15, 31, 63, 127, 235)$, $N = 500$, $W = 23$, $\pi =$ (2 1 5 4 6 8 7 3).

Suponhamos que alguém pretende cifrar a mensagem: ''knapsack''.

O computador converte todas as letras em números. Recorrendo ao padrão ASCII, a mensagem é dada na base binária pela seguinte sequência de conjuntos de 8 bits:

$$M = (01101011, 01101110, 01100001, 01110000, 01110011, 01100001, 01100011, 01101011).$$

Note-se que o conjunto público C tem 8 elementos, tanto quanto os bits de uma letra. Para cifrar a primeira letra da mensagem, calculamos

$$\sum_{i=1}^{8} x_i c_i = 161 + 421 + 46 + 449 + 405 = 1482.$$

Repetindo o cálculo anterior para os 7 blocos restantes, obtemos a mensagem cifrada:

$$m = 1482, 1290, 987, 927, 1781, 987, 1436, 1482.$$

Quem interceptar esta mensagem terá de resolver o problema geral da soma de subconjunto para reaver a mensagem original. É claro que este caso é fácil pois teríamos de, apenas, testar 2^8 somas.

O inverso de W módulo N é $U = 87$ pois $87 \times 23 \equiv 1 \pmod{500}$. Para decifrar o bloco m_1, calculamos $1482 \cdot 87 \equiv 434 \pmod{500}$. Resolvemos o problema da soma de subconjuntos supercrescente, que é fácil de solucionar conhecendo o conjunto D, constituinte da chave privada. Um subconjunto de $D = (2, 3, 7, 15, 31, 63, 127, 235)$, com soma dos seus elementos 434, é $2 + 7 + 63 + 127 + 235 = 434$, ou seja, $M_1' = 10100111$. Finalmente, para restaurar a ordem dos dígitos aplicamos a permutação $\pi = (2\ 1\ 5\ 4\ 6\ 8\ 7\ 3)$ aos elementos de M_1' para obter: $M_1 = 01101011$.

Podemos recuperar a mensagem inteira aplicando o cálculo anterior aos restantes blocos.

1.3.3 Parâmetros e funcionamento do sistema

Todos os sistemas de cifragem baseados no problema da soma de subconjuntos ofereciam uma particularidade atractiva: serem extremamente rápidos. O sistema de cifragem RSA

é muito lento quando comparado com sistemas de cifragem simétricos como o DES (Data Encryption Standard). Utilizando módulo de 500 bits, o que já não é suficiente, podemos cifrar dados a uma taxa de 10^2 bits por segundo, o que é pouco uma vez que o sistema DES atinge a taxa 10^5 bits por segundo. O sistema de cifragem RSA chega a ser 1000 vezes mais lento do que sistemas de cifragem clássicos.

Uma vez que o melhor algoritmo conhecido para resolver o problema da soma de subconjuntos tem custo da ordem $2^{\frac{n}{2}}$, um parâmetro aceitável para N é $\simeq 100$. Para um módulo de 500 bits, o sistema de cifragem de *Merkle-Hellman* chega a ser 100 vezes mais rápido do que o sistema de cifragem RSA. Esta vantagem deve-se principalmente ao facto de ser necessário apenas uma multiplicação modular. No entanto, o tamanho das chaves é maior no sistema de cifragem de *Merkle-Hellman*, bem como o tamanho das mensagens cifradas.

Se alguns elementos d_i do conjunto D ou o módulo N forem "grandes", então o criptossistema poderá ser ineficiente, uma vez que n bits de informação são transformados "grosseiramente" em $\log_2 M$ bits. Por outro lado, os "pesos" d_i não devem ter valores pequenos. Se $d_1 = 1$ e se $c_j = W$ para algum j, podemos testar todos c_j como possíveis candidatos a W e pôr em causa a segurança do sistema.

Se escolhermos valores $d_j = k2^{j-1}$, $1 \leq j \leq n$, para algum $k \approx 2^n$, podemos obter um sistema de cifragem muito inseguro. Basta observarmos que, para j entre 1 e $n-1$, temos

$$c'_{j+1} = 2c'_j \quad \text{ou} \quad c'_{j+1} = 2c'_j - N.$$

É fácil deduzir o módulo N, pois N aparece várias vezes na matriz de dimensão $n \times n$, com valores

$$2c_i - c_j, \quad 1 \leq i, j \leq n.$$

Para esconder melhor a estrutura original do conjunto supercrescente D, *Merkle* propôs efectuar várias iterações da operação de multiplicação modular 1.25. Sejam $N_1 = N$ e $W_1 = W$, $c_j^{(0)} = d_j$ e $c_j^{(1)} = c'_j$. Construímos uma sequência de módulos e multiplicadores escolhendo iterativamente inteiros positivos N_k e W_k, primos entre si, verificando

$$N_k > \sum_{j=1}^{n} a_j^{(k-1)}.$$

Por fim, construímos o conjunto público $C = (c_1^{(k)}, \ldots, c_n^{(k)})$ calculando recursivamente

$$c_j^{(k)} \equiv c_j^{(k-1)} W_k \pmod{N_k}.$$

Neste caso, a chave pública consiste de um conjunto $C = (c_{\pi(1)}^{(k)}, \ldots, c_{\pi(n)}^{(k)})$ e a chave privada consiste de $((d_1, \ldots, d_n), N_1, W_1, \ldots, N_k, W_k, \pi)$.

Este processo parece dissimular melhor o conjunto D mas isto não é suficiente para se obter um criptossistema seguro, pois o sistema foi novamente quebrado por *Brickell.*

1.3.4 Segurança do criptossistema *Merkle-Hellman*

Durante certo tempo, pensou-se que este sistema de cifragem seria seguro, uma vez que, o problema da soma de subconjuntos pertence a uma classe de problemas muito difíceis (problemas $\mathcal{NP}$–completos). Todavia, existe uma falha neste raciocínio: o conjunto público C é obtido a partir de um conjunto supercrescente através uma transformação simples que consiste de uma só multiplicação modular.

Na conferência Crypto'82, *Adi Shamir* apresentou um algoritmo polinomial que permite "quebrar" o sistema de cifragem de *Merkle-Hellman.* Vamos apresentar um "esboço" da ideia de *Shamir* para mostrar em que consiste a falha de segurança do sistema de cifragem *knapsack.*

Seja $((d_1, \ldots, d_n), N, W, \pi)$ a chave privada e $(c_1, \ldots, c_n)$ a chave pública do sistema de cifragem *Merkle-Hellman.* Suponhamos que só conhecemos o conjunto público $(c_1, \ldots, c_n)$, este algoritmo permite encontrar um par (U', N'), a partir do qual se calcula um conjunto supercrescente

$$d'_j \equiv c_j \cdot U' \pmod{N'} \tag{1.28}$$

que satisfaz $\sum_{j=1}^{n} d'_j < N'$.

Uma vez encontrado um conjunto supercrescente D' podemos decifrar mensagens facilmente.

Dividindo a equação 1.28 por N' obtemos

$$\frac{d'_j}{N'} \equiv \left(c_j \cdot \frac{U'}{N'} \right) \pmod{1}, \quad \text{e} \tag{1.29}$$

$$\sum_{j=1}^{n} \frac{d'_j}{N'} \equiv \sum_{j=1}^{n} \left(c_j \cdot \frac{U'}{N'} \right) \pmod 1 < 1, \tag{1.30}$$

em que $a \pmod 1$ significa $a - [a]$.

Consideremos $r' = \frac{U'}{N'}$. A função $f(r') = c_j \cdot r' \pmod 1$, para números reais r', é representada na figura 1.2. (Note-se que por conveniência o eixo horizontal é maior que o eixo vertical.)

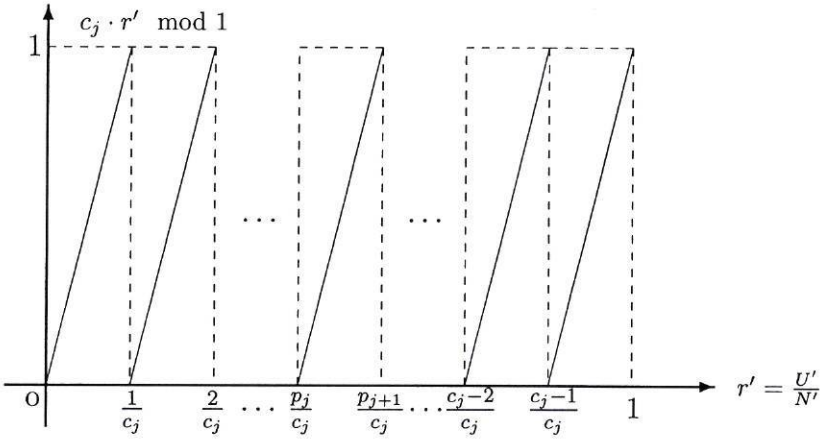


Figura 1.2: *j–ésima função serra.*

Primeiramente, calculamos um ponto r'_0 no eixo r' que verifique a desigualdade 1.30. A existência deste ponto é garantida porque é um requisito de construção do sistema de cifragem. Consideremos o valor r'_0 o valor minimizante de $g(r') = \sum_{j=1}^{n} c_j \cdot r' \pmod 1$.

Vamos procurar encontrar um intervalo $[r_1, r_2]$ em que todos os seus pontos verificam a desigualdade 1.30.

Se imaginarmos todas as j funções serra sobrepostas, obtemos n rectas que intersectam a recta $r' = r'_0$. Em cada uma das rectas obtidas temos um minimizante p_j dessa recta, ou seja, p_j é o p_j–ésimo minimizante da j–ésima função serra (figura 1.2). Podemos encontrar as variáveis p_j resolvendo os dois sistemas de desigualdades,

$$\begin{array}{ll} 1 \leq p_1 \leq c_1 - 1, & -\varepsilon_2 \leq \frac{p_1}{c_1} - \frac{p_2}{c_2} \leq \varepsilon'_2 \\ 1 \leq p_2 \leq c_2 - 1, & -\varepsilon_3 \leq \frac{p_1}{c_1} - \frac{p_3}{c_3} \leq \varepsilon'_3 \\ \cdots & \cdots \\ 1 \leq p_n \leq c_n - 1, & -\varepsilon_n \leq \frac{p_1}{c_1} - \frac{p_n}{c_n} \leq \varepsilon'_n \end{array} \tag{1.31}$$

através do algoritmo polinomial de *Lenstra* (ver [24]).

Considere p_j um dos valores determinados pelo processo anterior e sejam $r'_1, \ldots r'_k$ os pontos de descontinuidade de todas as funções serra tal que

$$r'_1, \ldots r'_k \in \left[\frac{p_j}{c_j}, \frac{p_j + 1}{c_j}\right[,$$

ordenados de forma crescente. Entre dois pontos de descontinuidade r'_t e r'_{t+1}, cada uma das j funções serra representa uma recta de equação

$$r' \cdot c_j - b^t_j, \quad r'_t \leq r' \leq t'_{t+1}$$

em que b^t_j denota o minimizante da recta que está entre r'_t e r'_{t+1} pertencente à j–ésima função serra.

Para cada t entre 1 e k, as condições 1.29 e 1.30 podem ser formuladas como um sistema de desigualdades em r', tal que $r'_t < r' < r'_{t+1}$:

$$\sum_{j=1}^{n}(r' \cdot c_j - b^t_j) < 1, \quad \text{e}$$

$$(r' \cdot c_i - b^t_i) > \sum_{j=1}^{i-1}(r' \cdot c_j - b^t_j), \quad \text{para } i = 1, \ldots, n.$$

A solução do sistema anterior fornece um sub-intervalo $]r'_t, r'_{t+1}[$. Todo o ponto $r' = \frac{U'}{N'}$ pertencente a este sub-intervalo é um par de parâmetros que permite construir um conjunto supercrescente com o qual podemos decifrar mensagens cifradas com o conjunto público $C = (c_1, \ldots, c_n)$.

Note-se que *Shamir* não obteve um algoritmo polinomial que resolve um problema $\mathcal{NP}$-completo. O algoritmo explora uma falha de segurança que reside na transformação modular do conjunto privado D no conjunto público C. Encontrou-se uma forma de construir o conjunto supercrescente D' a partir do conjunto público C.

1.4 Assinatura digital

Uma das partes mais importantes de uma mensagem é a *assinatura.* A assinatura certifica de que o documento é realmente de quem se diz ser. Na Idade Média, os nobres chancelavam as cartas com o cunho de um anel no lacre, uma marca de família que mais ninguém possuia. Hoje em dia, uma caneta no punho de um indivíduo produz uma assinatura idiossincrásica que constitui parte física e integrante de documentos como cheques,

contratos, compras e documentos oficiais. É supostamente difícil de reproduzir mas, como todos sabemos, não é impossível forjar uma assinatura escrita à mão.

Numa comunicação electrónica onde não é possível enviar uma assinatura física, pode ser necessário recorrer a outros métodos. Por exemplo: o gerente de uma empresa que pretende efectuar uma transferência bancária via telefone tem de fornecer, à operadora bancária, informações pessoais que somente o banco e o gerente conhecem.

A criptografia de chave pública permite criar esquemas de *assinatura digital* – um modo simples e elegante de provar que uma mensagem é de quem se diz ser, suprimindo qualquer possibilidade de forjar a nossa identidade. No entanto, as assinaturas digitais, como as convencionais, podem ser forjadas. A diferença é que a assinatura digital pode ser matematicamente verificada atestando a sua integridade e autenticidade.

1.4.1 Esquemas de assinaturas digitais

Considere (C_P, D_P) o par de chaves pública e privada do Pedro. Qualquer pessoa pode usar a chave pública C_P do Pedro para cifrar uma mensagem M, calculando $m = f_{C_P}(M)$. Mas o Pedro, o legítimo proprietário da chave privada, é o único que pode decifrar a mensagem, ou seja, $M = f_{D_P}(m)$. Invertendo a ordem de uso das chaves pública e privada, obtém-se uma mensagem $S = f_{D_P}(M)$, a *assinatura digital*, que só pode ter sido cifrada pelo Pedro, mas que pode ser decifrada por qualquer pessoa, visto que C_P e $M = f_{C_P}(S)$. Claro que não há privacidade na mensagem S, mas o objectivo é obter um efeito de personalização do documento M, realizável por uma única pessoa, algo semelhante ao efeito de uma assinatura. Um esquema deste tipo é denominado de *assinatura digital.* Para que este esquema funcione, o criptossistema de chave pública utilizado deve satisfazer a propriedade seguinte.

Propriedade 1.4.1 *Dado um sistema de cifragem de chave pública, as funções de decifragem e de cifragem são inversas, ou seja, para qualquer indivíduo I,*

$$f_{C_I}(f_{D_I}(M)) = M.$$

Um esquema de assinatura digital deve possuir ainda as seguintes propriedades:

- A assinatura S é autêntica quando qualquer pessoa, usando a chave pública C_P,

restaura a mensagem original $M = f_{C_P}^{-1}(S)$ para confirmar que foi o Pedro, e só o Pedro que assinou a mensagem.

- A assinatura não pode ser falsificada, porque só o Pedro conhece sua chave privada D_P, e mais ninguém pode assinar o documento em lugar do Pedro.
- A alteração de um documento assinado invalida a sua assinatura.
- A assinatura não deve ser reutilizável: deve ser uma função do documento que não pode ser transferida para outro documento.
- A assinatura não pode ser rejeitada, isto é, o Pedro não pode posteriormente negar ter assinado o documento.

O DSS (Digital Signature Standard) é o padrão adoptado a 1 de Dezembro de 1994 pelo NIST (National Institute of Standards and Technology, EUA). O algoritmo de assinatura digital proposto é o DSA (Digital Signature Algorithm), desenvolvido pela NSA (Nacional Security Agency, EUA), que tem por base o problema de logaritmos discretos, tal como o método de troca de chaves de *Diffie* e *Hellman* [25]. O DSA só pode ser usado como esquema de assinatura. Mas o esquema de assinatura digital que vamos apresentar baseia-se no criptossistema de chave pública RSA, que permite combinar a assinatura e a cifragem de mensagens.

1.4.2 Assinatura no RSA

Cada utilizador I possui uma chave pública (n_I, c_I) e uma chave privada (n_I, d_I) do sistema de cifragem RSA. Suponhamos que o Pedro pretende enviar ao João uma mensagem M, assinada e cifrada.

- Primeiramente, o Pedro vai assinar a mensagem M aplicando a sua chave privada (n_P, d_P), obtendo:

$$S \equiv M^{d_P} \pmod{n_P}.$$

- Para cifrar a mensagem, o Pedro usa a chave pública do João (n_J, c_J):

$$X \equiv S^{c_J} \pmod{n_J}.$$

Note-se que o Pedro é o único que conhece a chave privada d_P, logo ele é o único que consegue produzir a assinatura digital S. Como a mensagem assinada foi cifrada com a chave pública do João, só ele a poderá decifrar! O Pedro pode mandar a mensagem X, assinada e cifrada, para o João através de um canal público.

- O João, ao receber X, pode usar a sua chave privada (n_J, d_J), para decifrar a mensagem:

$$S \equiv X^{d_J} \pmod{n_J}.$$

- Agora que o João está na posse da mensagem assinada S, ele pode usar a chave pública do Pedro (n_P, c_P), para verificar a autencidade da mensagem calculando:

$$M \equiv S^{c_P} \pmod{n_P}.$$

Este esquema funciona porque a propriedade 1.4.1 é garantida por argumento semelhante ao utilizado no Teorema 1.2.1.

Pode ser necessário dividir a mensagem em blocos M_i, para que o valor numérico de cada bloco M_i seja menor que n_P. Por outro lado, caso $n_J < n_P$, pode acontecer que os blocos assinados S_i excedam o módulo n_J. Neste caso, basta reajustar o tamanho dos blocos para que se verifique $S_i' < n_J$ (Note-se que os módulos são parte da chave pública).

Outro método consiste em escolher um valor limite h, por exemplo $h = 10^{299}$. Um utilizador pode possuir dois pares de chave pública RSA, um para a cifragem e outro para assinatura. Todo módulo n da chave usada para assinatura seria menor que h e todo o módulo usado para a cifragem de dados seria maior que h. Desta forma, desaparece a necessidade de reajustar o tamanho dos blocos M_i.

1.4.3 Funções unidireccionais (*hash*)

Na prática, os algoritmos de chave pública para esquemas de assinatura digital são muitas vezes ineficientes para assinar mensagens longas. De facto, cifrar mensagens longas pode demorar muito tempo. Por outro lado, uma mensagem assinada pelo processo descrito na secção 1.4.2 não permite visualizar a mensagem sem verificar a assinatura. Uma melhoria dos esquemas de assinatura digital consiste em implementar funções unidireccionais (*one-way hash functions*).

A maioria das funções unidireccionais são implementadas a partir de uma função f que produz uma saída de tamanho fixo de m bits, dadas duas entradas de m bits cada. Uma entrada consiste num bloco de texto da mensagem M e outra no *hash* resultante do processamento do bloco anterior. Matematicamente, tem-se que $h_i = f(M_i, h_{i-1})$. A saída do último bloco torna-se o valor *hash* de toda a mensagem. Desta maneira, uma função unidireccional produz sempre uma saída de tamanho fixo, independente do tamanho da mensagem. A fim de resolver um eventual problema de segurança resultante do facto de duas mensagens de comprimento diferentes produzirem o mesmo valor de *hash*, por vezes alguma informação binária sobre o tamanho da mensagem M é concatenada a M antes de iniciar o cálculo do valor de *hash*.

Em resumo, uma função *hash* unidireccional opera sobre uma mensagem M de qualquer tamanho, e produz um valor de *hash* $h(M)$ de tamanho fixo. Estas funções devem ter características adicionais:

- Dado M, deve ser rápido e fácil calcular $h(M)$.
- Dado $h(M)$, é muito demorado e difícil calcular M.
- Dado M, deve ser muito difícil encontrar outra mensagem M' tal que $h(M) = h(M')$, ou seja, mudando um só bit da mensagem original M, h produz um valor de *hash* totalmente diferente.

O algoritmo mais utilizado para funções unidireccionais é o MD5 (Message Digest Algorithm) desenvolvido por *Ron Rivest* e que produz um valor *hash* de 128 bits. Em 1996, *Hans Dobbertin* quase quebrou este algoritmo, descobrindo sérias fraquezas do MD5. Outro algoritmo, projectado pelo NIST (Nacional Institute of Standards and Technology) e NSA (National Security Agency), é o SHA-1 (Secure Hash Algorithm), considerado por vários especialistas uma função unidireccional criptograficamente segura, que produz um hash de 160 bits, em vez de 128 bits. Todas as novas versões do PGP usam a função unidireccional SHA-1 para a criação de assinaturas digitais com o DSS. Mas, por questões de compatibilidade com antigas versões, o PGP usa o MD5 para assinar mensagens com o RSA.

Em vez de assinar a mensagem inteira, assinamos um pequeno pedaço de dados que "representa" a mensagem. Para isso, calcula-se o valor *hash* da mensagem, também

chamado *message digest* ou uma *impressão digital.* Assina-se somente a *impressão digital*, que normalmente é de tamanho reduzido (entre 128 e 512 bits).

Suponha-se que o Pedro pretende enviar ao João uma mensagem M assinada usando este esquema. Se o Pedro usa o sistema de cifragem RSA com chave pública (n_P, c_P) e chave privada (n_P, d_P), então ele seguirá os seguintes passos:

- Calcula a *impressão digital* h da mensagem M, através da função unidireccional MD5, ou seja, $h = \text{MD5}(M)$.
- Assina somente a *impressão digital* com a sua chave privada (n_P, d_P):

$$S \equiv h^{d_P} \pmod{n_P}.$$

  A assinatura da mensagem M é o valor S que só o Pedro consegue calcular, uma vez que só ele conhece d_P.

- Envia (M, S), a mensagem e a assinatura, para o João.

Para que o João verifique a autenticidade da assinatura S basta:

- Calcular a *impressão digital* da mensagem, $h' = \text{MD5}(M)$.
- Restaurar a *impressão digital* assinada, usando a chave pública do Pedro, calculando:

$$h \equiv S^{c_P} \pmod{n_P}.$$

  Note-se que qualquer pessoa poderia efectuar esta operação, uma vez que c_P é um parâmetro da chave pública RSA.

- Se h e h' são iguais, então a mensagem M e a assinatura S são consideradas válidas, certificando que a mensagem não foi modificada após a assinatura da mesma.

A rejeição da assinatura significa que a mensagem ou a assinatura sofreram alterações. Infelizmente, a assinatura digital pode apenas certificar que a mensagem foi modificada, mas não o que foi modificado e quanto foi modificado.

Este esquema tem vantagens adicionais.

- O documento e a assinatura podem ser guardados em locais diferentes, o que dificulta ainda mais as falsificações.

- O documento pode ser armazenado em forma legível, o que facilita o acesso e a leitura.
- O espaço necessário para guardar a assinatura é bem pequeno.

E o mais interessante, relacionado com privacidade e outros aspectos legais:

- O documento pode ser mantido secreto; somente a sua assinatura necessita de ser tornada pública. Só quando a autoria de um documento, ou de uma ideia, tenha de ser comprovada é que o documento precisa ser tornado público.

Por outro lado, um utilizador I pode querer proteger apenas a autenticidade de uma mensagem sem esconder o seu conteúdo. Tais esquemas são denominados MAC (Message Authentication Code) que passamos a descrever:

- Geramos uma chave secreta K.
- Concatenamos a chave secreta K à mensagem M e calculamos a *impressão digital*:

$$h = \text{MD5}(M \parallel K).$$

(O símbolo $\parallel$ representa a concatenação)

- Com a chave privada do utilizador I, assinamos o valor h,

$$S \equiv h^{d_I} \pmod{n_I}.$$

Somente alguém que conhece a chave K pode repetir o processo sobre M para verificar a autenticidade da mensagem M. Isto impede terceiros de forjar uma assinatura de um documento modificado M' a partir do documento M, uma vez que também necessitam da chave privada K para assinar o documento M'!

Capítulo 2

Geração de números pseudoaleatórios e algumas aplicações

2.1 Introdução

As sequências de números aleatórios são uma importante fonte de recursos, de uso bastante variado. São usadas para testar *microprocessadores* de computadores e para programar *slot machines*. Permitem conceber algoritmos probabilísticos eficientes que resolvem problemas difíceis de solucionar por meio de algoritmos determinísticos.

Por exemplo, o cálculo de um integral pode ser muito difícil mesmo através de integração numérica, mas pode ser rapidamente obtido recorrendo a pequenos programas que usam sequências aleatórias. Um método que use sequências de números aleatórios para efectuar cálculos é também denominado " *método de Monte Carlo*" 2.3.

Muitos criptossistemas necessitam de gerar quantidades imprevisíveis que devem ser mantidas secretas, que não podem ser descobertas por terceiros. Por exemplo, são necessários números aleatórios para gerar chaves privadas e públicas de criptossistemas assimétricos tal como RSA (§1.2), DSA [25], e *Diffie e Hellman* [25]. As chaves secretas para criptossistemas simétricos são também geradas aleatoriamente. A cifra de Vernam – o sistema de cifragem mais seguro – necessita de uma quantidade de números aleatórios para gerar a chave secreta tão grande como a quantidade de letras da mensagem a enviar!

(ver exemplo 1.0.1).

2.2 Técnicas de geração

A aleatoriedade "pura" deriva de processos físicos imprevisíveis: a turbulência de ar num disco rígido, por exemplo, ou a desintegração radioactiva de determinado isótopo de Césio medida por um *contador Geiger*[1].

Tais processos são denominados "*geradores de números aleatórios*" (GNA). Estes geradores são usualmente implementados em dispositivos físicos que tendem a ser lentos, de difícil implementação. Também pode acontecer que estes dispositivos sejam influenciados por factores externos e que os bits produzidos sejam tendenciosos. Isto significa que a probabilidade de emitir um bit 1 não é $\frac{1}{2}$. Por outro lado, a probabilidade de o GNA emitir o bit 1 pode depender dos bits anteriores, ou seja, os bits são correlacionados.

Suponha-se que um GNA produz uma sequência de bits tendenciosos mas não correlacionados. Existem várias técnicas de triagem para produzir uma sequência de bits verdadeiramente aleatória. *Von Neumann* [29] propôs agrupar os bits em pares e transformar os pares "01" em 0, os pares "10" em 1 e remover os pares "00" e "11". O resultado é uma sequência de bits não correlacionados e não tendenciosos.

Também pode acontecer que os GNA sofram avarias *físicas*. Portanto, os GNA devem ser frequentemente alvo de testes para certificar que funcionam correctamente.

Visto que os geradores facultam quantidades imprevisíveis que por vezes podem estar sujeitas a influências, é importante que a quantidade de informação que se pode extrair do resultado produzido pelo gerador seja tão próxima quanto possível do seu "tamanho" em bits. Vamos formular primeiro algumas definições para relacionar o conceito de entropia com uma das principais características que as sequências de números aleatórios devem satisfazer.

2.2.1 Quantidade de informação

Seja X uma *variável aleatória* que toma valores num conjunto finito $\Omega = \{x_1, x_2, \ldots, x_n\}$, com probabilidade $P(X = x_i) = p_i$, onde $0 \leq p_i \leq 1$ para cada $i, 1 \leq i \leq n$, e onde

[1] Instrumento que serve para detectar radiações ionizantes, quer corpusculares, quer electromagnéticas.

$\sum_{i=1}^{n} p_i = 1$. Seja f_i a frequência de cada um dos eventos x_i possíveis da variável aleatória X.

A unidade de informação é o *bit* que tem dois estados possíveis, 0 ou 1. Uma codificação binária é uma função injectiva

$$f : \Omega \to \{0,1\}^b, \quad b \text{ é inteiro}$$

que associa uma quantidade menor de bits a estados da variável aleatória X mais prováveis. Cada evento de probabilidade pode ser codificado com $-\log_2(P(X = x_i))$ bits. Definimos desta forma a quantidade de informação obtida pela ocorrência do evento x_i por $I_i = -\log_2(P(X = x_i))$, o número de bits necessários para codificar o evento x_i.

Exemplo 2.2.1 Seja X uma variável aleatória e $\Omega = \{x_1, x_2, x_3\}$, onde $p(X = x_1) = \frac{1}{2}$ e $p(X = x_2) = p(X = x_3) = \frac{1}{4} = \frac{1}{2^2}$. Como $-\log_2(\frac{1}{2}) = 1$ e $-\log_2(\frac{1}{4}) = 2$, definimos a codificação binária f injectiva em que: $f(x_1) = 0$, $f(x_2) = 10$ e $f(x_3) = 11$.

Pretendemos encontrar uma expressão para a quantidade de informação *média* para todos os possíveis estados da variável aleatória X.

Seja $S = s_1 \ldots s_m$ uma mensagem, onde cada $s_i \in \Omega$. Se a frequência de x_i na mensagem S é denotada por f_i, então a probabilidade $p_i = P(X = x_i) = \frac{f_i}{m}$, para cada i. O número de bits usado para a codificação binária de cada x_i é $b_i = -\log_2 p_i$. O número total de bits necessários para a codificação da mensagem S é $-\sum_{i=1}^{n} f_i \log_2 p_i$. É fácil verificar que a média de bits por cada s_i na mensagem S é $H(S) = -\sum_{i=1}^{n} \frac{f_i}{m} \log_2 p_i = -\sum_{i=1}^{n} p_i \log_2 p_i$.

Definição 2.2.1 *A* entropia *da variável aleatória X, que toma valores no conjunto Ω, é definida por*

$$H(X) = -K \sum_{i=1}^{n} p_i \log_2 p_i = K \sum_{i=1}^{n} p_i \log_2 \left(\frac{1}{p_i}\right),$$

onde K representa uma constante opcional. Por convenção, se $p_i = 0$ então $p_i \log_2 p_i = p_i \log_2 \left(\frac{1}{p_i}\right) = 0$.

A *entropia* de X é uma medida da quantidade de informação obtida pela observação da variável aleatória X ou, de forma equivalente, a quantidade de incerteza que vamos obter ao realizar a observação.

Propriedades da entropia. Seja X uma variável aleatória que pode tomar valores num conjunto finito de n elementos. Então:

(i) $0 \leq H(X) \leq \log_2(n)$

(ii) $H(X) = 0 \Leftrightarrow p_i = 1$ para algum i, e $p_j = 0$ para todo $j \neq i$ (Não há incerteza sob o que se vai obter ao realizar a observação)

(iii) $H(x) = \log_2 n \Leftrightarrow p_i = \frac{1}{n}$ para cada $i, 1 \leq i \leq n$ (Toda a informação obtida pela observação de X é "igual")

Considere um gerador de números aleatórios que produziu um resultado com n-bits. Seja X uma variável aleatória que toma valores no conjunto $\Omega = \{0, 1, \ldots, 2^n\}$ e $p_i = P(X = x_i)$ é a probabilidade do nosso gerador produzir x_i, onde $0 \leq x_i \leq 2^n$.

Para que o gerador se aproxime da perfeição, cada p_i deve ter valor igual a $\frac{1}{2^n}$ e a entropia do resultado produzido deve ser igual a n. Isto significa que qualquer valor x_i tem a mesma probabilidade de ocorrer e que a informação extraída do resultado produzido pelo gerador não pode ser representada, ou codificada, por uma sequência de tamanho inferior que n-bits.

2.2.2 Geradores de números pseudoaleatórios

Na prática, pode ser necessário reproduzir várias vezes a *mesma sequência* de números aleatórios para repetir simulações numéricas, por exemplo no método de *Monte Carlo* (ver 2.3. Por outro lado, pode ser impraticável produzir "grandes" quantidades de números aleatórios puros para a cifra de *Vernam*. É preferível guardar uma pequena chave verdadeiramente aleatória e, quando necessário, expandi-la numa sequência aleatória grande. Isto motiva o uso de *sequências de números pseudoaleatórios.*

As sequências de números pseudoaleatórios são sequências finitas de números produzidos por algoritmos eficientes e determinísticos gerados partindo de um valor de entrada: a "*semente inicial*". Tais algoritmos determinísticos são chamados geradores de números pseudoaleatórios (GNPA).

Para gerar números pseudoaleatórios, procuramos obter amostras independentes de uma distribuição de probabilidade num determinado conjunto. Em particular, a tarefa

de gerar números pseudoaleatórios pode ser reduzida a obter amostras independentes de *bits* de uma distribuição de probabilidade no conjunto $\{0, 1\}$, tarefa que chamamos de *gerador de bits pseudoaleatórios*. Um número inteiro entre 0 e n pode ser obtido gerando uma sequência de *bits pseudoaleatórios* de tamanho $\lfloor \log n \rfloor + 1$ e verificando que o número gerado não excede n.

O principal objectivo é usar uma pequena sequência verdadeiramente aleatória para expandir e criar uma sequência grande de modo que o adversário não consiga distinguir entre uma verdadeira sequência aleatória e a sequência produzida pelo GNPA. De facto, um GNPA pode produzir sequências que sejam muito difíceis de distinguir de uma verdadeira sequência de números aleatórios. O GNPA deve ter outras características:

- Dada a mesma semente deverá produzir sempre a mesma sequência de números pseudoaleatórios.
- Deve ser rápido e de baixo custo computacional.
- Deve ter um período grande. GNPA com pequenos períodos causam a repetição de eventos na sequência produzida.
- Valores sucessivos devem ser independentes e uniformemente distribuídos. A correlação entre valores sucessivos deve ser muito pequena.

Se um GNPA tomar como valor de entrada uma semente de 256 bits, então este GNPA não poderá produzir uma quantidade de números realmente aleatórios maior do que 256 bits. Mas adivinhar uma semente de 256 bits é algo que é muito difícil. No entanto, para aplicações de criptografia, certas precauções adicionais devem ser tomadas relativamente à selecção da semente para aplicar em GNPA.

É necessário certificar que um GNPA não use uma semente que seja previsível e que o resultado produzido não possa ser distinguido de uma verdadeira sequência de números aleatórios, ou seja, que nenhuma informação estatística possa ser extraída.

Como não é possível obter aleatoriedade pura dentro de um sistema determinístico, por vezes, certos programadores tentam obter *entropia*, secção 2.2.1, para a semente inicial a partir de dispositivos de entrada/saída de um computador. Por exemplo: tempo de acesso a discos rígidos, endereços de processos ou percentagens de recursos em uso, relógio de

sistema. No *PGP v7.0.3*, é pedido ao utilizador que mexa no rato e nas teclas para a produção de novas chaves!

Para a geração da semente inicial deve-se usar o máximo de recursos disponíveis. Cada fonte de aleatoriedade deve ser combinada através de uma função unidireccional criptográfica 1.4.3. Isto permite encadear as sequências de cada fonte numa única sequência e "destilar" os "verdadeiros" bits aleatórios.

Mesmo que todos os cuidados relativos à selecção da semente inicial sejam tomados, pode acontecer que dada uma sequência suficientemente longa produzida por um GNPA, seja possível prever com um algoritmo polinomial os parâmetros que caracterizam o gerador. Ou seja, o GNPA é previsível. Por exemplo: o gerador congruência linear 2.2.3 e o gerador $\frac{1}{p}$ são previsíveis (ver [21]).

Vamos estudar com algum pormenor o gerador *congruência linear* que é, de facto, o mais comunmente usado na geração de números pseudoaleatórios de maneira rápida na maioria dos sistemas operativos e pacotes de software matemático (Maple, Mathematica, Matlab, etc) e também no PGP e no GnuPG.

2.2.3 Gerador congruência linear

Esta função é uma das mais utilizadas para gerar números pseudoaleatórios e tem como principal aplicação servir simulações e algoritmos probabilísticos.

Considere as quantidades inteiras positivas secretas a, b e m de tal forma que m é maior que a e que b. Para um inteiro $x < m$, definimos recursivamente as sequências infinitas $x_0, x_1, \ldots, x_i, \ldots$ e $x'_1, \ldots, x'_i, \ldots$ por

$$x_i \equiv \begin{cases} x & \text{se } i = 0 \\ ax_{i-1} + b \pmod{m} & \text{se } i > 0; \end{cases} \tag{2.1}$$

e

$$x'_i \equiv x_i - x_{i-1} \pmod{m}, \quad \text{para } i \geq 1. \tag{2.2}$$

Definição 2.2.2 *Definimos o* gerador congruência linear multiplicativo, *GCL, pela função recursiva* 2.1, *ou seja,*

$$\mathrm{GCL}(x_0, a, b, m) = x_0, x_1, \ldots, x_i, \ldots.$$

Os parâmetros a, b e m descrevem a função GCL *e x_0 é a semente, o primeiro termo da sequência.*

Exemplo 2.2.2 $\mathrm{GCL}(3, 11, 7, 39) = 3, 1, 18, 10, 0, 7, 6, 34, 30, 25, 9, 28, 3, 1, 18, 10 \ldots$

A qualidade deste GNPA depende da escolha dos seus parâmetros, definição 2.1, que são previsíveis por algoritmos em tempo polinomial, ou seja, dada parte de uma sequência obtida por este gerador podemos reconstruir o resto da sequência, mesmo sem conhecer os parâmetros m, a e b. Logo, este gerador é inadequado para ser usado em criptografia.

O próximo lema garante que sempre se consegue achar um inteiro diferente de 1 que é divisor comum de todos os elementos iniciais da sequência GCL.

Lema 2.2.1 *Para cada $i \geq 1$, sejam $g_i = \mathrm{mdc}(x'_1, x'_2, \ldots, x'_i)$ e i_0 o menor inteiro $i \geq 1$ tal que $g_i \mid x'_{i+1}$. Então têm-se $i_0 \leq 2 + \lfloor \log_2 m \rfloor$ e, além disso, $g_i = g_{i_0}$ para todos os $i \geq i_0$.*

Demonstração. Sabemos que $g_1 = x'_1$ e que, para todo i, $g_{i+1} = (g_i, x'_{i+1})$. Podemos ver facilmente que, se $g_i \nmid x'_{i+1}$, então $g_{i+1} \leq \frac{g_i}{2}$. Como i_0 é o menor i tal que $g_i \mid x'_{i+1}$, então para j entre 1 e $i_0 - 1$, temos $g_j \nmid x'_{j+1}$. Portanto

$$g_{i_0-1} \leq \frac{g_{i-2}}{2} \leq \ldots \leq \frac{g_1}{2^{i_0-2}}.$$

Como g_i é inteiro positivo, e 1 é divisor de qualquer inteiro, tem de existir $i_0 \geq 1$, o menor inteiro tal que $g_{i_0} \mid x'_{i+1}$.

Resulta da expressão anterior que, $2^{i_0-2} \leq \frac{g_1}{g_{i_0-1}} < m$ e isto prova que $i_0 \leq 2 + \lfloor \log_2 m \rfloor$. Se $g_{i_0} \mid x'_{i_0+1}$, então $g_{i_0+1} = (g_{i_0}, x'_{i_0+1}) = g_{i_0}$ e todos os g_i são idênticos a g_{i_0}, para $i \geq i_0$. □

Exemplo 2.2.3 Para a sequência do exemplo 2.2.2 produzida por $\mathrm{GCL}(3, 11, 7, 39)$, a sequência x'_i é dada por,

$$37, 17, 31, 29, 7, 38, 28, 35, 34, 23, 20, 14, 37, 17, 31, \ldots$$

Pelo lema 2.2.1, o índice $i_0 \leq 2 + \lfloor \log_2 39 \rfloor = 7$. A sequência g_i é dada por,

$$37, 1 = (37, 17), 1 = (37, 17, 31), \ldots$$

Visto que $x'_2 = 17$ temos $g_1 = 37 \nmid 17$ e, como $x'_3 = 31$ e $g_2 = 1 \mid 31$, fica provado que $i_0 = 2 \leq 7$.

Os algoritmos que tornam o gerador congruência linear previsível são traduzidos pelos Teoremas 1 e 2 de Plumbstead [21, 30].

Com o seguinte Teorema podemos, conhecido o módulo m, determinar os parâmetros a e b a menos de módulo, de tal maneira que, se consiga replicar a mesma sequência de números pseudoaleatórios.

Teorema 2.2.2 (Plumbstead 1) *Considere a sequência $x_0, x_1, \ldots, x_{i_0}, x_{i_0+1}$ produzida pela função recursiva* $\mathrm{GCL}(x_0, a, b, m)$, *onde $i_0 \leq 2 + \lfloor \log_2 m \rfloor$ é o menor $i \geq 1$ tal que $g_i \mid x'_{i+1}$. Existe um algoritmo polinomial, de custo $O(\log_2(m))$, que calcula inteiros a' e b' tais que, para todo $i \geq 1$, $x_i \equiv a'x_{i-1} + b' \pmod{m}$.*

Demonstração. O algoritmo é definido pelos seguintes passos:

Dados: $[x_0, x_1, \ldots, x_{t+1}]$

1: Calcule $x'_i \equiv x_i - x_{i-1}$, para $1 \leq i \leq t+1$.

2: Calcule $d = \mathrm{mdc}(x'_1, x'_2, \ldots, x'_t)$.

3: Calcule $u_1, \ldots, u_t$ tal que $d = \sum_{i=1}^{t} u_i x'_i$.

Resultado: $a' = \sum_{i=1}^{t} u_i \frac{x'_{i+1}}{d}$ e $b' = x_1 - a'x_0$.

É importante notar que a' e b' são calculados sem recorrer ao módulo m.

Vamos primeiro provar que $a'x'_i \equiv x'_{i+1} \pmod{m}$, para todo $i \geq 1$.
Seja $g = (m, d)$. Então

$$ad \equiv a\sum_{i=1}^{t} u_i x'_i \equiv \sum_{i=1}^{t} u_i a x'_i \equiv \sum_{i=1}^{t} u_i x'_{i+1} \equiv d\sum_{i=1}^{t} u_i \frac{x'_{i+1}}{d} \equiv a'd \pmod{m}$$

A congruência $ad \equiv a'd \pmod{m}$ é equivalente a $m \mid (a - a')d$. Pela definição de g temos $\frac{m}{g} \mid (a - a')\frac{d}{g}$. Como $(\frac{m}{g}, \frac{d}{g}) = 1$, resulta que $\frac{m}{g} \mid (a - a')$. Logo $a \equiv a' \pmod{\frac{m}{g}}$.

Para cada $i \geq 1$, temos que $g \mid (x'_i, m)$, ou seja, existe um inteiro l tal que $(x'_i, m) = gl$. Desta forma, existe um inteiro k tal que

$$\frac{m}{g} \mid (a - a') \Leftrightarrow (a - a') = \frac{m}{g}k \Leftrightarrow (a - a') = \frac{m}{gl}kl,$$

e portanto $\frac{m}{(x'_i, m)} \mid (a - a')$.

Obtemos que, para todo o $i \geq 1$,

$$a \equiv a' \quad \left(\mathrm{mod}\ \frac{m}{(x'_i, m)}\right) \tag{2.3}$$

Mas a é uma solução da congruência

$$ux'_i \equiv x'_{i+1} \pmod{m}, \tag{2.4}$$

e, pelo Teorema 1.2.25, toda a solução de 2.4 é da forma

$$a + \frac{hm}{(x'_i, m)}, \quad h = 0, 1, \ldots, (x'_i, m) - 1.$$

Portanto, pela congruência 2.3, resulta também que a' é solução da congruência 2.4, isto é, $a'x'_i \equiv x'_{i+1} \pmod{m}$.

Para concluir a prova, vamos mostrar que $a'x_i + b' - x_{i+1} \equiv 0 \pmod{m}$ usando a própria definição 2.1:

$$\begin{aligned} a'x_i + b' - x_{i+1} &\equiv a'x_i + (x_1 - a'x_0) - x_{i+1} \equiv a'(x_i - x_0) - (x_{i+1} - x_1) \\ &\equiv a' \sum_{j=1}^{i} (x_j - x_{j-1}) - \sum_{j=1}^{i} (x_{j+1} - x_j) \equiv a' \sum_{j=1}^{i} x'_j - \sum_{j=1}^{i} x'_{j+1} \\ &\equiv \sum_{j=1}^{i} (a'x'_j - x'_{j+1}) \equiv 0 \pmod{m}. \end{aligned}$$

□

Observemos que o Teorema 2.2.2 não permite ainda determinar o módulo m a partir da sequência gerada por $\mathrm{GCL}(x_0, a, b, m)$, pois m é uma das suposições na prova deste resultado. Não se conhece um número mínimo de elementos consecutivos da sequência 2.1 que permita calcular o módulo m' para a sequência inteira. Porém, pelo Teorema 2.2.4 abaixo, é possível encontrar um módulo para segmentos de sequências 2.1 suficientemente grandes.

Considere uma sequência gerada por 2.1 com comprimento maior ou igual a $2 + \lfloor \log_2 m \rfloor$. O multiplicador a' e o incremento b' podem ser calculados usando o Teorema 2.2.2, mesmo que desconheçamos o módulo m.

Para um dado inteiro M, definimos a seguinte sequência por indução sobre i:

$$x_i(M) \equiv \begin{cases} x_0 & \text{se } i = 0; \\ a'x_{i-1}(M) + b' \pmod{M} & \text{se } i > 0. \end{cases} \tag{2.5}$$

Lema 2.2.3 *Se $M^* \mid M$, então $x_i(M) \equiv x_i(M^*) \pmod{M^*}$, para todo $i \geq 0$.*

Demonstração. Vamos provar o resultado por indução sobre i.

Pela definição 2.5 temos que $x_0(M^*) = x_0$. Suponhamos que o lema é válido para i, isto é,

$$x_i(M) \equiv x_i(M^*) \pmod{M^*}. \tag{2.6}$$

Queremos provar que esta relação vale para $i+1$.

Por hipótese $M^* \mid M$ logo existe um inteiro l tal que $M = lM^*$. Usando 2.5 obtemos, $x_{i+1}(M) \equiv a'x_i(M) + b' \pmod{M} \Leftrightarrow x_{i+1}(M) = a'x_i(M) + b' + kM \Leftrightarrow x_{i+1}(M) = a'x_i(M) + b' + klM^*$, para algum $k \in \mathbb{Z}$.

Portanto pela hipótese 2.6

$$x_{i+1}(M) \equiv a'x_i(M) + b' \pmod{M^*} \Leftrightarrow x_{i+1}(M) \equiv a'x_i(M^*) + b' \pmod{M^*},$$

e de 2.5 concluímos $x_{i+1}(M) \equiv x_{i+1}(M^*) \pmod{M^*}$. □

Dada uma sequência finita $x_0, x_1, \ldots, x_s$, o seguinte Teorema dá-nos um algoritmo para a previsão do módulo m', baseado no último lema.

Teorema 2.2.4 (Pumbstead 2) *Considere a sequência finita $x_0, x_1, \ldots, x_s$ produzida pela função $\mathrm{GCL}(x_0, a, b, m)$, em que o índice $s > i_0 = 2 + \lfloor \log_2 m \rfloor$ (o menor $i \geq 1$ tal que $g_i \mid x'_{i+1}$). Então existe um algoritmo polinomial, de custo $O(s)$, que calcula inteiros a', b' e m' tais que, para todo o $i = 1, \ldots, s$, se verifica $x_i \equiv a'x_{i-1} + b' \pmod{m'}$.*

Demonstração. No que se segue vamos convencionar $(a, \infty) = a$ e $a \pmod{\infty} = a$. Para cada módulo M, definimos $k(M)$ como *o menor* inteiro $k \geq 1$ tal que

$$x_{k+1}(M) \not\equiv x_{k+1} \pmod{M}, \tag{2.7}$$

em que $x_{k+1}(M)$ é definido como em 2.5, com a' e b' dados pelo Teorema 2.2.2.

Inicialmente, escolhemos o módulo $M = \infty$, isto é, pela convenção enunciada acima, olhamos para a sequência definida recursivamente por $x_{k+1}(M) = a'x_k + b'$, $k \geq 0$. Se $k(M) \geq s$, ou seja, $x_k(M) \equiv x_k \pmod{M}$ para todo $k \leq s$, então $m' = M$. Caso contrário, se $k(M) < s$, substituímos o valor de M por $M^* = (M, x_{k+1}(M) - x_{k+1})$. Como $M^* \mid M$, pelo lema 2.2.3 temos que $x_{k(M)+1}(M) \equiv x_{k(M)+1} \pmod{M^*}$, ou seja, $k(M^*) \geq k(M) + 1$.

Podemos assim continuar a actualizar, no máximo em s interacções, o módulo M até que $k(M) \geq s$. Então, dada uma sequência $x_0, x_1, \ldots, x_s$ produzida pela função $\mathrm{GCL}(x_0, a, b, m)$, com $s > 2 + \lfloor \log_2 m \rfloor$, podemos definir a sequência de módulos M_0, M_1, $M_2, \ldots, M_r$ que verifica para cada $i < r$,

$$M_i = \begin{cases} \infty & \text{se } i = 0, \\ M_{i-1}^* & \text{se } k(M_i) < s, \end{cases}$$

e satisfaz $k(M_r) \geq s$. O valor procurado para m' é M_r. Logo $x_i(m') \equiv x_i \pmod{m'}$, para todo $i \leq k(m')$, ou seja, $x_{i+1} \equiv a'x_i + b' \pmod{m'}$, para todo $1 \leq i \leq s$. □

Por vezes, é necessário que o período do gerador congruência linear (definido pelos parâmetros a, b, x_0 e m) seja de *comprimento máximo*, ou seja, m. É importante que o período tenha mais elementos do que a quantidade de números a usar pela aplicação.

Será possível obter período m? Se considerarmos $a = b = 1$ obtemos a sequência $x_{n+1} \equiv x_n + 1 \pmod{m}$ que obviamente tem período m. Mas infelizmente, esta sequência não tem nada de aleatório! Vamos procurar todas as possíveis escolhas para os parâmetros a e b que permitem obter um período de comprimento m para o GCL. O Teorema 2.2.5 informa como escolher os parâmetros a e b para obter uma sequência de comprimento máximo, ver [18].

Teorema 2.2.5 *O gerador congruência linear definido pelos parâmetros (a, b, x_0, m) tem período máximo m se, e só se,*

(i) b é primo com m.

(ii) $a - 1$ é múltiplo de p, para todo o primo p que divide m.

(iii) $a - 1$ é múltiplo de 4, se m é múltiplo de 4.

É fácil ver que se m é o produto de primos distintos, então só $a = 1$ produz uma sequência de comprimento máximo. Suponhamos que $m = p_1 p_2 \cdots p_k$. Pelo Teorema 2.2.5 temos que $a - 1$ é divisível por p_i, $1 \leq i \leq k$. Logo, pelo Teorema Chinês dos Restos 1.2.26, o sistema de congruências $a \equiv 1 \pmod{p_i}$, para cada $1 \leq i \leq k$, tem uma única solução módulo m, $a = 1$.

Para demonstrar o Teorema 2.2.5 vamos considerar primeiro os seguintes resultados auxiliares.

Lema 2.2.6 *Seja p um número inteiro primo e α um inteiro positivo em que $p^\alpha > 2$. Se*

$$x \equiv 1 \pmod{p^\alpha}, \quad x \not\equiv 1 \pmod{p^{\alpha+1}},$$

então

$$x^p \equiv 1 \pmod{p^{\alpha+1}}, \quad x^p \not\equiv 1 \pmod{p^{\alpha+2}}.$$

Demonstração. Temos $x = 1 + kp^\alpha$, para algum inteiro positivo k que não é múltiplo de p. Pela expansão binomial,

$$\begin{aligned} x^p &= 1 + \tbinom{p}{1} kp^\alpha + \ldots + \tbinom{p}{p-1} k^{p-1} p^{(p-1)\alpha} + k^p p^{p\alpha} \\ &= 1 + kp^{\alpha+1} \left(1 + \tfrac{1}{p}\tbinom{p}{2} kp^\alpha + \ldots + \tfrac{1}{p}\tbinom{p}{p} k^{p-1} p^{(p-1)\alpha} \right). \end{aligned}$$

Para cada $1 < i < p$, o coeficiente binomial $\binom{p}{i}$ é divisível por p, logo

$$\frac{1}{p}\binom{p}{i} k^{i-1} p^{(i-1)\alpha}$$

é divisível por $p^{(i-1)\alpha}$. O termo $\frac{1}{p}k^{p-1}p^{(p-1)\alpha}$ é divisível por p visto que $(p-1)\alpha > 1$ quando $p^\alpha > 2$. Temos $\alpha \geq 1$ porque α é um inteiro positivo. Se $\alpha = 1$, temos $p > 2$. Se $\alpha > 1$, temos $p \geq 2$. Para completar a demonstração, basta observar que $x^p \equiv 1 + kp^{\alpha+1} \pmod{p^{\alpha+2}}$. Isto implica que $x^p \equiv 1 \pmod{p^{\alpha+1}}$ e $x^p \not\equiv 1 \pmod{p^{\alpha+2}}$. □

Lema 2.2.7 *Seja $m = p_1^{\alpha_1} \cdots p_t^{\alpha_t}$ uma decomposição de m em factores primos. O período λ do* GCL *de parâmetros (x_0, a, b, m) é o mínimo múltiplo comum dos períodos dos* GCL *de parâmetros $(x_0 \pmod{p_j^{\alpha_j}}, a \pmod{p_j^{\alpha_j}}, b \pmod{p_j^{\alpha_j}}, p_j^{\alpha_j})$, $1 \leq j \leq t$.*

Demonstração. Por indução em t, basta provar que se m_1 e m_2 são primos entre si, então o comprimento da sequência GCL determinada pelos parâmetros $(x_0, a, b, m_1 m_2)$ é o mínimo múltiplo comum entre os comprimentos λ_1 e λ_2, períodos das sequências determinadas pelos parâmetros $(x_0 \pmod{m_1}, a \pmod{m_1}, b \pmod{m_1}, m_1)$ e $(x_0 \pmod{m_2}, a \pmod{m_2}, b \pmod{m_2}, m_2)$.

Sabemos que $x_n \equiv x_k \pmod{m}$ se, e só se $x_n \equiv x_k \pmod{m_1}$ e $x_n \equiv x_k \pmod{m_2}$.

A congruência $x_n \equiv x_k \pmod{m}$ equivale a $m = m_1 m_2 \mid x_n - x_k$. Logo $m_1 \mid (x_n - x_k)$ e $m_2 \mid (x_n - x_k)$, ou seja, $x_n \equiv x_k \pmod{m_1}$ e $x_n \equiv x_k \pmod{m_2}$. Reciprocamente, se $m_1 \mid (x_n - x_k)$ e $m_2 \mid (x_n - x_k)$, então $m_1 m_2 \mid (x_n - x_k)$ porque $(m_1, m_2) = 1$, ou de forma equivalente, $x_n \equiv x_k \pmod{m_1 m_2}$.

Seja $y_n = x_n \pmod{m_1}$ e $z_n = x_n \pmod{m_2}$, para todo $n \geq 0$. Seja λ' o mínimo múltiplo comum entre λ_1 e λ_2. Queremos provar que $\lambda = \lambda'$. Por um lado, se $x_n = x_{n+\lambda}$ para todo n, então temos $y_n = y_{n+\lambda}$ e $z_n = z_{n+\lambda}$. Logo, λ é múltiplo de λ_1 e de λ_2. Portanto, $\lambda' \leq \lambda$. Por outro lado, $y_n = y_{n+\lambda'}$ e $z_n = z_{n+\lambda'}$, para todo n. Logo $x_n = x_{n+\lambda'}$, ou seja, $\lambda' \geq \lambda$. □

Agora, estamos em condições de demonstrar o Teorema 2.2.5.

Demonstração do Teorema 2.2.5. Pelo Lema 2.2.6, basta provar o Teorema quando m é uma potência de um primo, porque

$$p_1^{\alpha_1} \cdots p_t^{\alpha_t} = \lambda = [\lambda_1, \ldots, \lambda_t] \leq \lambda_1 \cdots \lambda_t \leq p_1^{\alpha_1} \cdots p_t^{\alpha_t}$$

é verdade se, e só se $\lambda_i = p_i^{\alpha_i}$, para $1 \leq i \leq t$.

Seja $m = p^\alpha$, com p primo e α um inteiro positivo. Se $a = 1$ o Teorema é óbvio. Suponhamos que $a > 1$. O período da sequência é m se, e só se, cada inteiro $0 \leq x < m$ aparece na sequência uma só vez. Para isso, basta ver que o período da sequência com semente $x_0 = 0$ tem comprimento m.

Note-se que a recorrência $x_n \equiv ax_{n-1} + b \pmod{m}$ pode ser expressa em função de x_0, ou seja,

$$x_n \equiv a^n x_0 + \left(\frac{a^n - 1}{a - 1}\right) b \pmod{m}.$$

Se $x_0 = 0$, a congruência anterior reduz-se a

$$x_n \equiv \left(\frac{a^n - 1}{a - 1}\right) b \pmod{m}.$$

Se b não for primo com m, então x_n nunca pode ser igual a 1, logo a primeira condição do Teorema é necessária. O período da sequência tem comprimento m se, e só se, o mais pequeno valor de n em que $x_n = x_0 = 0$ é $n = m$.

Quando b é primo com m, o presente Teorema reduz-se ao seguinte Lema.

Lema 2.2.8 *Seja $1 < a < p^\alpha$, em que p é primo. Se λ é o mais pequeno inteiro positivo que verifica $\frac{a^\lambda - 1}{a-1} \equiv 0 \pmod{p^\alpha}$, então*

$$\lambda = p^\alpha \quad \textit{se, e só se,} \quad \begin{cases} a \equiv 1 \pmod{p} & \textit{se } p > 2, \\ a \equiv 1 \pmod{4} & \textit{se } p = 2. \end{cases}$$

Demonstração. $\Rightarrow$) Seja $\lambda = p^\alpha$. Se $a \not\equiv 1 \pmod{p}$, então $\frac{a^{p^\alpha}-1}{a-1} \equiv 0 \pmod{p^\alpha}$ é equivalente a $a^{p^\alpha} - 1 \equiv 0 \pmod{p^\alpha}$. Como $p^{\alpha-1} \mid p^\alpha$ vem que $a^{p^\alpha} \equiv 1 \pmod{p}$. Mas pelo Teorema de Fermat 1.2.7 temos $a^{p^\alpha} \equiv (a^p)^{p^{\alpha-1}} \equiv a^{p^{\alpha-1}} \equiv \cdots \equiv a \pmod{p}$. Como $a \not\equiv 1 \pmod{p}$ temos uma contradição!

Se $p = 2$ e a é par, então $a^\alpha = (2k)^\alpha \equiv 2^\alpha k^\alpha \equiv 0 \pmod{2^\alpha}$. Como $\alpha < 2^\alpha$, concluímos que x_n é constante para $n \geq \alpha$. Se $(a, 2^\alpha) = 1$, pelo Teorema de *Euler* 1.2.6 temos a congruência $a^{2^{\alpha-1}} - 1 \equiv 0 \pmod{2^\alpha}$. Logo,

$$a^{2^{\alpha-1}} - 1 = (a-1)(1 + a + \cdots + a^{2^{\alpha-1}-1}) \equiv 0 \pmod{2^\alpha}.$$

Se 2^α não dividir $a - 1$, então 2^2 também não divide $a - 1$ ($\alpha \geq 2$). Por um lado, temos $(a, 2^\alpha) = 1$ e por outro lado, temos $a \not\equiv 1 \pmod{4}$. Portanto, se $a \equiv 3 \pmod{4}$, então $(1 + a + \cdots + a^{2^{\alpha-1}-1}) \equiv \frac{a^{2^{\alpha-1}}-1}{a-1} \equiv 0 \pmod{2^\alpha}$.

Concluímos que quando $\lambda = p^\alpha$ é necessário escolher $a = 1 + qp^e$, em que $p^e > 2$, q não é múltiplo de p e $e < \alpha$.

$\Leftarrow$) Seja p primo e $p^e > 2$. Aplicando várias vezes o Lema 2.2.6 temos

$$a^{p^g} \equiv 1 \pmod{p^{e+g}}, \quad a^{p^g} \not\equiv 1 \pmod{p^{e+g+1}},$$

para todo o $g \geq 0$. Como $p^e \mid p^{e+g}$ temos

$$\frac{a^{p^g}-1}{a-1} \equiv 0 \pmod{p^g}, \quad \text{e} \tag{2.8}$$

$$\frac{a^{p^g}-1}{a-1} \not\equiv 0 \pmod{p^{g+1}}. \tag{2.9}$$

Em particular, $\frac{a^{p^\alpha}-1}{a-1} \equiv 0 \pmod{p^\alpha}$. Como a congruência de parâmetros $(0, a, 1, p^\alpha)$ é determinada pela recorrência $x_n \equiv \frac{a^n-1}{a-1} \pmod{p^\alpha}$, se λ é o período temos $x_n = 0$ se, e só se, n é múltiplo de λ. Concluímos que p^α é múltiplo de λ e isto acontece somente se $\lambda = p^g$, para algum g.

Para concluir a prova observemos que $g = \alpha$ pois as congruências 2.8 e 2.9 implicam que $\lambda = p^\alpha$. Basta notar que, se $\lambda = p^{\alpha-1}$ a congruência 2.9 diz-nos que $\frac{a^{p^{\alpha-1}}-1}{a-1} \not\equiv 0 \pmod{p^\alpha}$. Uma contradição! □

Concluímos a prova do Teorema 2.2.5. □

Em [18], são estudados com detalhe vários problemas relativos à escolha dos parâmetros a, b e m da função GCL para diferentes tipos de aplicações.

2.2.4 Geradores pseudoaleatórios criptograficamente seguros

Podemos usar uma *função de um só sentido* f para gerar números ou bits pseudoaleatórios, 1.1. Primeiro seleccionamos uma semente aleatória x_0 e seguidamente calculamos $x_i = f(x_{i-1})$.

Um gerador de bits pseudoaleatórios diz-se criptograficamente seguro se passa o teste do bit seguinte.

Definição 2.2.3 (Teste do bit seguinte) *Diz-se que um gerador de bits pseudoaleatórios passa o teste do* bit seguinte *se, dados k bits de uma sequência s, não existe nenhum algoritmo polinomial que consiga prever o* $(k+1)$*-ésimo bit de s com uma probabilidade maior que* $\frac{1}{2}$.

O primeiro GNPA criptograficamente seguro foi proposto por *Shamir* [39] e é baseado na dificuldade do problema da inversão da função RSA. Com este gerador obtemos uma sequência de números no lugar de uma sequência de bits aleatórios e provou-se que um adversário, dada uma sequência de números, não consegue prever o número seguinte. Isto não chega para garantir que, na cifra de *Vernam*, cada bit da mensagem está bem protegido.

Blum e *Micali* [4] propuseram o primeiro gerador de bits pseudoaleatórios criptograficamente seguro baseado em *funções de um só sentido.* Seja D um conjunto finito e a permutação $f : D \rightarrow D$, uma *função de um só sentido.* Seja B uma função de D em $\{0,1\}$ tal que $B(y)$ é fácil de calcular quando conhecido $x = f^{-1}(y)$; e $B(y)$ é difícil de calcular conhecendo apenas y.

Dada uma semente $x_0 \in D$, podemos criar uma sequência $x_0 x_1 \ldots x_k$, de comprimento k, usando a recorrência $x_{i+1} = f(x_i)$. Para produzir a sequência binária $b_0 b_1 \ldots b_{k-1}$ definimos $b_i = B(x_{k-i})$. *Blum* e *Micali* mostraram que este gerador passa o test do bit seguinte.

Suponhamos que este gerador não passa o teste do bit seguinte. Vamos chegar a uma contradição calculando $B(y)$ conhecendo apenas y. Como f é uma permutação existe um elemento x_0 tal que $y = x_{k-i-1}$, em que x_i é a sequência gerada a partir da semente x_0. Aplicando f, podemos calcular $x_{k-i} \ldots x_k$ e com a função B determinamos $b_0 b_1 \ldots b_i$. Por um lado, não conhecemos $f^{-1}(y)$, pois f é uma *função de um só sentido.* Por outro lado,

como este gerador não passa o teste do bit seguinte, existe um algoritmo polinomial que prevê o bit b_{i+1}, logo $b_{i+1} = B(x_{k-i-1}) = B(y)$. Uma contradição pois $B(y)$ é difícil de calcular sem conhecer $f^{-1}(y)$.

Yao [44] provou que GNPA assim construídos são *perfeitos* no sentido que, nenhum algoritmo probabilístico pode dizer, em tempo polinomial, se a sequência produzida de comprimento k é aleatoriamente seleccionada de $\{0,1\}^k$. Por outras palavras, um gerador que passe o teste do bit seguinte é perfeito no sentido que passa todos os testes estatísticos polinomiais.

Assumindo que certos problemas da Teoria de Números são intratáveis, podemos construir geradores criptograficamente seguros que produzem sequências imprevisíveis em tempo polinomial.

Vamos apresentar um exemplo de um gerador de bits pseudoaleatórios criptograficamente seguro baseado na *função de um só sentido* RSA.

Exemplo 2.2.4 (GNPA RSA) *Definimos um algoritmo para obter uma sequência de bits pseudoaleatórios* criptograficamente seguros $z_k \ldots z_2 z_1$ de comprimento k.

(1) Geramos uma chave pública (n, e) do sistema de cifragem RSA.

(2) Escolhemos um inteiro x_0 (a semente) entre 1 e $n-1$.

(3) Para i de 1 até k faça:

   (a) $x_i \leftarrow x_{i-1}^e \pmod{n}$.

   (b) $z_i \leftarrow x_i \pmod{2}$.

(4) A sequência produzida é: $z_k \ldots z_2 z_1$.

Este gerador tem o inconveniente de ser lento por causa de operações de exponenciação modulares. Mas se escolhermos $e = 3$, apenas é necessário efectuar um quadrado e uma multiplicação modular. *Micali* e *Schnorr* [25] propuseram um gerador que melhora a eficiência do gerador, alterando o passo (3b) extraindo j bits de x_i, onde $j = c \log\log n$ e c é uma constante.

No pacote de criptografia BSAFE^{TM} (disponível em RSA Data Security, Inc), são usados GNPA em aplicações de criptografia, denominados de MD5Random, SHA1Random,

que usam sementes com 128 bits e 160 bits de tamanho, respectivamente. Note-se que estes GNPA usam funções unidireccionais criptograficamente seguras MD5 e SHA-1.

No *PGP v7.0.3*, o algoritmo X9.17-cast5 é usado para gerar números pseudoaleatórios, o algoritmo é descrito em [25].

2.3 Método de *Monte Carlo*

O primeiro matemático a mencionar este método com este nome foi Stanslaw Ulam. Pensa-se que o nome teve origem num casino do principado de Mónaco devido a um gerador (simples) de números aleatórios – a roleta.

O método de *Monte Carlo* tem vindo a ser desenvolvido desde 1944 e possibilita aproximar soluções de vários problemas de matemática, de resolução analítica muito complexa, recorrendo a experiências de amostras estatísticas com o auxílio do computador. Consiste em estimar o valor de n pontos de uma sequência de números aleatórios num determinado espaço de dimensão m para produzir uma solução aproximada. Dependendo do tipo de estudo, existem várias variantes de este método. Contudo, o método de *Monte Carlo* tende a ser mais lento do que qualquer outro método tradicional, *caso exista algum método determinístico.*

Podemos encontrar aplicações do método de *Monte Carlo* em quase todos ramos da ciência, tais como Economia, Física Nuclear e Química. Mas a aplicação de maior importância consistiu na resolução do problema da difusão de neutrões, parte dos cálculos que foram necessários para a construção da bomba atómica.

Uma das principais aplicações de este método é a *Integração Monte Carlo*, secção 2.3.3. Outros exemplos da aplicação do método de *Monte Carlo* são os testes probabilísticos de primalidade abordados na secção 3.2. No entanto, anteriormente existiram várias aplicações isoladas de este método sem grande desenvolvimento. Por exemplo, na segunda metade do Século XIX, foram realizadas várias experiências do problema da *agulha de Buffon* inferindo o valor de π, secção 2.3.2.

Um passo essencial para a aplicação do método de *Monte Carlo* consiste em gerar amostras de números aleatórios. Aqui, a noção aleatório é frequentemente substituída pela noção pseudoaleatório, secção 2.2.2, pois, na prática, pode não ser possível gerar

números perfeitamente aleatórios por intermédio da realização de uma experiência sem influências. Certas sequências pseudoaleatórias são mais adequadas para uso no *método de Monte Carlo*, tais como as que são geradas pela função congruência linear, secção 2.2.3.

2.3.1 Cálculo de π

A área de uma região do plano pode ser aproximada ao se gerar pontos aleatórios e ao se calcular a fracção dos pontos que "caem" na região. Esta é uma das maneiras mais simples de estimar o valor de π, recorrendo à proporção entre a área do círculo e a área do quadrado circunscrito ao círculo.

$$\frac{A_{\odot}}{A_{\square}} = \frac{\pi R^2}{(2R)^2} = \frac{\pi}{4} \tag{2.10}$$

Por analogia, podemos imaginar o jogo das setas: lançamos aleatoriamente várias setas no "tabuleiro de setas" e calculamos a fracção das que acertaram no círculo!

Definição 2.3.1 *Considere o círculo C de centro (r, r) e raio r. Seja Q o quadrado $[(0,0),(2r,0),(2r,2r),(0,2r)]$ circunscrito ao círculo C. Uma estimativa de π é dada pelo seguinte algoritmo:*

- *Geramos um par de números aleatórios (x, y) obtidos como amostras independentes de duas distribuições uniformes de probabilidade no conjunto $[0, 2r]$;*
- *Incrementamos o número total de experiências N_t;*
- *Incrementamos a variável N_d caso o ponto (x, y) esteja dentro do círculo;*

Ao fim de N "lançamentos", calculamos a proporção das áreas, obtendo:

$$\frac{A_{\odot}}{A_{\square}} = \frac{\pi}{4} \approx \frac{N_d}{N_t}$$

A estimativa de π torna-se tanto mais exacta quanto maior fôr a quantidade N de experimentos, e quanto mais "aleatórios" forem os números x e y.

Mas como converge esta estimativa π quando aumentamos o número de experiências?

Consideremos a experiência 2.3.1 com N "lançamentos" e seja X_i a variável aleatória que toma o valor 1 se (x, y) está dentro do círculo e toma o valor 0 caso contrário. A

proporção de sucessos é a média dos N lançamentos

$$\overline{X} = \frac{1}{N}\sum_{j=1}^{N} x_i$$

Se repetirmos novamente a experiência 2.3.1 com mais N lançamentos, então obtemos duas estimativas $\overline{X}_1$ e $\overline{X}_2$. Se o número de lançamentos N em cada experiência for grande, as estimativas tendem a aproximarem-se.

Imaginemos que realizámos M experiências de N lançamentos cada. A variação dos valores de $\overline{X}_1, \overline{X}_2, \ldots, \overline{X}_M$ pode ser caracterizada por uma estimativa do erro, o *desvio padrão*,

$$\sigma = \sqrt{\mathrm{Var}(\overline{X}_1, \overline{X}_2, \ldots, \overline{X}_M)} = \sqrt{\frac{1}{M-1}\sum_{j=1}^{M}(\overline{X}_j - \overline{X})^2},$$

em que $\overline{X}$ é a média das M experiências.

Podemos agora formular a questão: "como é que o *desvio padrão* dos resultados obtidos em várias experiências depende do número de lançamentos, N, em cada experiência?"

Pelo Teorema do Limite Central, podemos ver facilmente que

$$\mathrm{Var}(\overline{X}) \simeq \frac{v}{N},$$

em que v é a variância da distribuição de probabilidade, independente do número de lançamentos.

Isto prova que o erro experimental do resultado de uma experiência com N lançamentos decresce $\frac{1}{\sqrt{N}}$ quando o valor de N cresce. Para aumentar a precisão de uma estimativa de π em um dígito decimal, usando o método de *Monte Carlo*, devemos aumentar o número de lançamentos de pontos aleatórios independentes (x, y) por um factor 100.

2.3.2 A agulha de Buffon

Uma outra forma de calcular uma estimativa de π é o método da *agulha de Buffon*. O seu autor é *George Louis Leclerc* (1707-1788), conde de *Buffon*, que descreve o seguinte problema no seu *"Essai d'Arithemetique Morale"*.

Definição 2.3.2 *Sejam L e L' duas rectas paralelas no plano à distância d uma da outra (ver figura 2.1). Seja r um segmento de recta ("agulha") de comprimento $l < d$ colocada aleatoriamente no plano. Qual a probabilidade da agulha tocar uma das linhas paralelas?*

Consideremos o centro C da agulha sendo o ponto médio da agulha, x a distância do centro C à recta L' e φ o ângulo formado pela agulha e a recta perpendicular a L que passa por C. A posição da agulha pode ser determinada pelo par de coordenada (x, φ) onde $-\frac{\pi}{2} \leq \varphi \leq \frac{\pi}{2}$ e $0 \leq x \leq d$.

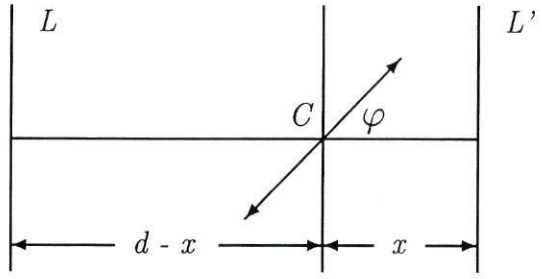


Figura 2.1: *Agulha de Buffon.*

Podemos ver facilmente que a agulha não toca nas duas rectas L e L' se e só se

$$-\frac{\pi}{2} \leq \varphi \leq \frac{\pi}{2} \quad \text{e} \quad \frac{l}{2}\cos(\varphi) \leq x \leq d - \frac{l}{2}\cos(\varphi). \tag{2.11}$$

Seja Ω o conjunto dos pontos (x, φ) que satisfazem 2.11. A probabilidade de a agulha não tocar nenhuma das rectas paralelas é:

$$\frac{\text{Área}(\Omega)}{d \cdot \pi} = \frac{\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \int_{\frac{l}{2}\cdot\cos(\varphi)}^{d-\frac{l}{2}\cdot\cos(\varphi)} 1 \, dx d\varphi}{d \cdot \pi} = \frac{d \cdot \pi - 2 \cdot l}{d \cdot \pi} = 1 - \frac{2 \cdot l}{d \cdot \pi} \tag{2.12}$$

Consequentemente, a probabilidade de a agulha tocar numa das rectas L ou L' é $\frac{2 \cdot l}{d \cdot \pi}$. Se escolhermos $d = 2$ e $l = 1$, então a probabilidade de a agulha tocar uma das rectas L ou L' é $\frac{1}{\pi}$.

Consideremos a experiência que consiste em lançar, de forma aleatória, n agulhas contra um tabuleiro limitado por duas rectas paralelas L e L'. Seja X_i a variável aleatória que assume valor 1 se a i-ésima agulha intersecta um das rectas L ou L', e assume o valor 0 caso contrário.

Pela Lei dos grandes números de *Bernoulli* [21], temos que para todo $\varepsilon > 0$

$$P\left[\left|\frac{X_1 + X_2 + \cdots + X_n}{n} - \frac{1}{\pi}\right| \geq \varepsilon\right] \leq \frac{1}{4n\varepsilon^2}. \tag{2.13}$$

Isto prova que $\pi \approx \frac{n}{X_1+X_2+\cdots+X_n}$ com uma probabilidade alta. A desigualdade 2.13 permite concluir que, para aumentar a precisão de uma estimativa de π em um dígito decimal, usando o método de *Monte Carlo*, devemos aumentar o número de agulhas a lançar de forma aleatória por um factor 100, tal como no exemplo da secção 2.3.1.

2.3.3 Integração de Monte Carlo

A integração numérica pode ser muito difícil, dependendo do domínio, de condições especiais ou de numerosas variáveis. O método de *Monte Carlo* ganha vantagem na resolução de problemas em que não se conhece nenhum método determinístico, ou em que métodos determinísticos não são praticáveis.

Podemos aplicar o método de *Monte Carlo* para o calcular um integral definido

$$I = \int_a^b f(x)dx, \tag{2.14}$$

em que $f(x)$ é um função positiva e limitada.

Consideremos o rectângulo $[(a,0),(b,0),(b,f_m),(a,f_m)]$, em que f_m é o valor máximo que a função toma no intervalo $[a,b]$. O integral de $f(x)$ pode ser entendido como uma fracção do rectângulo, ou seja, a área abaixo da curva $f(x)$ contida no rectângulo. O método de integração em si consiste em gerar uma grande quantidade de pares de números aleatórios (pseudoaleatórios) (x,y) no rectângulo e contar os pontos que estão sob a curva $f(x)$ para obter a área aproximada

$$I \approx \frac{N_s}{N_t}V,$$

em que N_t é o número total de pontos gerados, N_s é o número de pontos sob a curva $f(x)$ e V a área do rectângulo que limita $f(x)$. Para verificar se um ponto está sob a curva basta testar $y \leq f(x)$.

Este método é *muito ineficiente* pois são necessárias grandes quantidades de pontos aleatórios para convergir para a solução.

No entanto, existem maneiras mais fáceis e menos dispendiosas de usar números aleatórios para calcular integrais.

Notemos que

$$I = \int_a^b f(x)g(x)dx = \frac{1}{V}\int_a^b f(x)g(x)Vdx, \tag{2.15}$$

em que

$$g(x) = \begin{cases} 1 & \text{se } x \text{ pertence ao domínio,} \\ 0 & \text{caso contrário.} \end{cases}$$

e V é o *volume* do domínio. A equação 2.15 pode ser interpretada como o valor esperado da função $h(x) = f(x)g(x)V$, para a variável aleatória x de uma distribuição de probabilidade

uniforme no domínio. Logo

$$I \approx \frac{1}{n}\sum_{i=1}^{n} h(x_i) = \frac{V}{n}\sum_{i=1}^{n} f(x_i) \tag{2.16}$$

Uma estimativa baseada em 2.16 é muito mais rápida do que em 2.14.

Para aplicações como a integração de *Monte Carlo*, os números pseudoaleatórios podem ter propriedades excessivamente *aleatórias*. Pode acontecer que, ao "lançar" números pseudoaleatórios no domínio, determinadas regiões são representadas em excesso com um amontoado de pontos, enquanto que outras têm buracos, ou seja, não têm representantes. Isto pode mesmo dar origem a uma estimativa errada!

Podemos melhorar a nossa estimativa se alterarmos o nosso gerador de números aleatórios de modo a cobrir (quase) uniformemente todo o domínio. Aos números obtidos por este gerador chamamos de números *quasi–aleatórios*.

Exemplo 2.3.1 O valor exacto de

$$I = \int_0^1 \int_0^1 (x^2 + y^2) dxdy$$

é $\frac{2}{3}$. Ao aplicarmos 2.16 efectuando o "*lançamento*" de 5000 pontos usando números *pseudoaleatórios*, obtivemos 0.6634. No mesmo teste usámos números *quasi–aleatórios* e obtivemos é 0.6664.

Capítulo 3

Testes de primalidade

Num sistema de cifragem assimétrico é importante gerar eficientemente os parâmetros da chave pública e privada. Por exemplo, no sistema de cifragem RSA, geramos dois inteiros primos p e q para obter o módulo $n = pq$. Neste caso, os primos p e q devem ter "tamanho" suficiente para que a factorização de n seja extremamente difícil. Os primos devem ser "aleatórios" no sentido em que a probabilidade de escolher um primo em particular é suficientemente pequena, para prevenir que se tire vantagens de uma pesquisa optimizada baseada nessa probabilidade.

3.1 Geração de primos grandes

O método mais natural consiste em gerar um número aleatório n de um determinado tamanho e verificar se é primo. Podemos verificar se n é primo efectuando divisões sucessivas por todos os primos até $\sqrt{n}$.

De facto, suponha que n é composto, que p é o menor divisor primo de n e que $p > \sqrt{n}$. Então $m = \frac{n}{p} \in \mathbb{Z}$ e $m < \frac{n}{\sqrt{n}} = \sqrt{n}$. Se m é composto, como a decomposição em factores primos é única, existe um primo q que divide m e $q < m < \sqrt{n}$. O absurdo resulta de considerarmos que $p > \sqrt{n}$ é o menor primo divisor.

Efectuar divisões sucessivas por todos os primos até $\sqrt{n}$ não resulta na prática. Imaginemos que n é um número com 100 dígitos e que um computador realiza cerca de 10^{12} divisões por segundo. O número total de divisões que teríamos que efectuar seria da ordem de $\frac{\sqrt{10^{100}}}{2}$, ou seja, $\frac{10^{50}}{2}$. É fácil calcular, nestas condições, quantos anos são necessários

para realizar todas estas divisões, nada mais do que:

$$\frac{10^{50}}{10^{12} \times 3\,600 \times 24 \times 365} = 3\,170\,979\,198\,376\,458\,650\,431\,253\,170\,979,2!$$

Seriam precisos cerca de 10^{31} anos para efectuar todas as divisões, o que é algo desanimador! O facto é que o algoritmo envolvendo divisões pode provar que um número n é primo, e ainda fornece a sua decomposição em factores primos. Mas nós sabemos que factorizar é muito difícil!

Talvez se possa ganhar tempo se não determinarmos a factorização prima de n. Realmente o nosso objectivo é, literalmente, determinar se n é primo ou composto, sem se conhecer os seus factores primos. De facto, veremos mais adiante que há algoritmos, rápidos, que determinam com um grau de certeza elevado a primalidade de n.

Mediante a função `gen_prime(nbits, secret, random_level)`, vamos analisar com algum detalhe o processo utilizado pelo *GnuPG v1.06*, para gerar um número primo p, de tamanho `nbits`.

(i) Gera um número aleatório p de tamanho `nbits` como sendo o candidato a primo;

(ii) Assegura que o primeiro e último bit têm valor 1, para garantir que p é ímpar e que p tem realmente tamanho `nbits`, respectivamente;

(iii) Calcula a tabela `mods[i]` $\equiv p \pmod{p_i}$, para primos p_i menores que 5000, contidos na tabela `small_prime_numbers[i]`;

(iv) Calcula o primeiro número $p + j$, para $j = 2, 4, \ldots, 20\,000$, que passa os seguintes testes de primalidade:

  (a) Verifica se $p + j$ é divisível por algum primo pequeno contido na tabela `mods[i]`, calculando $p + j \equiv$ `mods[i]` $+ j \pmod{p_i}$;

  (b) Verifica $p + j$ é um *pseudoprimo de Fermat* para a base $b = 2$;

  (c) Através da função `isprime(p+j,5,&count2)`, aplica o *teste probabilístico 3.2.4 de Miller–Rabin* a $p + j$ para um total de $t = 5$ bases. É testada a base $b = 2$ e quatro bases aleatórias $b \neq \pm 1$;

(v) Se nenhum dos inteiros $p+i$, para $i = 0, 2, 4, \ldots, 20\,000$, passou os testes, então voltamos ao passo (i).

Este processo só pára quando um número $p + j$ passa os 3 testes (a), (b) e (c).

O *GnuPG v1.06* recruta os seus candidatos de forma "mista", isto é, gera um número aleatório n e restringe os candidatos a primos ao conjunto finito $\mathcal{R}(n) = \{n, n+2, \ldots, n+20\,000\}$. Se este conjunto esgotar, então o *GnuPG v1.06* gera outro número aleatório para substituir o conjunto $\mathcal{R}(n)$.

No passo (iv), o teste de primalidade pode ser uma *prova* de que o candidato n é primo ou um teste que estabelece um resultado mais fraco: que n é *provavelmente* um número primo. Na seguinte secção vamos estudar com detalhe os dois testes probabilísticos usados pelo *GnuPG v1.06* e um terceiro teste usado no *PGP v7.0.3*.

3.2 Testes probabilísticos de primalidade

Os testes de primalidade são uma aplicação do método de *Monte Carlo* 2.3, são métodos que testam aleatoriamente números inteiros para fornecer informação acerca da sua primalidade. Considere, para cada inteiro ímpar, o conjunto $\mathcal{W}(n) \subset \mathbb{Z}_n$ que verifica as seguintes propriedades:

(i) Dado $a \in \mathbb{Z}_n$, então é rápido verificar se $a \in \mathcal{W}(n)$;

(ii) Se n é primo, então $\mathcal{W}(n) = \emptyset$;

(iii) Se n é composto, então $\#\mathcal{W}(n) \geq \frac{n}{2}$.

Definição 3.2.1 *Se n é um número composto, os elementos de $\mathcal{W}(n)$ são chamados as "testemunhas" de que n é composto, e os elementos do complementar $\mathcal{L}(n) = \mathbb{Z}_n - \mathcal{W}(n)$ são chamados os "mentirosos".*

Os testes probabilísticos de primalidade usam as propriedades do conjunto $\mathcal{W}(n)$ para obter informação acerca da primalidade de n da seguinte maneira.

Se n é um número ímpar aleatório, candidato a primo, geramos um número aleatório $a \in \mathbb{Z}_n$ e verificamos se $a \in \mathcal{W}(n)$, ou seja, se a é uma *testemunha* de que n é composto. Se $a \in \mathcal{W}(n)$, diz-se que n *falhou o teste de primalidade para a base* a e, neste caso, temos a certeza de que n é composto. Por outro lado, se $a \notin \mathcal{W}(n)$, diz-se que n *passou* o teste

primalidade para a base a. Neste caso, nada se pode concluir com toda a certeza acerca da primalidade de n: o teste responde "primo" com um erro máximo de $\frac{1}{2}$.

Ao realizarmos um teste probabilístico de primalidade t vezes, independentemente, sobre um número composto n, a probabilidade de este teste responder "primo" é, no máximo, $\left(\frac{1}{2}\right)^t$. Estes testes são usualmente denominados por *testes de composição*, pois determinam com certeza se um número é *composto*, ou se um número é *primo*, com uma baixa probabilidade de errar. A um número inteiro n que se acredite ser primo, baseado num teste de probabilístico de primalidade, chamamos um número *provavelmente primo*.

3.2.1 Teste de primalidade de Fermat

Seja p um número primo e a um número inteiro tal que $(a,p) = 1$. Pelo Teorema de Fermat sabemos que $a^{p-1} \equiv 1 \pmod{p}$. Leibniz acreditava que se podia estabelecer o recíproco para $a = 2$, ou seja, se $2^{p-1} \equiv 1 \pmod{p}$, então n seria primo. Seria óptimo se esta relação fosse verdadeira, pois para verificar a primalidade de um número n é primo bastaria calcular $2^{n-1} \pmod{p}$ e conferir se é igual a 1. De facto, esta relação é verdadeira para $n < 341$, mas é falsa para $n = 341$, porque $341 \mid 2^{340} - 1$, embora 341 seja composto $(341 = 11 \times 31)$.

Pode-se ver rapidamente que $341 \mid 2^{340} - 1$, tendo em conta que $2^5 \equiv 1 \pmod{31}$ e $2^{10} \equiv 1 \pmod{11}$.

Claro que se n é um inteiro ímpar, se se mostrar que existe um inteiro a tal que $1 < a \leq n-1$, $(a,n) = 1$ e $a^{n-1} \not\equiv 1 \pmod{n}$, então podemos afirmar que n é composto pelo Teorema de Fermat.

Podemos aplicar o Teorema de Fermat como um teste probabilístico de primalidade. Se a é tal que $a^{n-1} \not\equiv 1 \pmod{n}$, então n *falha* o teste probabilístico de Fermat para a base a $(a \in \mathcal{W}(n))$, ou seja, o teste responde que n é composto. Se $a^{n-1} \equiv 1 \pmod{n}$, n *passa* o teste probabilístico de Fermat $(a \notin \mathcal{W}(n))$, neste caso o teste responde que n é provavelmente primo.

Definição 3.2.2 *Seja n um número inteiro ímpar composto e seja a tal que $1 < a \leq n-1$. Diz-se que n é um* pseudoprimo de Fermat *para a base a, se $a^{n-1} \equiv 1 \pmod{n}$. Chamamos ainda ao inteiro "a" um* mentiroso de Fermat *(para a prova de que n é composto).*

Exemplo 3.2.1 O número $n = 341 (= 11 \times 31)$ é um pseudoprimo de Fermat para a base 2 pois $2^{340} \equiv 1 \pmod{341}$.

O seguinte Lema dá informação acerca da quantidade de bases a que são testemunhas de Fermat, caso exista uma.

Lema 3.2.1 *Se existe um número inteiro $a \in \mathbb{Z}_n^*$, (isto é, $(a, n) = 1$) tal que $a^{n-1} \not\equiv 1 \pmod{n}$, então o mesmo acontece para pelo menos metade dos elementos de $\mathbb{Z}_n^*$.*

Demonstração. Sejam $a_1, a_2, \ldots, a_k$ os elementos de $\mathbb{Z}_n^*$ que satisfazem $a_i^{n-1} \equiv 1 \pmod{n}$. Vejamos primeiro que os elementos $aa_1, aa_2, \ldots, aa_k$ são todos distintos. Suponha que $aa_i = aa_j$ e que a_i e a_j não pertencem a mesma classe residual, então $aa_i \equiv aa_j \pmod{n}$ se, e só se, $a(a_i - a_j) \equiv 0 \pmod{n}$. A congruência equivale a $n \mid a(a_i - a_j)$ mas, como $(a, n) = 1$, então $n \mid (a_i - a_j)$, o que equivale a $a_i \equiv a_j \pmod{n}$, uma contradição.

Como $(aa_i)^{n-1} \equiv a^{n-1}a_i^{n-1} \not\equiv 1 \pmod{n}$, então pelo menos metade dos elementos $a \in \mathbb{Z}_n^*$ satisfazem $a^{n-1} \not\equiv 1 \pmod{n}$. □

Em geral, os pseudoprimos para uma base a fixa são raros[1]. Por exemplo, há 455 052 512 primos menores que 10^{10}, enquanto que temos somente 14 884 pseudoprimos para a base 2. Por outro lado, o facto de n ser composto não garante que exista uma base a tal que $a^{n-1} \not\equiv 1 \pmod{n}$, ou seja, há números compostos para os quais falha o teste probabilístico de primalidade de Fermat para a base a. Pode até acontecer que um número composto n seja pseudoprimo para todas as bases a, $(1 < a \leq n - 1)$ com $(a, n) = 1$.

Definição 3.2.3 *Seja n um número inteiro composto. Diz-se que n é um número de* Carmichael *se $a^{n-1} \equiv 1 \pmod{n}$, $\forall a \in \mathbb{Z}_n$ tal que $(a, n) = 1$ ou, por outras palavras, se n é um pseudoprimo para todas as bases a que são primos com n.*

Propriedade 3.2.1 *Um número inteiro composto n é um número de* Carmichael *se, e só se, n é livre de quadrados e $(p - 1) \mid (n - 1)$ para todo o primo p que divide n.*

Vejamos um exemplo tomando o mais pequeno número de *Carmichael*, $n = 561 = 3 \times 11 \times 17$. Como 2, 10 e 16 dividem 560, para todo o número inteiro b tal que $(b, 561) = 1$,

[1] A conjectura de que os pseudoprimos de Fermat para a base 2 são raros quando comparados com os números primos foi provada em 1950, por Erdös [13]. O número de pseudoprimos de Fermat para a base 2 menores que n é inferior a $xe^{-\frac{1}{3}\log(x)^{\frac{1}{4}}}$.

temos que:

$$\begin{cases} b^{560} \equiv (b^2)^{280} \equiv 1 \pmod 3 \\ b^{560} \equiv (b^{10})^{56} \equiv 1 \pmod{11} \\ b^{560} \equiv (b^{16})^{35} \equiv 1 \pmod{17} \end{cases}.$$

Pelo Teorema Chinês dos Restos temos que $b^{560} \equiv 1 \pmod{561}$, ou seja, 561 é pseudo-primo para todas as bases b.

Se n é um número de *Carmichael*, então as únicas bases que podem ser *testemunhas de Fermat* são os números a tais que $(a, n) > 1$. Se os factores primos do número de *Carmichael* n forem grandes, então o teste de primalidade de Fermat responderá com uma grande probabilidade "n é primo" mesmo que o número de testes realizados seja grande, ou seja, o teste de Fermat não dá a certeza nem fornece uma prova concreta de que n é primo.

Seja $\mathcal{C}(n)$ a quantidade de *números de Carmichael* em $[2, n]$.

Numericamente, podemos observar que $\mathcal{C}(n)$ é muito pequeno quando comparado com o número de primos no mesmo intervalo. Logo, ao realizar t vezes o teste de Fermat, este decide correctamente a primalidade para a *maioria* dos candidatos a primos com uma probabilidade de $1 - \left(\frac{1}{2}\right)^t$. Mas seria bem mais interessante se um teste respondesse correctamente com qualquer candidato, quando testado com bases diferentes. No entanto, não vamos desistir do teste de Fermat! Vamos procurar uma forma de reconhecer os números compostos que são números de *Carmichael*.

Apesar de o teste de Fermat não ser um teste muito forte, ele continua a ser utilizado como uma primeira abordagem à primalidade de número inteiro n, por exemplo no *GnuPG v1.06*, devido à sua rapidez e facilidade de implementação computacional.

3.2.2 Teste de primalidade de Miller–Rabin

Já sabemos que existem muitos números que se mascaram como primos quando se aplica o teste de primalidade de Fermat. O que vamos fazer é modificar um pouco o teste de Fermat de modo que este tente reconhecer os números de Carmichael.

Lema 3.2.2 *Seja $p > 2$ um inteiro primo. Então 1 só tem duas raízes quadradas inteiras módulo p: são elas ± 1.*

Demonstração. Visto que p é primo, existe uma raiz primitiva g de p, logo

$$\begin{aligned} x^2 \equiv 1 \pmod{p} &\Leftrightarrow 2\operatorname{ind}_g x \equiv \operatorname{ind}_g 1 \pmod{\phi(p)} \\ &\Leftrightarrow 2\operatorname{ind}_g x \equiv 0 \pmod{\phi(p)} \end{aligned}$$

que só tem solução se $(2, \phi(p)) = 2 \mid 0$. A congruência tem $(2, p-1) = 2$ soluções: são elas $\operatorname{ind}_g x_0 = 0$ e $\operatorname{ind}_g x_1 = 0 + \frac{p-1}{2}$. Portanto, $x_0 = g^0 = 1$ e, pelo critério de Euler, temos que $x_1 = g^{\frac{p-1}{2}} \equiv -1$, pois g tem índice 1 que é ímpar. □

Podemos utilizar este facto com o número $n = 561$ e a base $b = 5$, $(5, 561) = 1$. Apesar de $5^{560} \equiv 1 \pmod{561}$ temos que $5^{\frac{561-1}{2}} \equiv 5^{280} \equiv 67 \pmod{561}$. Concluímos que 1 tem 67 como raiz quadrada inteira módulo 561, que é diferente de ± 1. Ao testar $b^{\frac{p-1}{2}} \equiv \pm 1 \pmod{p}$, conseguimos reconhecer que o número de *Carmichael* 561 é composto.

Uma questão que se poderá colocar é se esta modificação reconhece todos os números de *Carmichael*. Infelizmente, não! Se considerarmos o número $n = 1729 = 7 \times 13 \times 19$, este satisfaz ambas as relações $b^{n-1} \equiv 1 \pmod{n}$ e $b^{\frac{n-1}{2}} \equiv 1 \pmod{n}$, para cada b primo com n. Até agora, a única modificação que fizemos foi averiguar se a raiz quadrada de $1 \equiv b^{n-1} \pmod{n}$ é igual a ± 1, ou seja, se $b^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n}$. Como neste caso $b^{\frac{n-1}{2}} \equiv 1 \pmod{n}$, podemos continuar a explorar o Lema 3.2.2 para verificar se $1 \equiv b^{\frac{n-1}{2}} \pmod{n}$ têm raiz quadrada igual a ± 1, repetindo o argumento até à exaustão. Se escrevermos $n-1$ na forma $2^s t$ com t ímpar, teremos no máximo s verificações a realizar. Por exemplo, se $n = 1729$ e $b = 5$, temos que $n - 1 = 2^6 \times 27$ e:

$5^{\frac{n-1}{2^j}}$	$5^{\frac{1729-1}{2}}$	$5^{\frac{1729-1}{2^2}}$	$5^{\frac{1729-1}{2^3}}$	$5^{\frac{1729-1}{2^4}}$	$5^{\frac{1729-1}{2^5}}$
(mod 1729)	1	1	1	1	1065

Mostramos que $n = 1729$ é composto pois 1 tem 1065 como raiz quadrada inteira módulo 1729, que é diferente de ± 1. Este processo é semelhante ao algoritmo de *Miller* que na prática, funciona na direcção oposta, como veremos.

Sejam n um número inteiro ímpar e b um inteiro primo com n (geralmente um primo pequeno). Sejam s e t dois inteiros tal que $n - 1 = 2^s t$, com t ímpar. Vamos escrever as b–sequências de n: calculamos $b^t \pmod{n}$ como primeiro termo da sequência, seguidamente elevamos o primeiro termo ao quadrado, sucessivamente, para obter a sequência de $s+1$ elementos de $\mathbb{Z}_n$, $b^t, b^{2t}, b^{2^2 t}, \ldots, b^{n-1} \pmod{n}$. Representando por $\star$ um número não congruente com ± 1 módulo n, a Tabela 3.1 apresenta as cinco possíveis b–sequências que

Tipo	b^m	b^{2m}	b^{4m}	$\cdots$	.	.	.	$\cdots$	b^{n-1}
Tipo I	$+1$	$+1$	$+1$	$\cdots$	$+1$	$+1$	$+1$	$\cdots$	$+1$
	$\star$	$\star$	$\star$	$\cdots$	$\star$	-1	$+1$	$\cdots$	$+1$
Tipo II	$\star$	$\star$	$\star$	$\cdots$	$\star$	$+1$	$+1$	$\cdots$	$+1$
	$\star$	$\star$	$\star$	$\cdots$	$\star$	$\star$	$\star$	$\cdots$	$\star$
	$\star$	$\star$	$\star$	$\cdots$	$\star$	$\star$	$\star$	$\cdots$	-1

Tabela 3.1: *Tipos de b–sequências de n.*

se podem formar, dividindo-se essas sequências em dois tipos: I e II. Uma b–sequência é do Tipo I se todos os termos da sequência são $+1$ ou se o primeiro termo que não é 1 for -1. Caso contrário, a b–sequência é do Tipo II.

Se n é um número primo, então o último termo da b–sequência tem que ser 1. De facto, pelo *Teorema de Fermat*, temos que $b^{n-1} \equiv 1 \pmod{n}$ e, como as únicas raízes quadradas de 1 em $\mathbb{Z}_n$ são ± 1, o termo anterior a um termo 1 só pode ser ± 1. Então o caminho que a b–sequência percorre tem que ser do Tipo I. Podemos finalmente enunciar o Teste de Primalidade de *Miller*.

Proposição 3.2.1 (Teste de Miller) *Sejam n um inteiro ímpar, s e t inteiros tais que $n-1 = 2^s t$, em que t é ímpar. Se b é um inteiro tal que $1 < b \leq n-1$, $b \nmid n$ e*

$$b^t \equiv 1 \pmod{n} \quad \textit{ou} \quad b^{2^j t} \equiv -1 \pmod{n}, \qquad \textit{para algum } j \textit{ entre } 0 \textit{ e } s-1,$$

então dizemos que n passa o Teste de Miller *($b \in \mathcal{L}(n)$), ou seja, as b–sequências são do Tipo I.*

Se $b \mid n$ ou n falha o Teste de Miller, *isto é, n é uma testemunha ($b \in \mathcal{W}(n)$), então n é composto, ou seja, as b–sequências são do Tipo II.*

O custo do Teste de *Miller* não é maior do que o custo do Teste de *Fermat*, ou seja, o custo de uma potência modular: $O(log^3 n)$. Se n passa o Teste de *Miller* para a base b, diz-se que n é um *pseudoprimo forte* para a base b.

Exemplo 3.2.2 O número composto $n = 1729$ ($1729 - 1 = 2^6 \cdot 27$) é um *pseudoprimo forte* para as bases 103 e 191, pois as b–sequências de 1729 são do Tipo I:

103–**sequência de** 1729	103^{27}	$103^{2\cdot 27}$	$103^{2^2\cdot 27}$	$103^{2^3\cdot 27}$	$103^{2^4\cdot 27}$	$103^{2^5\cdot 27}$	103^{1729-1}
	-1	1	1	1	1	1	1

191–sequência de 1729	191^{27}	$191^{2\cdot 27}$	$191^{2^2\cdot 27}$	$191^{2^3\cdot 27}$	$191^{2^4\cdot 27}$	$191^{2^5\cdot 27}$	191^{1729-1}
	1	1	1	1	1	1	1

Para uma base fixa $b > 0$, existem infinitos pseudoprimos fortes para a base b (ver [34]), mas felizmente há muito poucos inteiros que são pseudoprimos fortes para várias bases b. De facto, o Lema 3.2.3 provado em 1980 por *Monier* e *Rabin*, mostra que não há nenhum número composto n que seja pseudoprimo forte para todas as bases b, $(b, n) = 1$. Portanto, nada análogo aos números de *Carmichael* existe para o teste de *Miller*.

Lema 3.2.3 *Se n é um número composto ímpar, então n é pseudoprimo forte com relação a, no máximo, um quarto das bases a tais que $1 \leq a \leq n-1$ e $(a, n) = 1$.*

Demonstração. Sabemos que $2 \nmid n$, senão n seria ímpar.

Caso 1: (p não é livre de quadrados) Seja p um número inteiro primo maior que 2 tal que $p^2 \mid n$, ou seja, $n = p^2 h$. Queremos provar que se verifica $a^{n-1} \equiv 1 \pmod{n}$ para, no máximo, um quarto das bases entre 1 e $n-1$. Note-se que $a^{n-1} \equiv 1 \pmod{n}$ é equivalente a $a^{n-1} + nk = 1$, ou ainda $a^{n-1} + p^2 hk = 1$.

Se a verifica $a^{n-1} \equiv 1 \pmod{n}$, então $a^{n-1} \equiv 1 \pmod{p^2}$. Por um lado, para cada classe a módulo p^2, há exactamente h classes b módulo n tais que $b \equiv a \pmod{p^2}$:

$$b = a, a + p^2, a + 2p^2, \ldots, a + (h-1)p^2.$$

Por outro lado, se tivermos b^{n-1} com $(b, n) = 1$, então temos que para qualquer inteiro $k \geq 0$

$$(b + kp^2)^{n-1} = b^{n-1} + \sum_{i=0}^{n-1} b^{n-1-i} \cdot (kp^2)^i \equiv b^{n-1} \equiv 1 \pmod{p^2}$$

Uma vez no conjunto $\{a, a + p^2, a + 2p^2, \ldots, a + (h-1)p^2\}$ há exactamente h classes de congruência módulo n, obtemos

$$\begin{aligned}
\frac{\#\{1 \leq b \leq n-1 \mid b^{n-1} \equiv 1 \pmod{n}\}}{n-1} &\leq \frac{h\,\#\{1 \leq a \leq p^2-1 \mid a^{n-1} \equiv 1 \pmod{p^2}\}}{n-1} \\
&= \frac{h\,\#\{1 \leq a \leq p^2-1 \mid a^{n-1} \equiv 1 \pmod{p^2}\}}{p^2 h - 1} \\
&= \frac{\#\{1 \leq a \leq p^2-1 \mid a^{n-1} \equiv 1 \pmod{p^2}\}}{p^2 - \frac{1}{h}} \\
&\leq \frac{\#\{1 \leq a \leq p^2-1 \mid a^{n-1} \equiv 1 \pmod{p^2}\}}{p^2-1}.
\end{aligned}$$

Desta forma, a percentagem de inteiros a, entre 1 e $n-1$, que satisfazem $a^{n-1} \equiv 1 \pmod{n}$ é menor ou igual à percentagem de inteiros a, entre 1 e $p^2 - 1$, que satisfazem $a^{n-1} \equiv 1$

$(\text{mod } p^2)$. Vamos mostrar que a proporção de inteiros a, entre 1 e p^2-1, que satisfazem $a^{n-1} \equiv 1 \pmod{p^2}$ é menor ou igual a $0,25$.

Sabemos que existe uma raiz primitiva g para p^2. Seja $\{g, g^2, \ldots, g^{\phi(p^2)}\}$ um sistema reduzido de resíduos módulo p^2. Como $a^{n-1} \equiv 1 \pmod{n}$, então $a^{n-1} \equiv (g^i)^{n-1} \equiv g^{i(n-1)} \equiv 1 \pmod{p^2}$ se, e só se, $\phi(p^2) \mid i(n-1)$, ou se $p(p-1) \mid i(n-1)$. Como $p \mid n$, então $p \nmid (n-1)$, logo $p \mid i$. Visto que existem $p-1$ índices no sistema reduzido de resíduos $\{a^1, \ldots, a^p, \ldots, a^{2p}, \ldots, a^{(p-1)p}\}$ que são divisíveis por p, então existem no máximo $p-1$ elementos de $\mathbb{Z}_{p^2}^*$ que satisfazem $a^{n-1} \equiv 1 \pmod{p^2}$. A percentagem de inteiros entre 1 e p^2-1 que satisfazem $a^{n-1} \equiv 1 \pmod{p^2}$ é menor ou igual a:

$$\frac{p-1}{p^2-1} = \frac{p-1}{(p-1)(p+1)} = \frac{1}{p+1} \leq \frac{1}{4} \quad (\text{porque } p > 2).$$

Caso 2: (p é livre de quadrados) Seja $n = p_1 \cdot p_2 \cdots p_r$ $(r \geq 2)$, em que os inteiros p_i são primos distintos. Cada p_i pode-se escrever como $p_i - 1 = 2^{s_i} t_i$, onde t_i é um inteiro ímpar. Se n é um pseudoprimo forte para a base $a \in \mathbb{Z}_n^*$ e $n-1 = 2^s t$, com t ímpar, então ocorre uma das seguintes relações, pelo Teste de *Miller*:

$$\begin{aligned} a^t &\equiv 1 \pmod{n} \\ a^{2^j t} &\equiv -1 \pmod{n}, \quad \text{para algum } j \text{ entre } 0 \leq j \leq s-1. \end{aligned}$$

Por um lado, se $a^t \equiv 1 \pmod{n}$, o Teorema Chinês dos Restos, mostra que isto acontece se, e só se $a_i^t \equiv 1 \pmod{p_i}$, para $1 \leq i \leq r$ e $a_i \in \mathbb{Z}_{p_i}^*$. Como existe uma raiz primitiva g_i para p_i, se $\{g_i, g_i^2, \ldots, g_i^{\phi(p_i)}\}$ for um sistema reduzido de resíduos módulo p_i, existirá $j_i \in \{1, \ldots, \phi(p_i)\}$ tal que $a_i = g_i^{j_i}$. Então $a_i^t \equiv (g_i^{j_i})^t \equiv g_i^{j_i t} \equiv 1 \pmod{p_i}$, ou seja, $(p_i - 1) \mid j_i t$.

Suponhamos que $(t, p_i - 1) = \alpha_i$. Então $(p_i - 1) \mid j_i t$ equivale a $\frac{(p_i-1)}{\alpha_i} \mid j_i \frac{t}{\alpha_i}$, que ocorre exactamente para $j_i = \frac{(p_i-1)}{\alpha_i}, \frac{2(p_i-1)}{\alpha_i}, \ldots, \frac{\alpha_i(p_i-1)}{\alpha_i}$, pois $1 = \left(\frac{p_i-1}{\alpha_i}, \frac{t}{\alpha_i}\right)$. Deduzimos que $a_i^t \equiv 1 \pmod{p_i}$ tem exactamente $\alpha_i = (t, p_i - 1)$ soluções módulo p_i. Como $p_i - 1 = 2^{s_i} t_i$, temos $\alpha_i = (t, t_i)$. Logo, pelo Teorema Chinês dos Restos, o número de soluções da congruência $a^t \equiv 1 \pmod{n}$ é igual a $(t, t_1) \cdot (t, t_2) \cdots (t, t_r) \leq t_1 \cdot t_2 \cdots t_r$.

Por outro lado, se $a^{2^j t} \equiv -1 \pmod{n}$ com $0 \leq j \leq s-1$, então $a^{2^j t} \equiv -1 \pmod{p_i}$, para todo i tal que $1 \leq i \leq r$, também pelo Teorema Chinês dos Restos. Como vimos atrás, existe $k_i \in \{1, \ldots, \phi(p_i)\}$ tal que $a \equiv g_i^{k_i} \pmod{p_i}$ e $\alpha_i = (p_i - 1, t)$. Logo $a^{2^j t} \equiv (g_i^{k_i})^{2^j t} \equiv g_i^{2^j k_i t} \equiv -1 \pmod{p_i}$ tem solução se, e apenas se, $(p_i - 1) \mid 2^{j+1} k_i t$ e $(p_i - 1) \nmid 2^j k_i t$. Como $p_i - 1 = 2^{s_i} t_i$, então devemos ter $2^{s_i} t_i \mid 2^{j+1} k_i t$ e $2^{s_i} t_i \nmid 2^j k_i t$.

Se $s_i \leq j$, deduzimos que para qualquer inteiro k_i tem-se $2^{s_i}t_i \mid 2^j k_i t$, ou seja, a congruência $a^{2^j t} \equiv -1 \pmod{p_i}$ não tem soluções.

Se $s_i > j$, então devemos ter

$$2^{s_i}\frac{t_i}{\alpha_i} \mid 2^{j+1}k_i\frac{t}{\alpha_i} \quad \text{e} \quad 2^{s_i}\frac{t_i}{\alpha_i} \nmid 2^j k_i\frac{t}{\alpha_i}$$

com $\frac{t_i}{\alpha_i}$ e $\frac{t}{\alpha_i}$ primos entre si e $k_i \in \{1, \ldots, p_i - 1 = 2^{s_i}t_i\}$ ímpar. Concluímos que k_i deve ser um múltiplo ímpar de

$$2^{s_i-j-1}\frac{t_i}{\alpha_i} = 2^{s_i-j-1}(t, t_i).$$

Existem $2^{j+1}\alpha_i$ múltiplos de $2^{s_i-j-1}(t, t_i)$ neste intervalo, logo há $2^j\alpha_i$ múltiplos ímpares de $2^{s_i-j-1}(t, t_i)$ entre 1 e $2^{s_i}t_i$. Portanto, a congruência $a^{2^j t} \equiv -1 \pmod{p_i}$ tem $2^j(t, t_i)$ soluções.

Definindo $\overline{s} = \min\{s_i : 1 \leq i \leq r\}$, o número de soluções da congruência $a^{2^j t} \equiv -1 \pmod{n}$ é dado por,

$$\begin{cases} 2^j(t, t_1) \cdot 2^j(t, t_2) \cdots 2^j(t, t_r) \leq 2^{jr}t_1 t_2 \cdots t_r & \text{se } j < \overline{s} \\ 0 & \text{se } j \geq \overline{s} \end{cases}.$$

Logo, o número bases a, $1 \leq a \leq n-1$, para as quais n é pseudoprimo forte é menor ou igual a,

$$\underbrace{t_1 t_2 \cdots t_r}_{\text{Caso 1}} + \underbrace{2^{0\cdot r}t_1 t_2 \cdots t_r}_{j=0} + \cdots + \underbrace{2^{(\overline{s}-1)\cdot r}t_1 t_2 \cdots t_r}_{j=\overline{s}-1} + \underbrace{0}_{j=\overline{s}} + \cdots + \underbrace{0}_{j=s-1}.$$

Pondo $t_1 t_2 \cdots t_r$ em evidência, obtemos

$$t_1 t_2 \cdots t_r \left(1 + 1 + 2^r + 2^{2r} + \cdots + 2^{(\overline{s}-1)r}\right). \tag{3.1}$$

Podemos concluir que a proporção de inteiros a $(1 \leq a \leq n-1)$ para os quais n é pseudoprimo forte é menor que:

$$\begin{aligned}
\frac{t_1 t_2 \cdots t_r\left(1+1+2^r+2^{2r}+\cdots+2^{(\overline{s}-1)r}\right)}{2^{s_1}t_1 2^{s_2}t_2 \cdots 2^{s_r}t_r} &= \frac{1}{2^{(s_1+s_2+\cdots+s_r)}}\left(1 + \frac{2^{r(\overline{s}-1+1)}-1}{2^r-1}\right) \\
&\leq \frac{1}{2^{r\overline{s}}}\left(1 + \frac{2^{r\overline{s}}}{2^r-1} - \frac{1}{2^r-1}\right), && (\overline{s} \leq s_i,\ 1 \leq i \leq r) \\
&= \frac{1}{2^r-1} + \frac{2^r-1-1}{2^{r\overline{s}}(2^r-1)} \\
&\leq \frac{1}{2^r-1} + \frac{2^r-2}{2^r(2^r-1)}, && (\overline{s} \geq 1) \\
&= \frac{2^r+2^r-2}{2^r(2^r-1)} = \frac{2}{2^r} = \frac{1}{2^{r-1}} \leq \frac{1}{4} && (r \geq 3)
\end{aligned}$$

Isto prova que n é pseudoprimo forte para, no máximo, um quarto das bases a entre 1 e $n-1$ quando $n = p_1 \cdot p_2 \cdots p_r$ e $r \geq 3$.

Se $r = 2$, a razão $\frac{1}{2^{r-1}}$ é menor ou igual a $\frac{1}{2}$.

Suponhamos que os inteiros $s_1 \geq 1$ e $s_2 \geq 1$ são diferentes. Então ao majorar $\frac{1}{2^{(s_1+s_2+\cdots+s_r)}}$ por $\frac{1}{2^{rs}}$, estamos a dispensar pelo menos um factor 2 que divide a expressão 3.1, logo podemos majorar a expressão por $\frac{1}{2^{rs+1}}$, obtendo desta forma:

$$\frac{t_1 t_2 \left(1 + 1 + 2^2 + 2^4 + \cdots + 2^{(\overline{s}-1)2}\right)}{2^{s_1} t_1 2^{s_2} t_2} \leq \frac{1}{2^2} = \frac{1}{4}$$

porque $r = 2$.

Suponhamos, por fim, que $s_1 = s_2$. Se $(t, t_i) = t_i$, então $t_i \mid t$. Neste caso temos, por um lado, $n - 1 \equiv 2^s t \equiv 0 \pmod{t_1}$. Por outro lado,

$$n - 1 \equiv p_1 p_2 - 1 \equiv (2^{s_1} t_1 + 1)\, p_2 - 1 \equiv p_2 - 1 \equiv (2^{s_2} t_2 + 1) - 1 \equiv 2^{s_2} t_2 \pmod{t_1}$$

ou seja, $t_1 \mid t_2$. Analogamente $t_2 \mid t_1$. Logo $t_1 = t_2$, o que é absurdo pois $p_1 \neq p_2$. Concluímos que $(t, t_i) \neq t_i$ e t_i é ímpar, logo $(t, t_i) \leq \frac{t_i}{3}$. Podemos substituir 3.1 pela expressão:

$$\frac{t_1 t_2}{3} \left(1 + 1 + 2^2 + 2^4 + \cdots + 2^{(\overline{s}-1)2}\right).$$

Logo, a proporção de inteiros a, entre 1 e $n-1$, para os quais n é pseudoprimo forte é majorada por:

$$\frac{t_1 t_2 \left(1 + 1 + 2^2 + 2^4 + \cdots + 2^{(\overline{s}-1)2}\right)}{3 \cdot 2^{s_1} t_1 2^{s_2} t_2} \leq \frac{1}{3 \cdot 2^{2-1}} \leq \frac{1}{6} < \frac{1}{4}.$$

Em todos os casos, n é pseudoprimo forte com relação a, no máximo, um quarto das bases a entre 1 e $n-1$. □

O Lema 3.2.3 permite alterar o teste de *Miller* de modo a obter o seguinte teste probabilístico de primalidade.

Definição 3.2.4 (Teste de Miller–Rabin) *Seja t um inteiro e n um inteiro ímpar. Sejam $a_1, a_2, \ldots, a_t$, t elementos distintos escolhidos ao acaso em $\mathbb{Z}_n^*$. Se n é pseudoprimo para todas bases a_i, $1 \leq i \leq t$, então n é primo com probabilidade igual a $1 - \left(\frac{1}{4}\right)^t$. Caso contrário n é composto.*

Se n é um inteiro composto, então, pelo Lema 3.2.3, a probabilidade de n ser um pseudoprimo forte para uma base a é $\frac{1}{4}$. Ao aplicar o teste de *Miller–Rabin* com t bases distintas, obtemos um erro máximo de $\left(\frac{1}{4}\right)^t$. Se t é igual a 50, a probabilidade de este algoritmo errar respondendo que n é primo é remota, pois $\left(\frac{1}{4}\right)^{50} = 10^{\log_{10}\left(\frac{1}{4}\right)^{50}} \simeq 10^{-30}$.

Apesar de esta possibilidade ser remota, se o teste errar ao afirmar a primalidade de p ou q, que escolhemos como parâmetros da nossa cifra (RSA) através do teste do *Miller–Rabin*, então é muito provável que o sistema de cifragem não consiga descodificar as mensagens cifradas, uma vez que o Teorema de *Euler*: $a^{\lambda\phi(n)+1} \equiv a \pmod{n}$ valerá para muito poucos inteiros a se p ou q não for primo! Torna-se necessário obter um algoritmo rápido que decida se p é primo ou composto de maneira a que a resposta seja sempre correcta. *G. Miller* [26], mostra que se n passar o teste de *Miller–Rabin* para todas as bases $a < 2\log^2(n)$, com $(a, n) = 1$, podemos declarar com toda a certeza que n primo, se a *Hipótese de Riemann Estendida* for verdadeira. Para tal, basta considerar o Teorema 3.2.4 de *E. Bach* [2], que assume a *Hipótese de Riemann Estendida*, Teorema esse que melhora anteriores resultados de *N. Ankeny* [1].

3.2.2.1 A Hipótese de Riemann Estendida

Seja n um inteiro. Um *carácter* módulo n é uma função $\chi : \mathbb{Z}_n^* \longrightarrow \mathbb{C}^*$ que é homomorfismo de grupos multiplicativos entre $\mathbb{Z}_n^*$ e $\mathbb{C}^*$. Todo o carácter χ pode ser extendido a uma função $\chi' : \mathbb{Z}^* \longrightarrow \mathbb{C}^*$ onde:

$$\chi'(m) = \begin{cases} \chi(m \pmod{n}) & \text{se } (m,n) = 1, \\ 0 & \text{se } (m,n) \neq 1. \end{cases}$$

Vamos usar o mesmo símbolo para denotar χ e χ'. Definimos ainda o *carácter principal* módulo n como sendo:

$$\chi_1(m) = \begin{cases} 1 & \text{se } (m,n) = 1, \\ 0 & \text{se } (m,n) \neq 1. \end{cases}$$

Para todo o carácter χ, a *L–função de Dirichlet* para χ é a função L_χ na variável complexa z, definida pela seguinte série infinita:

$$L_\chi(z) = \sum_{n=1}^{\infty} \frac{\chi(n)}{n^z}$$

L_{χ_1} pode ser meromorficamente (função analítica exceptuando singularidades isoladas que sejam polos) estendida a uma função analítica definida na região $\mathrm{Re}(z) > 0$, onde $z = 1$ é o único polo. Se $\chi \neq \chi_1$, então a série L_χ converge para todo o $z \in \mathbb{C}$ com $\mathrm{Re}(z) > 0$ e converge absolutamente para $\mathrm{Re}(z) > 1$ (Veja [22]).

Hipótese 3.2.1 (*Riemann Estendida*) *Para todo o carácter χ, os zeros da função L_χ em $\{z \in \mathbb{C} : 0 < \mathrm{Re}(z) \leq 1\}$ estão sobre a recta $\mathrm{Re}(z) = \frac{1}{2}$.*

Vamos enunciar o Teorema de *Bach*, principal resultado usado para a prova do lema 3.2.5.

Teorema 3.2.4 (Bach) *Assumindo a* Hipótese de Riemann Estendida, *se G é subgrupo multiplicativo próprio de $\mathbb{Z}_n^*$, então existe um inteiro $a < 2\log^2(n)$ que não está em G, ou seja, $a \in \mathbb{Z}_n^* \backslash G$.*

Lema 3.2.5 *Assumindo a* Hipótese de Riemann Estendida, *se n é um inteiro composto, então existe um inteiro a estritamente entre 1 e $2\log^2(n)$ tal que n não é pseudoprimo para a base a.*

Demonstração. Seja $n - 1 = 2^s t$, com t ímpar.

Caso 1: n é divisível por uma potência de um primo p, $p^2 \mid n$.

Seja G o conjunto dos elementos de $a \in \mathbb{Z}_{p^2}^*$ tais que $a^{p-1} \equiv 1 \pmod{p^2}$. Vamos ver que $G \neq \mathbb{Z}_{p^2}^*$. Como p é primo, existe uma raiz primitiva g de $\mathbb{Z}_{p^2}^*$. As únicas soluções de $\alpha^{p-1} \equiv 1 \pmod{p^2}$ são: $\{g^p, g^{2p}, \ldots, g^{(p-1)p}\}$. Então existe um inteiro i, estritamente entre 0 e $\phi(p^2)$, tal que $\alpha = g^i \in \mathbb{Z}_{p^2}^*$ e $(g^i)^{p-1} \not\equiv 1 \pmod{p^2}$, logo $\alpha = g^i \notin G$. Pelo Teorema 3.2.4 aplicado a G, existe um inteiro a, $1 < a < 2log^2(n)$, tal que $a^{p-1} \not\equiv 1 \pmod{p^2}$.

Pretendemos mostrar que $a^{n-1} \not\equiv 1 \pmod{n}$. Se $a^{n-1} \equiv 1 \pmod{n}$, então existiria um inteiro h tal que $a^{n-1} + nh = 1$. Como $p^2 \mid n$, então $a^{n-1} + p^2mh = 1$ ou, de forma equivalente, $a^{n-1} \equiv 1 \pmod{p^2}$. Logo $\mathrm{ord}_{p^2}(a) \mid (n-1)$ e $\mathrm{ord}_{p^2}(a) \mid \phi(p^2) = p(p-1)$, e portanto $\mathrm{ord}_{p^2}(a) \mid (p-1)$, porque a alternativa é $\mathrm{ord}_{p^2}(a) = p$, que não pode dividir $n-1$. Chegamos a uma contradição pois $a^{p-1} \not\equiv 1 \pmod{p^2}$. Podemos ver facilmente que se $a^{n-1} \not\equiv 1 \pmod{n}$, então $a^{n-1} = (a^t)^{2^s} \not\equiv 1 \pmod{n}$ e $a^{n-1} = (a^{2^i t})^{2^{s-i}} \not\equiv 1 \pmod{n}$ para todo i entre 0 e $s-1$. As a–sequências são do Tipo II.

Caso 2: n tem divisores p e q tais que $p-1 = 2^{s_p}t_p$ e $q-1 = 2^{s_q}t_q$, com t_p e t_q ímpares.

Seja G o conjunto de todos os elementos de $\mathbb{Z}_p^*$ que são resíduos quadráticos. Pelo Teorema 3.2.4 aplicado a G, existe um inteiro a, $1 < a < 2\log^2(p)$, que não é resíduo quadrático.

Como $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$ e $\mathrm{ord}_p(a)$ é o menor inteiro tal que $a^{\mathrm{ord}_p(a)} \equiv 1 \pmod{p}$, então $2^{s_p} \mid \mathrm{ord}_p(a)$ e $2^{s_p+1} \nmid \mathrm{ord}_p(a)$. Note-se que $a^t \not\equiv 1 \pmod{n}$. De facto, caso contrário, existe $h \in \mathbb{Z}$ tal que $a^t + hn = 1$ e, como $p \mid n$, temos $a^t + pmh = 1$, ou seja, $a^t \equiv 1 \pmod{p}$. Portanto, $\mathrm{ord}_p(a) \mid t$ e $2^{s_p} \mid \mathrm{ord}_p(a)$, o que é uma contradição, visto que t é ímpar.

Vamos supor que $s_q < s_p$. Suponhamos que $a^{2^i t} \equiv -1 \pmod{n}$ para algum i entre 0 e $s-1$. Como $p \mid n$ e $q \mid n$, temos $a^{2^i t} \equiv -1 \pmod{p}$ e $a^{2^i t} \equiv -1 \pmod{q}$. Logo, i é o menor inteiro tal que $a^{2^{i+1}t} \equiv 1 \pmod{p}$ e $a^{2^{i+1}t} \equiv 1 \pmod{q}$. Concluímos que $\mathrm{ord}_p(a) \nmid 2^i t$ e $\mathrm{ord}_p(a) \mid 2^{i+1}t$ logo $i+1 = s_p$, $\mathrm{ord}_p(a)$ tem 2^{i+1} como factor e se $s_p > i+1$, então i não poderia ser o menor inteiro nas condições anteriores. De igual modo $\mathrm{ord}_q(a) \nmid 2^i t$ e $\mathrm{ord}_q(a) \mid 2^{i+1}t = 2^{s_p}t$. Isto contradiz o facto de $\mathrm{ord}_q(a) \mid q-1 = 2^{s_q}t_q$, porque supomos $s_q < s_p$. Então não é possível termos a–sequências do Tipo I.

Por fim, suponhamos que $s_p = s_q$ e que G é o conjunto dos elementos de $\mathbb{Z}_{pq}^*$ tais que $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right)$. Vejamos que $G \neq \mathbb{Z}_{pq}^*$.

Como p é primo, existe uma raiz primitiva g de $\mathbb{Z}_p^*$. Pelo Teorema Chinês dos Restos, existe $\alpha \in \mathbb{Z}_{pq}^*$ tal que $\alpha \equiv g \pmod{p}$ e $\alpha \equiv 1 \pmod{q}$, ou seja, $\left(\frac{\alpha}{p}\right) \neq \left(\frac{\alpha}{q}\right)$. Logo existe um elemento $\alpha \in \mathbb{Z}_{pq}^*$ tal que $\alpha \notin G$. Pelo Teorema 3.2.4 aplicado a G, existe um inteiro a, $1 < a < 2\log^2(pq)$, tal que $\left(\frac{a}{p}\right) \neq \left(\frac{a}{q}\right)$.

Se $\left(\frac{a}{p}\right) = -1$ e $\left(\frac{a}{q}\right) = 1$, obtemos, pelo critério de *Euler*, que $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$ e $a^{\frac{q-1}{2}} \equiv 1 \pmod{q}$. Portanto $2^{s_p} \mid \mathrm{ord}_p(a)$ e $2^{s_q} \nmid \mathrm{ord}_q(a)$. Como vimos atrás, $a^t \not\equiv 1 \pmod{n}$

Supondo que a a–sequência é do Tipo I, então haverá $i \in \{0, \ldots, s-1\}$ tal que $a^{2^i t} \equiv -1 \pmod{n}$, implicando $\mathrm{ord}_p(a) \nmid 2^i t$ e $\mathrm{ord}_p(a) \mid 2^{i+1}t$; e também $s_p = i+1$, da mesma maneira que em parágrafos anteriores.

Analogamente, $\mathrm{ord}_q(a) \nmid 2^i t$ e $\mathrm{ord}_q(a) \mid 2^{i+1}t$. Mas $2^{s_q} \nmid \mathrm{ord}_q(a)$ e $s_p = s_q = i+1$ leva a $\mathrm{ord}_q(a) \mid= 2^{s_q}t$, uma contradição! Portanto a a–sequência tem de ser do Tipo II.

Em todos os casos, existe um $a < 2\log^2(n)$ tal que n não é pseudoprimo forte para a base a. □

No Lema 3.2.5, para provar o caso em que se supõe que $p^2 \mid n$ com p primo, podemos usar o lema 3.2.6 que não usa a *Hipótese de Riemann Estendida.*

Lema 3.2.6 (Lenstra) *Para todo o primo p, existe um inteiro $a < 4\log^2(p)$ tal que $a^{p-1} \not\equiv 1 \pmod{p^2}$*

Demonstração. Ver [23]. □

Finalmente, com o Lema 3.2.5 obtemos um algoritmo polinomial que decide a primalidade de um número n. Apesar de o teste 3.2.5 ser polinomial de custo $O(\log^5(n))$, existem testes, como o teste de *Lenstra–Cohen*, não polinomiais, que para números com cerca de 100 casas decimais são muito rápidos (cerca de 1 minuto num bom computador!).

Definição 3.2.5 (Teste de Miller–Rabin–HRE) *Seja n um inteiro ímpar. Se n é pseudoprimo para todas bases $a < 2\log^2(n)$ e a* Hipótese de Riemann Estendida *for verdadeira, então n é primo. Caso contrário n é composto.*

Podemos indagar se haverá algum contra-exemplo para o Teste de *Miller–Rabin–HRE*! *Pomerance*, *Selfridge* e *Wagstaff* [34], efectuaram cálculos em massa e mostraram que para as bases 2, 3 e 5 o teste de *Miller–Rabin* identifica todos os primos para inteiros $n < 25 \cdot 10^9$, à excepção de 13 números. Desses apenas o inteiro $3\,215\,031\,751 = 151 \cdot 751 \cdot 28\,351$ é pseudoprimo forte para a base 7, e este número não é pseudoprimo forte para a base 11.

Jaeschke [15], mostrou que só há 101 pseudoprimos fortes menores que 10^{12} para as bases 2, 3 e 5, há 9 se adicionarmos a base 7 e nenhum se adicionarmos a base 11.

A Tabela 3.2 mostra que, na realidade, os resultados são bem melhores do que se previa no teste de *Miller–Rabin–HRE.* Podemos ver que o número de bases necessárias para testar a primalidade de números n menores que 10^{14} é mais parecida com $c\log_{10} n$ do que $2(\ln n)^2$.

Apesar do sucesso das previsões do teste de *Miller–Rabin–HRE* 3.2.5 para valores de n menores que 10^{14}, é preferível aplicar somente o teste probabilístico de *Miller–Rabin* para um número razoável t de bases aleatórias. Assim, não usamos um resultado que ainda

ψ_k	$b \in \mathcal{W}(n)$	n	$2(\ln n)^2$	$\log_{10} n$
ψ_1	3	2 047	117	3.3
ψ_2	5	1 373 653	400	6.1
ψ_3	7	25 326 001	582	7.4
ψ_4	11	3 215 031 751	959	9.5
ψ_5	13	2 152 302 898 747	1613	12.3
ψ_6	17	3 474 749 660 383	1668	12.5
ψ_7	19	341 550 071 728 321	2240	14.5
ψ_8	23	341 550 071 728 321	2240	14.5

Tabela 3.2: *Lista de valores de* ψ_k $(1 \leq k \leq 8)$, ψ_k *é o menor inteiro que é pseudoprimo forte para os k primeiros números primos.*

não foi provado, pois o teste de *Miller–Rabin–HRE* depende da *Hipótese de Riemann Estendida*, que se pensa ser verdadeira mas que ainda continua a ser um problema em aberto.

3.2.3 Teste de Solovay–Strassen

Este teste é atribuído a *Solovay* e *Strassen* [42]. Trata-se de um dos testes probabilísticos de primalidade mais divulgados na criptografia de chave pública e em particular no RSA. É recomendado pelos autores do sistema de cifragem RSA [36].

No *PGP v7.0.3*, o método de gerar números primos aleatórios é semelhante ao método usado no *GnuPG v1.06* mas com uma pequena diferença: o teste de primalidade *Fermat* é substituído pelo teste de primalidade de *Solovay–Strassen*. Além de ser um teste de primalidade rápido, é um teste mais forte do que o teste de primalidade de *Fermat*, como se pode ver no Teorema 3.2.7.

Grosso modo, o teste de primalidade de *Solovay–Strassen* é baseado no critério de *Euler*, Teorema 1.2.17. Seja n um inteiro ímpar em que a é um número primo com n. Se a é tal que $a^{\frac{n-1}{2}} \not\equiv \left(\frac{a}{n}\right) \pmod{n}$, então diz-se que n *falha* o teste primalidade para a base a, ou seja, podemos afirmar que n é composto. Caso contrário, se $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}$, então diz-se que n *passa* o teste primalidade para a base a e obtemos como resposta: n é

provavelmente primo.

Um dos passo essenciais para aplicar este teste de primalidade consiste em calcular eficientemente o símbolo de *Jacobi*. Mas se b é um inteiro ímpar, $a < b$ e $(a, b) = 1$, então pelas propriedades do símbolo de Jacobi e pela Lei da reciprocidade quadrática 1.2.1.4, $\left(\frac{a}{b}\right)$ pode ser calculado eficientemente através da seguinte relação:

$$\left(\frac{a}{b}\right) = \begin{cases} 1 & \text{se } a = 1, \\ \left(\frac{-a}{b}\right) \cdot (-1)^{\frac{b-1}{2}} & \text{se } a \text{ é negativo}, \\ \left(\frac{a/2}{b}\right) \cdot (-1)^{\frac{b^2-1}{8}} & \text{se } a \text{ é par}, \\ \left(\frac{b \pmod a}{a}\right) \cdot (-1)^{\frac{(a-1)(b-1)}{4}} & \text{se } a \text{ é ímpar}. \end{cases} \tag{3.2}$$

Ao aplicar o teste de primalidade de *Solovay–Strassen* podemos certificar a primalidade de n com um certo grau de certeza. Por outro lado, tal como acontece no teste de probabilístico de *Fermat*, o teste de *Solovay–Strassen* não fornece uma prova da primalidade de n. Por exemplo, apesar de $n = 561 = 3 \times 11 \times 17$ ser composto, se escolhermos a base $a = 2$ obtemos $2^{\frac{561-1}{2}} \equiv \left(\frac{2}{561}\right) \pmod n$.

Definição 3.2.6 *Seja n um números inteiro composto e seja a tal que $1 < a \leq n - 1$. Diz-se que n é* pseudoprimo de Euler *para a base a, se $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod n$.*

Note-se que, se a é primo com n, então a congruência $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod n$ implica que $a^{n-1} \equiv \left(\frac{a}{n}\right)^2 \equiv 1 \pmod n$. Isto mostra que o teste de *Solovay–Strassen* implica o teste de *Fermat*. Em particular, provamos o Teorema seguinte.

Teorema 3.2.7 *O conjunto das bases $a \in \mathbb{Z}_n^*$ que verificam $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod n$ está contido no conjunto das bases $a \in \mathbb{Z}_n^*$ que verificam $a^{n-1} \equiv 1 \pmod n$.*

Desta forma, o teste de *Solovay–Strassen* permite reconhecer números compostos que não são reconhecido pelo teste de *Fermat*. Por exemplo, o número de Carmichael 561 não é um *pseudoprimo de Euler* para a base 5, pois $5^{\frac{561-1}{2}} \equiv 67 \pmod{561}$ e $\left(\frac{5}{561}\right) = 1$. O Teorema 3.2.7 mostra que é desnecessário aplicar o teste de primalidade de *Fermat* quando se aplica o teste de primalidade de *Solovay-Strassen*.

Para cada n, definimos

$$P_n = \left\{ b \in \mathbb{Z}_n^* : b^{\frac{n-1}{2}} \not\equiv \left(\frac{b}{n}\right) \pmod n \right\}.$$

Note-se que n não é pseudoprimo de *Euler* para toda a base $b \in P_n$.

Para cada n, considere os conjuntos

$$K_n = \left\{b \in \mathbb{Z}_n^* : b^{\frac{n-1}{2}} \equiv 1 \pmod{n}\right\}$$
$$L_n = \left\{b \in \mathbb{Z}_n^* : \left(\tfrac{b}{n}\right) \equiv 1 \pmod{n}\right\}$$
$$M_n = \left\{b \in \mathbb{Z}_n^* : b^{\frac{n-1}{2}} \cdot \left(\tfrac{b}{n}\right) \equiv 1 \pmod{n}\right\}.$$

Sabemos que $\left(\frac{b}{n}\right) = \pm 1$ e que a congruência $b^{\frac{n-1}{2}} \cdot \left(\frac{b}{n}\right) \equiv 1 \pmod{n}$ implica que $b^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n}$, respectivamente. Portanto, se $b \in M_n$, n é pseudoprimo de *Euler* para a base b. Logo podemos escrever $M_n = \mathbb{Z}_n^* - P_n$.

Para cada n, definimos os conjuntos

$$K'_n = \left\{b \in \mathbb{Z}_n^* : b^{\frac{n-1}{2}} \equiv -1 \pmod{n}\right\}$$
$$L'_n = \left\{b \in \mathbb{Z}_n^* : \left(\tfrac{b}{n}\right) \equiv -1 \pmod{n}\right\}$$
$$M'_n = \left\{b \in \mathbb{Z}_n^* : b^{\frac{n-1}{2}} \cdot \left(\tfrac{b}{n}\right) \equiv -1 \pmod{n}\right\}.$$

Vamos provar o Teorema de *Moinier* necessário para a prova do Lema 3.2.9. O Teorema de *Moinier* dá-nos informação acerca da cardinalidade de M_n.

Teorema 3.2.8 (Moinier) *Para todo n, se $p_1, p_2, \ldots, p_r$ são os factores primos distintos de n, então*

$$|M_n| = |\mathbb{Z}_n^* - P_n| = \delta_n \cdot \prod_{i=1}^{r} \left(\frac{n-1}{2}, p_i - 1\right),$$

em que δ_n toma um dos valores $\frac{1}{2}$, 1 ou 2.

Demonstração. Considere $b \in M_n$, então $b^{\frac{n-1}{2}} \cdot \left(\frac{b}{n}\right) \equiv 1 \pmod{n}$. Esta congruência equivale a $b^{\frac{n-1}{2}} \equiv \left(\frac{b}{n}\right) \equiv 1 \pmod{n}$ ou $b^{\frac{n-1}{2}} \equiv \left(\frac{b}{n}\right) \equiv -1 \pmod{n}$. Das congruência anteriores concluímos que $b \in (M_n \cap L_n)$ ou $b \in (M'_n \cap L'_n)$. Logo, verifica-se a igualdade

$$M_n = (K_n \cap L_n) \cup (K'_n \cap L'_n). \tag{3.3}$$

Suponhamos que $K'_n \cap L'_n \neq \emptyset$. Seja $b_0 \in K'_n \cap L'_n$, ou seja, $b_0^{\frac{n-1}{2}} \equiv \left(\frac{b_0}{n}\right) \equiv -1 \pmod{n}$. Considere a função

$$\begin{aligned} \lambda_n : \ \mathbb{Z}_n^* &\longrightarrow \mathbb{Z}_n^* \\ b &\longmapsto bb_0 \end{aligned}$$

A função $\lambda_n(b)$ é bijectiva pois $\mathbb{Z}_n^*$ é grupo multiplicativo finito e para todo o $a \in \mathbb{Z}_n^*$ temos $a = ab_0^{-1}b_0 = \lambda_n(ab_0^{-1})$.

Se $b \in (K_n \cap L_n)$, então $b^{\frac{n-1}{2}} \equiv \left(\frac{b}{n}\right) \equiv 1 \pmod{n}$. Mas como $(\lambda_n(b))^{\frac{n-1}{2}} \equiv (bb_0)^{\frac{n-1}{2}} \equiv b^{\frac{n-1}{2}} \cdot b_0^{\frac{n-1}{2}} \equiv 1 \cdot (-1) \equiv -1 \pmod{n}$, resulta que $\lambda_n(b) \in K'_n$. Por outro lado, temos $\left(\frac{\lambda_n(b)}{n}\right) \equiv \left(\frac{bb_0}{n}\right) \equiv \left(\frac{b}{n}\right)\left(\frac{b_0}{n}\right) \equiv 1 \cdot (-1) \equiv -1 \pmod{n}$, isto implica que $\lambda_n(b) \in L'_n$. Daqui resulta que, se $b \in (K_n \cap L_n)$, então $\lambda_n(b) \in (K'_n \cap L'_n)$. Analogamente se mostra que, se $b \in (K'_n \cap L'_n)$, então $\lambda_n(b) \in (K_n \cap L_n)$.

Visto que λ_n é bijectiva, ficou provado que quando $K'_n \cap L'_n \neq \emptyset$, temos $|K_n \cap L_n| = |K'_n \cap L'_n|$. Pela igualdade 3.3 e pela implicação anterior segue que

$$|M_n| = \begin{cases} |K_n \cap L_n| & \text{se } K'_n \cap L'_n = \emptyset, \\ 2|K_n \cap L_n| & \text{se } K'_n \cap L'_n \neq \emptyset. \end{cases} \tag{3.4}$$

Note-se que $K_n = (K_n \cap L_n) \cup (K_n \cap L'_n)$. Se $K_n \cap L'_n \neq \emptyset$, com um argumento semelhante usando a função bijectiva $\mu_n(b) = bb_0$, $b_0 \in (K_n \cap L'_n)$, provamos que $|K_n \cap L_n| = |K_n \cap L'_n|$. Logo,

$$|K_n \cap L_n| = \begin{cases} |K_n| & \text{se } K_n \cap L'_n = \emptyset, \\ \frac{1}{2}|K_n| & \text{se } K_n \cap L'_n \neq \emptyset. \end{cases} \tag{3.5}$$

De 3.4 e 3.5, é fácil ver que

$$|M_n| = \begin{cases} |K_n| & \text{se } (K'_n \cap L'_n = \emptyset \text{ e } K_n \cap L'_n = \emptyset) \text{ ou } (K'_n \cap L'_n \neq \emptyset \text{ e } K_n \cap L'_n \neq \emptyset), \\ \frac{1}{2}|K_n| & \text{se } K'_n \cap L'_n = \emptyset \text{ e } K_n \cap L'_n \neq \emptyset, \\ 2|K_n| & \text{se } K'_n \cap L'_n \neq \emptyset \text{ e } K_n \cap L'_n = \emptyset. \end{cases}$$

Para concluir a demonstração do Teorema de *Moinier* só falta mostrar que $|K_n| = \prod_{i=1}^{r} \left(\frac{n-1}{2}, p_i - 1\right)$ e, com efeito, que

$$|M_n| = \delta_n \prod_{i=1}^{r} \left(\frac{n-1}{2}, p_i - 1\right), \quad \delta_n \in \left\{\frac{1}{2}, 1, 2\right\}.$$

Representamos $n = p_1^{k_1} p_2^{k_2} \cdots p_r^{k_r}$, onde $p_1, p_2, \ldots, p_r$ são factores primos distintos de n. Para cada $i = 1, \ldots, r$, seja g_i o gerador de $\mathbb{Z}^*_{p_i^{k_i}}$. Como $p_i^{k_i} \mid n$, então tomando a congruência $b^{\frac{n-1}{2}} \equiv 1 \pmod{n}$ obtemos

$$\tfrac{n-1}{2} \cdot \operatorname{ind}_{g_i}(b) \equiv 0 \pmod{\varphi(p_i^{k_i})}, \text{ para } i = 1, \ldots, r. \tag{3.6}$$

Pelo Teorema 1.2.25, cada congruência de 3.6 tem exactamente $\left(\frac{n-1}{2}, (p_i - 1)p_i^{k_i - 1}\right)$ soluções, em que $i = 1, \ldots, r$. Por outro lado, 1 pode ser escrito como combinação linear de n e

$\frac{n-1}{2}$, isto é, $n-2\left(\frac{n-1}{2}\right)=1$. Isto significa que n e $\frac{n-1}{2}$ não têm factores primos em comum. Logo $p_i^{k_i-1} \nmid \frac{n-1}{2}$ e, consequentemente, obtemos $\left(\frac{n-1}{2},(p_i-1)p_i^{k_i-1}\right)=\left(\frac{n-1}{2},p_i-1\right)$. Finalmente, podemos concluir que a congruência $b^{\frac{n-1}{2}} \equiv 1 \pmod{n}$ tem exactamente $\prod_{i=1}^{r}\left(\frac{n-1}{2},p_i-1\right)$ soluções, ou seja, $|K_n|=\prod_{i=1}^{r}\left(\frac{n-1}{2},p_i-1\right)$. □

O seguinte resultado fornece informação acerca da quantidade de bases a para o qual n é pseudoprimo de Euler e, em particular, que nenhum número composto é pseudoprimo de Euler para todas as bases a primas com n.

Lema 3.2.9 *Se n é um número composto ímpar, então n é pseudoprimo de Euler com relação a, no máximo, metade das bases a tais que $1 \leq a \leq n-1$ e $(a,n)=1$.*

Demonstração. Pretendemos mostrar que $\frac{|\mathbb{Z}_n^*-P_n|}{\varphi(n)} \leq \frac{1}{2}$. Escrevemos $n=\prod_{i=1}^{r} p_i^{k_i}$ em que $p_1,p_2,\ldots,p_r$ são os factores primos ímpares distintos de n.

Suponhamos que para algum i temos $k_i \geq 2$. Das propriedades multiplicativas da função de *Euler* e do Teorema de Moinier 3.2.8 resulta que

$$\begin{aligned}\frac{|\mathbb{Z}_n^*-P_n|}{\varphi(n)} &= \delta_n \prod_{i=1}^{r} \frac{\left(\frac{n-1}{2},p_i-1\right)}{p_i^{k_i-1}(p_i-1)} \leq \delta_n \prod_{i=1}^{r} \frac{p_i-1}{p_i^{k_i-1}(p_i-1)} \\ &\leq \delta_n \prod_{i=1}^{r} \frac{1}{p_i^{k_i-1}} \leq \frac{2}{3}\ .\end{aligned}$$

Logo, $\mathbb{Z}_n^*-P_n$ é um subgrupo próprio de $\mathbb{Z}_n^*$, pois o símbolo de Jacobi é multiplicativo. Daqui resulta que $|\mathbb{Z}_n^*-P_n| \mid |\mathbb{Z}_n^*|$, ou seja, existe um inteiro k tal que $|\mathbb{Z}_n^*|=k|\mathbb{Z}_n^*-P_n| \geq 2|\mathbb{Z}_n^*-P_n|$. Ficou provado que $\frac{|\mathbb{Z}_n^*-P_n|}{\varphi(n)} \leq \frac{1}{2}$.

Suponhamos agora que para todo i temos $k_i=1$, isto é, $n=p_1p_2\cdots p_r$. Por absurdo, vamos supor que $\mathbb{Z}_n^*=M_n$. Seja g_i um gerador de $\mathbb{Z}_{p_i}^*$. Usamos o Teorema Chinês dos Restos 1.2.26 para encontrar um elemento a de $\mathbb{Z}_n^*$ de maneira que $a \equiv g \pmod{p_1}$ e $a \equiv 1 \pmod{\frac{n}{p_1}}$. Por hipótese $\mathbb{Z}_n^*=M_n$, logo todos os elementos de $\mathbb{Z}_n^*$ verificam o critério de *Euler*, ou seja, $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}$. Mas $\left(\frac{a}{n}\right)=\left(\frac{a}{p_1}\right)\ldots\left(\frac{a}{p_r}\right)=\left(\frac{a}{p_1}\right)=\left(\frac{g}{p_1}\right)=-1$.

É fácil ver que g é não resíduo quadrático módulo p_1. Caso contrário, temos $g \equiv x^2 \pmod{p_1}$ e pelo Teorema de Fermat obtemos $g^{\frac{p_1-1}{2}} \equiv (x^2)^{\frac{p_1-1}{2}} \equiv x^{p_1-1} \equiv 1 \pmod{p_1}$. Logo $\mathrm{ord}_{p_1} g \leq \frac{p_1-1}{2} < \varphi(p_1)=|\mathbb{Z}_{p_1}^*|$. Isto contradiz o facto de g ser gerado de $\mathbb{Z}_{p_1}^*$.

Por fim, como $\frac{n}{p_1} \mid n$ temos $a^{\frac{n-1}{2}} \equiv -1 \pmod{\frac{n}{p_1}}$, visto que $a^{n-1}2 \equiv -1 \pmod{n}$. Isto é uma contradição com a congruência $a \equiv 1 \pmod{\frac{n}{p_1}}$. □

Com o lema anterior podemos definir o seguinte teste de primalidade.

Definição 3.2.7 (Teste de Solovay–Strassen) *Seja t um inteiro e n um inteiro ímpar. Sejam $a_1, a_2, \ldots, a_t$, t elementos distintos escolhidos ao acaso em $\mathbb{Z}_n^*$. Se n é pseudoprimo de Euler para todas bases a_i, $1 \leq i \leq t$, então n é primo com probabilidade igual a $1 - \frac{1}{2^t}$. Caso contrário n é composto.*

Como vimos, pelo Lema 3.2.9, se n é um inteiro composto, então a probabilidade de n ser um pseudoprimo de Euler para uma base a é $\frac{1}{2}$. Ao aplicar o teste de *Solovay–Strassen* com t bases distintas, obtemos um erro máximo de $\left(\frac{1}{2}\right)^t$. Por exemplo, se o número t de bases distintas é igual a 100, então a probabilidade de este algoritmo errar respondendo que n é primo é muito pequena, pois $\left(\frac{1}{2}\right)^{100} = 10^{\log_{10}\left(\frac{1}{2}\right)^{100}} \simeq 10^{-30}$.

Surge uma questão natural: qual a relação do teste de *Solovay–Strassen* com o teste de *Miller–Rabin*?

O próximo resultado responde parcialmente à questão anterior e mostra que ser um pseudoprimo forte implica ser pseudoprimo de Euler!

Proposição 3.2.2 *O conjunto das bases $a \in \mathbb{Z}_n^*$ em que n é pseudoprimo de Euler está contido no conjunto das bases $a \in \mathbb{Z}_n^*$ em que n é pseudoprimo forte.*

Demonstração. Seja n um inteiro ímpar e b um inteiro primo com n. Queremos mostrar que se n é um pseudoprimo *forte* para a base b, então n é um pseudoprimo de *Euler* para a mesma base. Sejam s e t inteiros tais que $n - 1 = 2^s t$.

Caso (i): Suponhamos que $b^t \equiv 1 \pmod{n}$. Então $b^{2^{s-1}t} \equiv b^{\frac{n-1}{2}} \equiv 1 \pmod{n}$. Queremos mostrar que $\left(\frac{b}{n}\right) = 1$. Temos $1 = \left(\frac{1}{n}\right) = \left(\frac{b^t}{n}\right) = \left(\frac{b}{n}\right)^t$. Como t é um número ímpar, temos forçosamente $\left(\frac{b}{n}\right) = 1$.

Caso (ii): Suponha-se que $b^{\frac{n-1}{2}} \equiv -1 \pmod{n}$. Pretendemos mostrar que $\left(\frac{b}{n}\right) = -1$. Vamos primeiro provar um resultado essencial para a demonstração deste caso.

Lema 3.2.10 *Suponhamos que $b^{\frac{n-1}{2}} \equiv -1 \pmod{n}$ e seja p um divisor de n. Sejam s' e t' inteiros tais que $p - 1 = 2^{s'} t'$, com t' ímpar. Temos $s' \geq s$ e*

$$\left(\frac{b}{p}\right) = \begin{cases} -1 & \text{se } s' = s, \\ 1 & \text{se } s' > s. \end{cases}$$

Demonstração. Temos que t' é ímpar, logo a congruência $b^{\frac{n-1}{2}} \equiv b^{2^{s-1}t} \pmod{n}$ é equivalente a $\left(b^{2^{s-1}t'}\right)^t \equiv -1 \pmod{n}$. Como $p \mid n$,

$$\left(b^{2^{s-1}t'}\right)^t \equiv -1 \pmod{p}. \tag{3.7}$$

Se $s' < s$, não se verifica o Teorema de *Fermat* porque a congruência (3.7) implica que $b^{p-1} \equiv b^{2^{s'}t'} \not\equiv 1 \pmod{p}$. Portanto, $s' \geq s$.

Se $s' = s$, como t é ímpar, então pelo critério de *Euler* a congruência 3.7 implica que $\left(\frac{b}{p}\right) \equiv b^{\frac{p-1}{2}} \equiv b^{2^{s'-1}t'} \pmod{p}$ é congruente com $-1 \pmod{p}$. Por outro lado, se $s' > s$, então a congruência 3.7 implica que

$$\left(b^{\frac{p-1}{2}}\right)^t \equiv \left(b^{2^{s'-1}t'}\right)^t \equiv \left(b^{2^{s-1}t'}\right)^{t2^{s'-s}} \equiv (-1)^{t2^{s'-s}} \equiv 1 \pmod{p}.$$

Como t é ímpar, concluímos que $\left(\frac{b}{p}\right) = 1$. □

Retomamos a demonstração do segundo caso. Escrevemos n como o produto de primos, ou seja, $n = \prod_{i=1}^{n} p_i^{\alpha_i}$. Para cada primo p_i existem inteiros s_i e t_i tais que $p_i - 1 = 2^{s_i}t_i$, com t_i ímpar. Definimos o inteiro k por $\sum_{1 \leq i \leq n,\, s_i = s} \alpha_i$. Pelo Lema 3.2.10, sabemos que $s_i \geq s$, para todo i entre 1 e n, e que

$$\left(\frac{b}{n}\right) = \prod_{i=1}^{n} \left(\frac{b}{p_i}\right)^{\alpha_i} = \prod_{s_i > s} 1^{\alpha_i} \cdot \prod_{s_i = s} (-1)^{\alpha_i} = (-1)^{\sum_{1 \leq i \leq n,\, s_i = s} \alpha_i} = (-1)^k.$$

Falta agora mostrar que k é ímpar.

Por um lado, se $s_i = s$, então

$$p_i = 1 + 2^{s_i}t_i \equiv 1 + 2^s t_i \equiv 1 + 2^s(2u_i + 1) \equiv 1 + 2^s \pmod{2^{s+1}}.$$

Por outro lado, se $s_i > s$, então $p_i = 1 + 2^{s_i}t_i \equiv 1 \pmod{2^{s+1}}$.

Visto que $n = 1 + 2^s t$ e t é ímpar, então $n \equiv 1 + 2^s(2u + 1) \equiv 1 + 2^s \pmod{2^{s+1}}$ e

$$1 + 2^s \equiv \prod_{i=1}^{n} p_i^{\alpha_i} \equiv (1 + 2^s)^k \equiv 1 + k2^s \pmod{2^{s+1}}.$$

Da congruência anterior obtemos $2^s \equiv k2^s \pmod{2^{s+1}}$. Logo, concluímos que k tem de ser ímpar e, por isso, $\left(\frac{b}{n}\right) = (-1)^k = -1$.

Caso (iii): Por fim, suponhamos que $b^{2^{r-1}t} \equiv -1 \pmod{n}$, para algum r entre 1 e $s - 1$. Então $b^{\frac{n-1}{2}} \equiv 1 \pmod{n}$. Queremos mostrar que $\left(\frac{b}{n}\right) = 1$.

Novamente, enunciamos o seguinte resultado auxiliar cuja prova é semelhante à prova do Lema 3.2.10.

Lema 3.2.11 *Suponhamos que* $b^{2^{r-1}t} \equiv -1 \pmod{n}$ *e seja* p *um divisor de* n. *Sejam* s' *e* t' *inteiros tais que* $p-1 = 2^{s'}t'$, *com* t' *ímpar. Temos* $s' \geq r$ *e*

$$\left(\frac{b}{p}\right) = \begin{cases} -1 & \text{se } s' = r, \\ 1 & \text{se } s' > r. \end{cases}$$

Escrevemos n como o produto de primos, ou seja, $n = \prod_{i=1}^{n} p_i^{\alpha_i}$. Para cada primo p_i existem inteiros s_i e t_i tais que $p_i - 1 = 2^{s_i}t_i$, com t_i ímpar. Definimos o inteiro k por $\sum_{1 \leq i \leq n,\, s_i = s} \alpha_i$. Pelo Lema 3.2.11, sabemos que $s_i \geq r$, para todo i entre 1 e n, e que $\left(\frac{b}{n}\right) = (-1)^k$.

Falta mostrar que k é par. Por um lado, se $s_i = r$, então $p_i = 1 + 2^{s_i}t_i \equiv 1 + 2^r \pmod{2^{r+1}}$. Por outro lado, se $s_i > r$, então $p_i = 1 + 2^{s_i}t_i \equiv 1 \pmod{2^{r+1}}$.

Visto que $n = 1 + 2^s t$ e t é ímpar, então $n \equiv 1 \pmod{2^{r+1}}$ e

$$1 \equiv \prod_{i=1}^{n} p_i \equiv (1 + 2^r)^k \equiv 1 + k2^r \pmod{2^{r+1}}.$$

Da congruência anterior concluímos que k tem de ser par. Logo $\left(\frac{b}{n}\right) = (-1)^k = 1$. □

O próximo resultado mostra que, em certas situações, ser pseudoprimo *forte* é equivalente a ser pseudoprimo de *Euler*.

Proposição 3.2.3 *Se* $n \equiv 3 \pmod{4}$, *então o conjunto das bases* $b \in \mathbb{Z}_n^*$ *em que* n *é pseudoprimo Euler é o mesmo em que* n *é pseudoprimo forte.*

Demonstração. Se $n \equiv 3 \pmod{4}$, então $n = 3 + 4k$, para algum inteiro k. Como $\frac{n-1}{2} = 1 + 2k$, neste caso $s = 1$ e $t = \frac{n-1}{2}$. Logo n é pseudoprimo *forte* para a base b se $b^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n}$.

Por definição, ser é pseudoprimo de *Euler* implica a congruência anterior. Reciprocamente, suponhamos que $b^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n}$. Pretendemos provar que $\left(\frac{b}{n}\right)$ coincide com o valor de $b^{\frac{n-1}{2}} \pmod{n}$. Para $n \equiv 3 \pmod{4}$ temos $\left(\frac{\pm 1}{n}\right) = \pm 1$. Basta ver que $1 \equiv 1^2 \pmod{n}$ e que $\left(\frac{-1}{n}\right) \equiv (-1)^{\frac{n-1}{2}} \equiv -1 \pmod{n}$, pois neste caso $\frac{n-1}{2} = 1 + 2k$ é ímpar. Notando que

$$\left(\frac{b}{n}\right) = \left(\frac{b \cdot (b^2)^{\frac{n-3}{4}}}{n}\right) = \left(\frac{b^{\frac{n-1}{2}}}{n}\right) = \left(\frac{\pm 1}{n}\right) = \pm 1,$$

fica provado que $\left(\frac{b}{n}\right) \equiv b^{\frac{n-1}{2}} \pmod{n}$. □

Finalmente, ao testarmos t bases, a probabilidade de o teste de *Solovay–Strassen* errar é de $\left(\frac{1}{2}\right)^t$, enquanto que para o teste de *Miller–Rabin* a probabilidade de errar é de $\left(\frac{1}{4}\right)^t$. Portanto, podemos concluir que o teste *Miller–Rabin* não é pior do que o teste de *Solovay–Strassen*.

3.3 Métodos de factorização

Vimos que a segurança do sistema de cifragem reside na dificuldade do problema da factorização de números inteiros. Parece ser um problema tão difícil que nem sequer é possível provar que é difícil. Mas, por outro lado, muitos algoritmos de factorização de inteiros sofisticados e eficientes são inventados obrigando o parâmetro n, o módulo, a crescer até aos 1024 bits.

Efectuar divisões sucessivas por todos os primos até $\sqrt{n}$ para a factorizar um inteiro n grande é, na prática, uma tarefa impossível de realizar. No entanto, por questões de rapidez, continua-se a utilizar divisões sucessivas até um limite L estabelecido, como uma "primeira tentativa" em testes de primalidade ou factorizações.

Exemplo 3.3.1 Vamos procurar alguns factores de $n = 3^{21} + 1 = 10460353204$, efectuando divisões sucessivas por todos os primos até 50. Logo, obtemos $n = 2^2 \cdot 7^2 \cdot 43 \cdot 1241143$. Como $2^{1241143-1} \equiv 793958 \pmod{1241143}$, pelo Teorema de Fermat concluímos que 1241143 é composto.

Vamos ver com algum pormenor três métodos sofisticados que ganham vantagens na factorização de números compostos com certas propriedades específicas.

3.3.1 Método de Fermat

O método de *Fermat* consiste em encontrar dois inteiros a e b que permitam representar o número natural n composto como diferença de dois quadrados, ou seja, $n = a^2 - b^2$. Isto permite encontrar uma factorização $n = (a - b)(a + b)$.

Podemos construir um algoritmo para encontrar os inteiros a e b. Dado um inteiro n ímpar começamos por tomar $a = \lfloor \sqrt{n} \rfloor + 1$.

Se $b = \sqrt{a^2 - n}$ é inteiro, então obtemos uma factorização de n como $(a-b)(a+b)$. Caso contrário, incrementamos a de uma unidade até obter $b = \sqrt{a^2 - n}$ inteiro.

Podemos verificar facilmente que o algoritmo usa um número finito de passos. Se n é um inteiro composto ímpar, temos que

$$n = ab = \left(\frac{a+b}{2}\right)^2 - \left(\frac{a-b}{2}\right)^2.$$

Por outro lado, nenhum dos factores de n é par, logo $\frac{a+b}{2}$ e $\frac{a-b}{2}$ são inteiros. Isto significa que este processo pára!

Para tornar este algoritmo mais eficiente, notamos que $(a+1)^2 = a^2 + 2a + 1$. Logo para calcular $(a+1)^2 - n$ basta somar $2a+1$ a cada membro da equação $a^2 - n = c$ e obtém-se

$$(a+1)^2 - n = c + 2a + 1.$$

Exemplo 3.3.2 Vamos factorizar $n = 13221$ usando o método de *Fermat*. Como $\lfloor \sqrt{n} \rfloor = 114$ temos

$$114^2 - n = -225$$
$$115^2 - n = -225 + (2 \cdot 114 + 1) = 4 = 2^2$$

Logo $n = 115^2 - 2^2$ e consequentemente $13221 = (115+2)(115-2) = 117 \cdot 113$.

Este método é eficiente quando aplicado a números compostos grandes?

Suponhamos que pretendemos factorizar um módulo $n = pq$, em que p e q são primos ímpares e $p > q$. A única maneira de escrever n como diferença de quadrados é

$$n = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2,$$

então o número de passos necessários é

$$\left(\frac{p+q}{2}\right) - \lfloor \sqrt{n} \rfloor \approx \frac{(p - \sqrt{n})^2}{2p}.$$

Se considerarmos $p = \lambda\sqrt{n}$, temos que

$$\frac{(p-\sqrt{n})^2}{2p} = \frac{(\lambda - 1)^2}{2\lambda}\sqrt{n}$$

Se $\lambda = 10^3$, isto acontece quando p tem cerca de 6 casas decimais a mais do que q, então

$$\frac{(\lambda-1)^2}{2\lambda} \approx 10^3.$$

Portanto, para usar no RSA devemos gerar primos p e q cujo quociente entre eles seja pelo menos 10^6.

3.3.2 Método $p-1$ de Pollard

Este método de factorização foi inventado por *John Pollard*, método que ganha vantagem para números compostos n com um factor primo p em que $p-1$ só tem "pequenos" divisores primos.

É possível determinar um múltiplo k de $p-1$ sem conhecer o produto de factores primos de $p-1$ nem de n.

Suponhamos que $p-1 \mid k$. Então pelo Teorema de Fermat temos

$$a^k \equiv 1 \pmod{p},$$

para todos os inteiros a primos com p. Isto significa $p \mid (a^k-1)$. Consequentemente, p divide (n, a^k-1). Pode acontecer que (n, a^k-1) seja igual a p ou seja um divisor próprio de n.

O algoritmo de *Pollard* selecciona os candidatos para k, com o produto de todas as potências de primos até um certo limite L, ou seja,

$$k = \prod_{q_i^{e_i} < L} q_i^{e_i}, \quad \text{em que } q_i \text{ é primo e } e_i = \max\{x \in \mathbb{N} : q_i^x < L\}.$$

Se as potências de primos que dividem $p-1$ são todas inferiores a L, então k é um múltiplo de $p-1$. O algoritmo calcula $m = (a^k-1, n)$ para uma base apropriada a. Note que o algoritmo de *Euclides* 1.1.2.2 permite calcular rapidamente m. Se m não é divisor de n, usamos um novo limite L.

Williams encontrou alguns factores primos grandes de alguns números com o método de *Pollard*. Por exemplo, $p = 121450506296081$ é um factor primo de $n = 10^{95}+1$. Note-se que $p-1$ só tem factores primos "pequenos", pois

$$p-1 = 2^4 \cdot 5 \cdot 13 \cdot 19^2 \cdot 15773 \cdot 20509.$$

Exemplo 3.3.3 No exemplo 3.3.1 faltava factorizar o número composto $n = 1241143$. Se usarmos o limite $L = 13$, obtemos $k = 2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13$ e $(2^k-1, n) = 547$. Temos que $p = 547$ é um divisor de n e o seu cofactor é $q = \frac{n}{547} = 2269$. Como 547 e 2269 são primos, concluímos a factorização.

Portanto, quando geramos os primos p e q a usar no sistema de cifragem RSA, devemos certificar que $p-1$ e $q-1$ possuem algum factor primo grande, para evitar que $n = pq$ seja factorizado com o método $p-1$ de *Pollard*.

3.3.3 Crivo quadrático de *Pomerance*

Depois aplicar o método das divisões sucessivas para procurar pequenos divisores de n, certificamos que o n é composto, por exemplo, aplicando o teste de *Miller–Rabin* ou o teste de *Solovay–Strassen*. Esgotado o método de *Fermat* e o método $p-1$ de *Pollard*, resta-nos uma grande ferramenta: o *crivo quadrático*.

O método de factorização de números inteiros *crivo quadrático* (Quadratic Sieve) foi desenvolvido em 1981 por *Carl Pomerance*. É um dos mais eficientes métodos de factorização.

Carl Pomerance, autor do artigo *"A Tale of Two Sieves"*[2] , descreve métodos de factorização de inteiros em Teoria de Números, partindo de exemplos elementares e conduzindo o leitor até aos métodos excelsos, o "Quadratic Sieve" e o "Number Field Sieve", método semelhante desenvolvido por *John Pollard* em 1988.

Nesta secção, vamos essencialmente descrever o funcionamento deste método e explicar como encontrar um divisor próprio de n. De um modo geral, podemos aplicar este algoritmo recursivamente para factorizar completamente o inteiro n.

O método procura dois inteiros x e y tais que

$$x^2 \equiv y^2 \pmod{n} \quad \text{e} \quad x \not\equiv \pm y \pmod{n}. \tag{3.8}$$

Isto implica que n é um divisor de $x^2 - y^2 = (x+y)(x-y)$ mas não é divisor de $(x-y)$ nem de $(x+y)$. Logo, $g = (x-y, n)$ é um divisor próprio de n. Novamente, g pode ser facilmente calculado com o *algoritmo de Euclides* 1.1.2.2.

Exemplo 3.3.4 Sejam $n = 7429$, $x = 227$, $y = 210$. Então $x^2 - y^2 = n$, $x - y = 17$ e $x + y = 437$. Portanto, $g = (x-y, n) = 17$ é um divisor próprio de n.

A ideia de procurar inteiros x e y nas condições 3.8 também é usada por outros algoritmos tais como "Number Field Sieve". Mas os algoritmos diferem no modo de descobrir os inteiros x e y. Vamos mostrar como é que o *crivo quadrático* encontra os inteiros x e y.

Seja

$$m = \lfloor \sqrt{n} \rfloor \quad \text{e} \quad f(X) = (X+m)^2 - n.$$

[2] O artigo está disponível em http://www.ams.org/notices/199612/pomerance.pdf

Primeiro calculamos $f(X_i)$ e escolhemos somente os objectos X_i para o qual $f(X_i)$ só tem factores primos pequenos. De entre a família de congruências

$$(X_i + m)^2 \equiv f(X_i) \pmod{n},$$

escolhemos um subconjunto $\{X_i\}_{i=1\ldots n}$ para o qual o produto dos $f(X_i)$ é um quadrado, ou seja, os expoentes dos factores primos de $\prod f(X_i)$ são pares!

Exemplo 3.3.5 Se $n = 7429$, então $m = \lfloor \sqrt{n} \rfloor = 86$ e $f(X) = (X + 86)^2 - 7429$. Temos

$$\begin{aligned} f(-3) &= 83^2 - 7429 = -540 = -1 \cdot 2^2 \cdot 3^2 \cdot 5 \\ f(1) &= 87^2 - 7429 = 140 = 2^2 \cdot 5 \cdot 7 \\ f(2) &= 88^2 - 7429 = 315 = 3^2 \cdot 5 \cdot 7. \end{aligned}$$

Isto implica que

$$\begin{aligned} 83^2 &\equiv -1 \cdot 2^2 \cdot 3^2 \cdot 5 \pmod{7429} \\ 87^2 &\equiv 2^2 \cdot 5 \cdot 7 \pmod{7429} \\ 88^2 &\equiv 3^2 \cdot 5 \cdot 7 \pmod{7429}. \end{aligned}$$

Se multiplicarmos as duas últimas congruências, obtemos

$$(87 \cdot 88)^2 \equiv (2 \cdot 3 \cdot 5 \cdot 7)^2 \pmod{n}.$$

Encontramos os inteiros $x = 227 \equiv 87 \cdot 88 \pmod{n}$ e $y = 210 \equiv 2 \cdot 3 \cdot 5 \cdot 7 \pmod{n}$.

No exemplo anterior foi fácil encontrar as congruências a multiplicar. Mas se n é grande, é preciso considerar mais factores primos e mais congruências. Como é que podemos seleccionar as congruências apropriadas?

O processo é uma aplicação da Álgebra Linear. Primeiro escolhemos um inteiro positivo L. Procuramos somente os inteiros X tais que $f(X)$ só tem factores primos que pertencem a base de factores

$$F(L) = \{p \mid p \leq L \text{ e } p \text{ primo}\} \cup \{-1\}.$$

A tabela 3.3 dá-nos uma ideia do tamanho da base de factores a considerar. Depois de encontrar tantos elementos X quanto o número de elementos da base de factores, resolvemos o sistema linear correspondente em $\mathbb{Z}_2$. Este sistema pode ser resolvido de várias formas, por exemplo, usando o algoritmo de eliminação de *Gauss*.

Exemplo 3.3.6 Vamos exemplificar o método geral de selecção de congruências para o exemplo 3.3.5. Podemos escolher de entre três congruências. O processo de selecção é controlado pelos coeficientes $\lambda_i \in \{0,1\}$, $1 \leq i \leq 3$. A congruência i é escolhida sómente se $\lambda_i = 1$. O produto das congruências escolhidas pode ser expresso como

$$\begin{aligned}&(-1 \cdot 2^2 \cdot 3^2 \cdot 5)^{\lambda_1} \cdot (2^2 \cdot 5 \cdot 7)^{\lambda_2} \cdot (3^2 \cdot 5 \cdot 7)^{\lambda_3} \\ &= (-1)^{\lambda_1} \cdot 2^{2\lambda_1+2\lambda_2} \cdot 3^{3\lambda_1+2\lambda_3} \cdot 5^{\lambda_1+\lambda_2+\lambda_3} \cdot 7^{\lambda_2+\lambda_3}.\end{aligned}$$

Queremos que este número seja um quadrado, ou seja, que os expoentes de todos os elementos da base de factores sejam pares. Logo, basta resolver o seguinte sistema:

$$\begin{cases} \lambda_1 \equiv 0 \pmod 2 \\ 2\lambda_1 + 2\lambda_2 \equiv 0 \pmod 2 \\ 3\lambda_1 + 2\lambda_3 \equiv 0 \pmod 2 \\ \lambda_1 + \lambda_2 + \lambda_3 \equiv 0 \pmod 2 \\ \lambda_2 + \lambda_3 \equiv 0 \pmod 2 \end{cases}.$$

Como o sistema tem solução $\lambda_1 = 0$, $\lambda_2 = \lambda_3 = 1$. Portanto, escolhemos a segunda e terceira congruências.

Para concluir a descrição do método, falta mostrar como encontrar objectos X_i tal que $f(X_i)$ só tem factores primos que pertencem à base de factores $F(L)$, para um inteiro L fixo.

Uma possibilidade é calcular $f(X)$ para $X = 0, \pm 1, \pm 2, \pm 3, \ldots$, e testar se cada $f(X)$ só tem factores primos que pertencem à base de factores. Para cada $f(X)$ temos efectuar divisões por todos os elementos da base de factores. Isto é ineficiente se n for grande, como se pode ver pelo números de elementos do conjunto de crivação na tabela 3.3!

Dígitos decimais de n	50	60	70	80	90	100	110	120
Base de factores ($\times 1000$)	3	4	7	15	30	51	120	245
Conjunto de crivação ($\times 10^6$)	0.2	2	5	6	8	14	16	26

Tabela 3.3: *Tamanho da base de factores e do intervalo de crivação.*

Outro método é baseado em técnicas de crivação. Apresentamos uma versão simplificada que demonstra a técnica. No entanto, em [35] são descritos métodos que tornam o crivo mais eficiente.

Primeiro fixamos um *conjunto de crivação*

$$C = \{-c, -c+1, \ldots, 0, 1, \ldots, c\}.$$

Queremos encontrar todos os elementos $X \in C$ tais que $f(X)$ só tem factores primos que pertencem à base de factores $F(L)$, para um inteiro L fixo. Para cada primo $p \in F(L)$ dividimos os valores de $f(X)$ pela maior potência possível de p. Os valores de X procurados são exactamente aqueles que, no fim do processo, têm valor 1 ou -1.

Se o conjunto de crivação for grande, podemos rapidamente encontrar os elementos divisíveis por $p \in F(L)$. Primeiro procuramos os zeros X_0 de $f(X)$ módulo p (f tem no máximo 2 zeros, pois f é um polinómio de grau 2). Os outros valores de X para o qual $f(X)$ é divisível por p são os inteiros $X_0 + kp$ que estão no conjunto de crivação, em que k é inteiro. O processo que em cada passo divide os inteiros $f(X_0 + kp)$ por p é denominado de crivação por p.

Exemplo 3.3.7 Como nos exemplos 3.3.5 e 3.3.6, seja $n = 7429$, $m = 86$ e $f(X) = (X + 86)^2 - 7429$. Consideremos a base de factores $\{2, 3, 5, 7\} \cup \{-1\}$ e o conjunto de crivação $\{-3, -2, -1, 0, 1, 2, 3\}$. A tabela 3.4 apresenta o crivo com os primos 2, 3, 5 e 7.

s	-3	-2	-1	0	1	2	3
$(s+m)^2 - n$	-540	-373	-204	-33	140	315	492
Crivo com 2	-135		-51		35		123
Crivo com 3	-5		-17	-11		35	41
Crivo com 5	-1				7	7	
Crivo com 7					1	1	

Tabela 3.4: *O crivo de Pomerance.*

Pela tabela 3.4, é fácil ver que os valores procurados de X são: -3, 1 e 2.

3.4 Primos fortes

Os números primos devem ter ainda propriedades adicionais para assegurar que o sistema de cifragem associado seja resistente a algoritmos que efectuem ataques especializados. O

algoritmo de *Pollard*, ver 3.3.2, procura factores p em que $p-1$ tem factores relativamente "pequenos". Este algoritmo foi generalizado por *Williams* [25], para primos em que $p+1$ tem factores relativamente "pequenos". Para prevenir ataques de estes dois algoritmos surge a noção de *primos fortes.*

Definição 3.4.1 (Primos fortes) *Diz-se que um número primo p é um* primo forte *se existirem inteiros r, s e t tais que:*

(i) $p-1$ *tem um factor primo grande* r,

(ii) $p+1$ *tem um factor primo grande* s,

(iii) $r-1$ *tem um factor primo grande* t.

Acredita-se que os *primos fortes* são pouco mais seguros que os *primos aleatórios*, uma vez que o "tamanho" dos primos usados no cálculo do módulo n, do sistema de cifragem RSA, satisfaz com grande probabilidade todos os requisitos de segurança. Por outro lado, os *primos fortes* não oferecem menos segurança e o trabalho extra exigido para a sua geração é pouco maior do que o dos primos aleatórios.

O seguinte Lema faculta um método para gerar *primos fortes.*

Lema 3.4.1 (Algoritmo de Gordon) *Algoritmo para gerar um primo forte:*

(i) Geramos dois primos aleatórios s e t com tamanho predefinido igual;

(ii) Seja i_0 um inteiro. Calculamos o primeiro primo da sequência $2it+1$, para $i = i_0, i_0+1, i_0+2, \ldots$, seja $s \neq r = 2it+1$;

(iii) Calculamos $p_0 = 2\left(s^{r-2} \pmod r\right) s - 1$;

(iv) Seja j_0 um inteiro. Calculamos o primeiro primo da sequência $p_0 + 2jrs$, para $j = j_0, j_0+1, j_0+2, \ldots$, seja $p = p_0 + 2jrs$.

Demonstração. Suponhamos que $r \neq s$. Tendo em conta que $p_0 \equiv 1 \pmod r$ e que $p_0 \equiv -1 \pmod s$.

i) $p-1 \equiv p_0 + 2jrs - 1 \equiv 1 + 2jrs - 1 \equiv 0 \pmod r$, logo $p-1$ tem um factor grande r;

ii) $p+1 \equiv p_0 + 2jrs + 1 \equiv -1 + 2jrs + 1 \equiv 0 \pmod s$, logo $p+1$ tem um factor grande s;

iii) $r-1 \equiv 2it + 1 - 1 \equiv 0 \pmod t$, logo $r-1$ tem um factor grande t. □

Para decidir a primalidade de s, t, r e p_0 no algoritmo de *Gordon* podemos efectuar divisões para primos menores que um inteiro L e aplicar o teste de *Miller–Rabin* 3.2.4 para um certo número de bases. O tamanho dos primos r, s e t depende da protecção necessária contra ataques específicos. Mas temos de ter cuidado com o tamanho dos primos s e t, e dos parâmetros i_0 e j_0. O tamanho (em bits) dos primos s e t deverá ser cerca de metade de p. O custo computacional para gerar um *primo forte* é cerca de mais 19% do que o custo um primo aleatório [25].

Capítulo 4

Pretty Good Privacy

4.1 Introdução

Uma empresa pode proteger informação recorrendo a vários mecanismos. A informação pode estar disposta em ficheiros numa sala de acesso restrito a pessoal autorizado. Pode controlar fluxo de informação entre duas redes de computadores por meio de uma "firewall" (hardware ou software utilizado para proteger sistemas em rede de utilizadores estranhos ao sistema).

O *PGP v7.0.3* (Pretty Good Privacy) introduz mais um mecanismo de segurança que oferece protecção de informação para computadores individuais. Com o PGP, dispomos de vários utilitários para segurança de mensagens electrónicas, ficheiros, disco rígido e redes de computadores.

O PGP permite realizar várias tarefas como Cifrar/Assinar e Decifrar/Verificar em diversas aplicações de correio electrónico. Estas tarefas são acessíveis a partir de "módulos" (aplicativos do PGP que se inserem nas aplicações de correio electrónico) sob forma de menu – o menu "PGP". Podemos gerar e efectuar a gestão de chaves públicas e privadas; cifrar/decifrar ficheiros e criar ficheiros que se auto-decifram; apagar permanentemente ficheiros, pastas e o espaço livre do disco rígido; e cifrar o tráfego de sistemas em rede (Virtual Private Network).

Em particular, vamos estudar com algum pormenor como utilizar PGP para explorar, na prática, métodos de cifragem de mensagens electrónicas. Para começar a usar o PGP, num computador pessoal com o Microsoft Windows 98 instalado, devemos efectuar os

seguintes passos:

(1) Fazer a transferência do ficheiro de instalação para o seu computador a partir de http://www.pgpi.org/(a página internacional do PGP).

(2) *Proceder à instalação do PGP.*

Após a instalação do PGP aparece um pequeno cadeado na *barra de tarefas* que permite efectuar todas as operações básicas do PGP (PGPtray). O PGP também está acessível no menu *Iniciar* do Windows (Iniciar→ Programas→ P G P) e no próprio *Explorador do Windows* no menu "Ficheiro→ P G P".

(3) *Gerar o par de chaves pública e privada.* É necessário um par de chaves (pública e privada) para cifrar/decifrar informação e assinar/verificar informação.

Temos uma opção que permite gerar um novo par de chaves durante o processo de instalação do PGP, mas podemos gerar um novo par de chaves a qualquer momento executando a aplicação "Iniciar→ Programas→ P G P→ PGPkeys".

(4) *Proceder à troca de chaves.* Uma vez criado o par de chaves, podemos começar a comunicar em segurança com outros utilizadores do PGP. Necessitamos apenas da chave pública dos destinatários. A nossa chave pública é um bloco de texto que pode ser enviada via correio electrónico ou que pode ser publicada num servidor público de chaves dedicado para o efeito, ver figura 1.1.

(5) *Validar chaves públicas.* Uma vez recebida a chave pública do destinatário e adicionada à nossa agenda de chaves públicas, devemos validar a chave certificando que a chave pertence realmente a quem se diz ser. Basta comparar a "impressão digital" da chave recebida com a "impressão digital" da chave original, ver funções unidireccionais, secção 1.4.3. Após certificar que possuímos a chave pública certa, assinamos a chave pública para indicar que confiamos nessa chave.

(6) *Enviar mensagens seguras.* Após ter gerado o par de chaves pública e privada e ter posteriormente efectuado a troca de chaves, podemos finalmente cifrar/decifrar e assinar/verificar mensagens electrónicas. Somente temos de seleccionar a mensagem e escolher a tarefa (cifrar, assinar e decifrar) a partir do menu PGP.

4.2 Criação e gestão de chaves

O PGP assenta em sistemas de cifragem assimétricos, de chave pública (ver Capítulo 1). Isto significa que cada utilizador do PGP tem de gerar uma chave privada e uma pública. Como os nomes indicam, só nós temos acesso à nossa chave privada, mas as chaves públicas são de domínio público.

O PGP usa a nossa chave privada para assinar e a chave pública do destinatário para a cifragem. Reciprocamente, o PGP usa a chave pública do signatário para verificar a assinatura e a chave privada do destinatário para a decifragem.

Logo, a primeira tarefa a realizar, antes de poder usar o PGP para enviar mensagens privadas e assinadas, é gerar uma chave pública e privada. Nesta secção, vamos descrever como gerar chaves públicas e privadas e distribuir chaves públicas para que se possa enviar mensagens electrónicas com privacidade e autenticidade.

Para gerar um novo par de chaves efectuamos os seguintes passos:

(G1) Executamos a aplicação PGPkeys acessível a partir de um dos seguintes caminhos:

- "Iniciar→ Programas→ P G P→ PGPkeys".
- Executar a aplicação PGPtray (o ícone presente na barra de tarefas) e seleccionar PGPkeys.
- Pressionar o botão no módulo do PGP instalado na aplicação de correio electrónico.

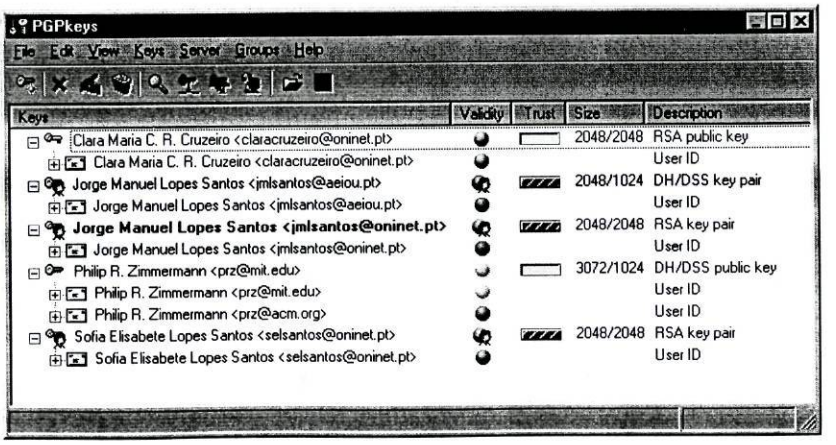


Figura 4.1: *A janela da aplicação PGPkeys.*

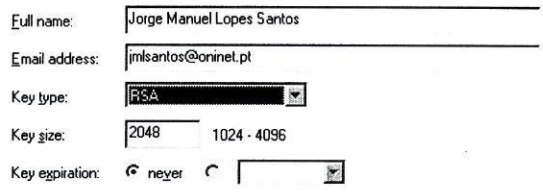
A janela da aplicação PGPkeys, figura 4.1, apresenta as chaves públicas e privadas por nós criadas. Também apresenta as chaves públicas de outros utilizadores por nós adicionadas.

É a partir desta janela, figura 4.1, que se vai efectuar a gestão de chaves.

(G2) Pressione o botão ou execute o menu "Keys→ New keys" para aparecer o assistente de geração de chaves (Key Generation Wizard).

(G3) Vamos pressionar o botão "Expert" para permitir escolher o tipo de algoritmo que vamos usar, o tamanho dos parâmetros e uma data para especificar o termo da chave.

(G4) Introduzimos os dados completando todos os campos



É importante introduzir correctamente o nome e o endereço de correio electrónico para que outros utilizadores nos possam identificar mais facilmente, ou para que possam tirar proveito dos módulos do PGP nas aplicações de correio electrónico.

No campo "Key Type", escolhemos o algoritmo de cifragem e o tamanho dos parâmetros. Temos três algoritmos à escolha:

- Diffie–Hellman/DSS (Digital Signature Standard). Com esta opção podemos adicionar funcionalidades como por exemplo subchaves e fotografias.
- RSA (a nossa opção). Esta opção permite adicionar as mesmas funcionalidades do que a chave Diffie–Hellman/DSS.
- RSA Legacy. Formato de chave RSA compatível com antigas versões do PGP. Este formato de chave não permite ter acesso a todas as funcionalidades dos dois formatos anteriores.

Por fim, escolhemos o tamanho 2048 bits e a data de termo da chave.

(G5) Pressionamos o botão "Next". O PGP pede-nos para introduzir uma "Passphrase".

Uma Passphrase é uma versão mais longa e mais segura da tradicional palavra passe. Pode ser constituída por várias palavras, espaços e sinais de pontuação.

O PGP usa a Passphrase para cifrar a chave privada no nosso computador pessoal. A chave privada é cifrada com um algoritmo simétrico que usa como chave secreta a impressão digital (calculado com uma função unidireccional) da Passphrase.

É necessário introduzir a Passphrase para decifrar a chave privada sempre que se pretende decifrar mensagens (cifradas com a chave pública correspondente).

(G6) Uma vez recolhida toda a informação necessária, começa o processo de geração de chaves, processo estudado no Capítulo 2 e 3.

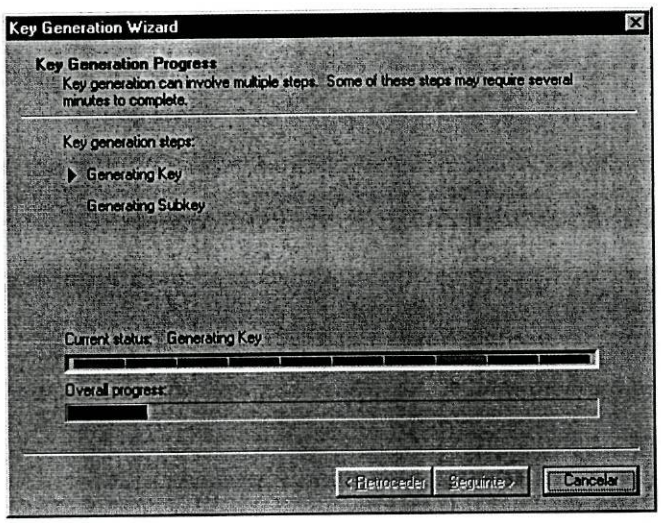


Figura 4.2: *O processo de geração de chaves no PGP.*

Para gerar as chaves pública e privada para o sistema de cifragem RSA (a nossa opção) é necessário gerar grandes quantidades de números aleatórios. Caso o PGP não obtenha entropia suficiente para a semente inicial do gerador de números aleatórios criptograficamente seguro, surge a caixa de diálogo PGP Random Data. Esta caixa de diálogo pede que o utilizador movimente o rato e pressione teclas até completar uma barra de progresso.

(G7) Pressione "Finish". O PGP automaticamente adiciona a nossa chave pública e privada à nossa agenda de chaves públicas e privadas.

(G8) Colocamos a nossa chave pública num servidor de chaves públicas executando o menu: "Server→ Send to→ Domain Server". O programa envia a chave pública para os servidores públicos de chaves:

- idap://europe.keys.pgp.com:11370
- idap://keyserver.pgp.com
- http://pgpkeys.mit.edu:11371

Como a nossa chave pública é um pedaço de texto, ela pode ser facilmente enviada por meio de mensagens de correio electrónico, ou disponibilizada em páginas de internet. Para isso, exportamos a chave pública para um ficheiro de extensão "asc" executando: "Keys→ Export..." e especificando o corpo do ficheiro.

4.3 Protecção e assinatura de mensagens electrónicas

Enviar uma mensagem electrónica não cifrada é como enviar um postal: a mensagem enviada pode ser facilmente visualizada por terceiros! O PGP possibilita enviar mensagens electrónicas seguras por meio de módulos que se inserem na aplicação de correio electrónico. Também é possível assinar digitalmente mensagens para garantir a sua autenticidade e integridade, ver 1.4.

4.3.1 Como funciona o PGP?

O PGP combina os métodos de cifragem tradicionais com o método de chave pública, ver o Capítulo 1. Trata-se de um sistema de cifragem híbrido!

Suponhamos que pretendemos cifrar uma mensagem com o PGP. Primeiro, o PGP comprime a mensagem com o popular algoritmo ZIP. A compressão de dados permite reduzir a quantidade de informação poupando tempo na transmissão de dados e espaço em disco.

O PGP cria uma chave de sessão, uma chave secreta que só será usada uma vez. Esta chave é criada por um gerador de números aleatórios criptograficamente seguro, ver Capítulo 2. Em cada utilização, adiciona-se entropia à semente a partir de várias fontes, por exemplo, dispositivos de entrada como o rato e o teclado. A chave de sessão é utilizada

em algoritmos de cifragem simétricos seguros e muito rápidos, por exemplo TripleDES ou CAST. O resultado é o texto cifrado.

Uma vez cifrada a mensagem, a chave de sessão é cifrada com a chave pública do destinatário, por exemplo com o sistema de cifragem RSA. O resultado final a transmitir ao destinatário é um "pacote" com a mensagem cifrada por um método simétrico ("tradicional") e com uma chave de sessão cifrada com a chave pública do destinatário.

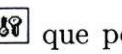
Para assinatura de mensagens, o PGP usa uma função unidireccional criptograficamente segura, por exemplo SHA-1. Com esta função unidireccional, o PGP gera uma impressão digital da mensagem de tamanho fixo, ver 1.4.3. Seguidamente, o PGP aplica a chave privada à impressão digital produzida pela função unidireccional para originar a "assinatura".

Para proceder à verificação da assinatura, basta aplicar a chave pública do signatário à assinatura e verificar se coincide com a impressão digital da mensagem.

Enquanto se usar uma função unidireccional criptograficamente segura, não há possibilidade de aplicar uma assinatura a outro documento diferente, ou de alterar um documento assinado mantendo a assinatura válida. A mais pequena alteração causa a falha do processo de verificação.

4.3.2 O PGP integrado no Outlook

Ao instalar o PGP ficam automaticamente disponibilizados os módulos para o Microsoft Exchange, Outlook, Express e QUALCOMM Eudora.

Por exemplo, o módulo do PGP para o Microsoft Outlook consiste de um menu e de três botões  que permitem várias tarefas. O ícone "*envelope com cadeado*" permite cifrar a mensagem ao enviar; o ícone "*papel com uma pena*" permite assinar a mensagem ao enviar; e, por fim, o ícone "*duas chaves*" executa a aplicação PGPkeys que permite efectuar a gestão de chaves. Analogamente, o menu possui, entre outras, as mesmas tarefas que os botões denominadas "Encrypt on send", "Sign on send" e "Lauch PGPkeys".

Enviar mensagens

Para enviar mensagem cifradas e assinadas com o Microsoft Outlook, efectuamos os seguintes passos:

(E1) Usamos o Outlook para escrever a nossa mensagem normalmente. Devemos ter o cuidado de seleccionar um texto para o "assunto" da mensagem que não revele o conteúdo do corpo da mensagem.

(E2) Uma vez finalizada a mensagem, seleccionamos o ícone para que a mensagem seja cifrada, e seleccionamos o ícone para que a mensagem seja assinada.

(E3) Enviamos a mensagem, como habitualmente, pressionando no botão *Enviar*.

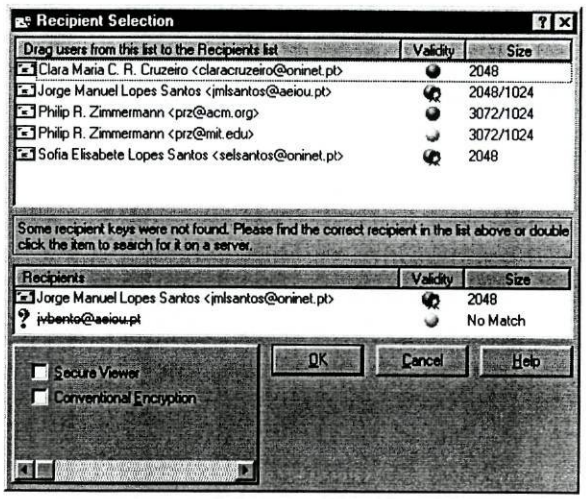


Figura 4.3: *Lista de chaves públicas e privadas.*

Caso existam na agenda, as chaves apropriadas são automaticamente usadas. Caso contrário, se o PGP não conhecer a chave pública do destinatário ou se a chave pública não é válida, o PGP efectua automaticamente uma busca em algum servidor de chaves. Seguidamente, surge uma janela com a lista de chaves públicas da agenda para que o utilizador possa adicionar uma chave encontrada no servidor público de chaves, ou que possa escolher a chave correcta associada ao destinatário. Podemos forçar o aparecimento da janela com a lista de chaves públicas constantes na agenda pressionando na tecla *Shift* quando pressionamos no botão *Enviar*.

(E4) Podemos escolher duas opções, dependendo do tipo de informação a ser cifrada, ver figura 4.3.

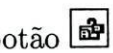
- *Secure viewer.* Esta opção permite o uso de um tipo de letra especialmente desenhada para que não seja legível por meio de equipamento de captura de radiações.
- *Conventional Encrypt.* Esta opção permite que o PGP use métodos de cifragem de chave secreta (simétrica) no lugar de métodos de cifragem de chave pública (assimétrica).

(E5) Pressionamos o botão *Ok* para cifrar e assinar a mensagem. Quando se pretende assinar uma mensagem, a chave privada correspondente à chave pública é necessária. Por isso, aparece uma caixa de diálogo para que o utilizador introduza a sua *Passphrase* para decifrar a chave privada que vai permitir assinar a mensagem.

Receber mensagens

Ao receber uma mensagem protegida temos de decifrar e/ou verificar a assinatura da mensagem. Para isso, seguimos os seguintes passos:

(R1) Abrimos a mensagem electrónica como habitualmente.

(R2) Pressionamos o botão  para decifrar e verificar mensagens.

O PGP verifica automaticamente uma mensagem assinada, pois apenas necessita da chave pública do remetente que está disponível em servidores públicos de chaves. Se

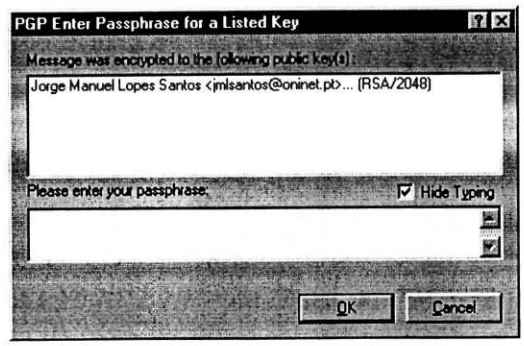


Figura 4.4: *Caixa de diálogo "Passphrase".*

a mensagem recebida foi cifrada com a chave pública do destinatário, necessitamos

da chave privada correspondente. Neste caso, aparece uma caixa de diálogo, figura 4.4, que pede a introdução da Passphrase permitindo reaver a chave privada (cifrada com um algoritmo simétrico cuja chave é a impressão digital da Passphrase).

(R3) Introduzimos a Passphrase.

A mensagem está agora decifrada e, caso tenha sido assinada, aparece uma indicação da validade da mensagem, do seu signatário, data da assinatura e da data da sua verificação.

Se o emissor seleccionou a opção "Secure Viewer" aparece uma mensagem de advertência. Só é permitido a leitura da mensagem num visualizador seguro que usa um tipo de letra especial para prevenir ataques TEMPEST.

(R4) Podemos guardar a mensagem na forma decifrada ou manter a mensagem original para que continue secreta.

Exemplo 4.3.1 A Sofia vai enviar a seguinte mensagem ao Jorge:

> Jorge, vende rapidamente todas as nossas acções da empresa ABC, descobri que amanhã elas vão descer muito! Por outro lado, parece que a empresa DEF é uma boa aposta para investimento. Ninguém pode saber! Sofia.

É obvio que o conteúdo da mensagem deve ser mantido secreto. Também é necessário que o Jorge tenha a certeza de que a mensagem foi realmente enviada pela Sofia. Seguindo os passos E descritos nesta secção, a mensagem cifrada e assinada pela Sofia é:

```
-----BEGIN PGP MESSAGE-----
Version: PGPfreeware 7.0.3 for non-commercial use <http://www.pgp.com>

hQEMA0W6FA0J9064AQgAmWdxvSIsh+IUxGu5tUtDXfK3NljMb9KqGDD3iwA6mC7F
oeifLt7npRJDtS+Cf5JDQulmUcoh8s0QMCCuRI6cPRF0aU4EphKiT3nPPCDozGMS
yQ5t+ITT6hwwsAbgROdT5GQWIIKoXdY/RU30W+Of2ZyaFPTkaY9/mte3qIA659HL
XuXS9y/1pJ3PCmqU0W5WQSbc/W8g7eHbOmkLtmAJP3Qz146YSsfeguCfvRhZOpax
L6QeAjAjqD1+C5KZzL7LS/5TjQzovAXcAMTWs06HW8JlSwM2q1ioCoB+PNQ6MngZ
yNW3rHCVlXa1n0NZLaQ7vVS7lSTO0lsX2wCiOOGstqUB/nmW9HTfYPfCHxFTmPoz
0lZ5Sc5XTcxKIvMpsuqOWS0Zy1nZ+YSCPIWPbPdLU9wMBRg1+Ehzy8K6s4v81jBl
PfkA6in5LZtvpeIn04X016Jtd+QIUp/qp1AAGtNsn6vpHlsCUW/3st4MI98asQHY
JRd5tcPLFNn7DVrdRuGkjYO9Ply1EvHJe7q2TS17h3LaqQhTobNXgNKq1on9/Fg2
```

```
wHCrUwr8ESoSZhOMGRwXM9rg3vuiBRcONJ/94t/k82JwovC8WJ/vXZcqALjO8MbB
pSQfrBBz96kDEjOdXhSDEEiJuCp6zuVMA3zV+mIVx9EXl3u72L7fmjIpr6AOYLOi
e1z7OCq+Ft6TRRYgtfKzGPmvp2bvE2mNPyLzuoh6hVvTn809ycvqmqyFgzKizMW2
lZqLGUtHsCkKyxJMS1mqyODSIQBEEIy1tA2HhY2yJnyJg7xY9Cs++SOiztgzyCZR
yW8kjALSrOEFRZJR+DK/etGNEQyhLI3vK9HKrgRE+6jo4zfLDgSVkRKu7P5egvvQ
k7wmyKOlJj680nRkt4NAWMesopZeOyNFmVsn376XvxyhOeRv+4854nfODq1KSjRm
3v3eb7X7n3/jwdSitWz3wDRa/b7ueRgCQwPvREKNdu/wz32VhQMbOr+wUlU1232h
IPZmORB7wphPIeuv/FGu1g==
=GyVp
-----END PGP MESSAGE-----
```

O Jorge recebe a mensagem cifrada e assinada. Seguindo os passos R descritos nesta secção obtém-se:

```
*** PGP Signature Status: good
*** Signer: Sofia Elisabete Lopes Santos <selsantos@oninet.pt>
*** Signed: 18-07-2002 21:26:35
*** Verified: 18-07-2002 21:34:54
*** BEGIN PGP DECRYPTED/VERIFIED MESSAGE ***

Jorge, vende rapidamente todas as nossas acções da empresa ABC, descobri
que amanhã elas vão descer muito! Por outro lado, parece que a empresa DEF
é uma boa aposta para investimento. Ninguém pode saber! Sofia.

*** END PGP DECRYPTED/VERIFIED MESSAGE ***
```

4.3.3 O PGP e a Cifragem/Assinatura em geral

Se não usarmos uma aplicação com módulo PGP instalado, podemos usar o PGP para cifrar/decifrar e/ou assinar blocos de texto seleccionados em aplicações como o Microsoft Word ou simplesmente o Notepad, e também para cifrar/decifrar e/ou assinar ficheiros.

Cifragem de blocos de texto

As funções do menu "Current Window" do PGPtray, na barra de tarefas, ver figura 4.5, permitem cifrar/decifrar e/ou assinar um bloco de texto seleccionado em qualquer

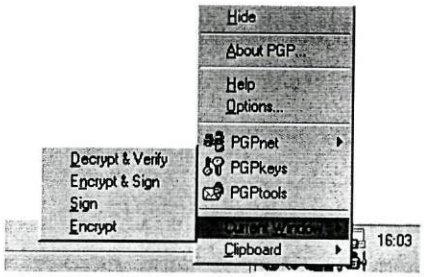


Figura 4.5: *O PGP e a cifragem/assinatura em geral.*

aplicação. Seleccionamos "Encrypt" para cifrar a mensagem, "Sign" para assinar a mensagem e "Encrypt & Sign" para cifrar e assinar a mensagem. Para decifrar e/ou verificar mensagens seleccionamos "Decrypt & Verify".

Cifragem de ficheiros

O PGP possibilita a cifragem de ficheiros a anexar a mensagens de correio electrónico. Para isso, usamos as funções do menu "Ficheiro→ PGP" do Explorador do Windows, ver figura 4.6.

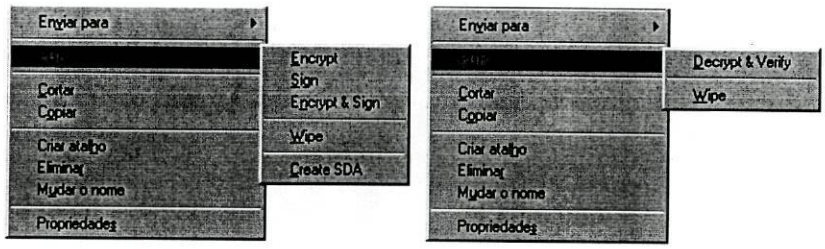


Figura 4.6: *O PGP e a cifragem/assinatura de ficheiros.*

Seleccionamos "Encrypt" para cifrar o ficheiro, "Sign" para assinar e "Encrypt & Sign" para cifrar e assinar o ficheiro. Para decifrar e/ou verificar ficheiros, seleccionamos "Decrypt & Verify".

Podemos seleccionar a função "Create SDA" (Self Decrypting Archive). Neste caso, o resultado é um ficheiro executável e cifrado por um método simétrico ("tradicional") usando uma chave de sessão criada a partir de uma Passphrase introduzida pelo utilizador. O ficheiro executável pode ser decifrado ao ser executado, seguido da introdução correcta da Passphrase.

Bibliografia

[1] N. Ankeny. The least quadratic non residue. *Ann. Math.*, 2(55):65–72, 1952.

[2] E. Bach. *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms.* PhD thesis, Mit Press, Cambriedge, 1985. ACM Distinguished Dissertations.

[3] D. Bleichenbacher. *Efficiency and Security of Criptosystems Based on Number Theory.* PhD thesis, Swiss Federal Institute of Technology Zürich, Zürich, 1996.

[4] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo–random bits. *SIAM J. Computing*, 13(4):850–863, November 1984.

[5] G. Brassard. *Modern Cryptology. A Tutorial.* Springer-Verlag, New York, 1988.

[6] D. Bressoud. *Factorization and Primality Testing.* Springer, New York, 1989.

[7] A. J. Buchmann. *Introduction to cryptography.* Undergraduate Texts in Mathematics, Springer, New York, 2001.

[8] N. Courtois. *La securité des primitives cryptographiques basées sur de problemes algébriques multivariables: MQ, IP, MinRank, HFE.* PhD thesis, Université de Paris 6 - Pierre et Marie Curie, 2001.

[9] R. Crandall and C. Pomerance. *Prime Numbers a Computacional Perspective.* Springer, New York, 2001.

[10] W. Diffie. The first ten years of public-key cryptography. *Proceeding of the IEEE*, 76(5):560–576, May 1988.

[11] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inform.*, IT–22(6):644–654, 1976.

[12] J. Ellis. The possibility of secure non-secret digital encription. *CESG Report*, January 1970.

[13] P. Erdös. On almost primes. *American Monthly*, 57:404–407, 1950.

[14] R. Herken. *The Universal Turing Machine. A Half Century Survey.* Springer-Verlag, Wien, second edition, 1995.

[15] G. Jaeschke. On strong pseudoprimes to several bases. *Math. Comput.*, 61:915–926, 1993.

[16] D. E. Joyce. Euclid's Elements. `http://aleph0.clarku.edu/~djoyce/java/elements/elements.html`, 1998.

[17] D. Kahn. *The Code-Breakers; The Comprehensive History of Secret Communication from Ancient Times to the Internet.* 1996.

[18] D. Knuth. *The art of Computer Programming: Seminumerical Algorithms*, volume II. Addison-Wesley, 1981.

[19] N. Koblitz. *A Course in Number Theory and Cryptography.* Springer-Verlag, New York, 1987.

[20] N. Koblitz. *Algebraic Aspects of Cryptography.* Springer-Verlag, Berlin, 1998.

[21] E. Kranakis. *Primality and Cryptography.* John Wiley & Sons - B. G. Teubner, Chinchester, 1986.

[22] M. Lemos. *Criptografia, Números Primos e Algoritmos.* IMPA, Rio de Janeiro, Brasil, Abril 1989. 17º Colóquio Brasileiro de Matemática.

[23] H. Lenstra. Miller's primality test. *Information Processing Letters*, 8(2):89, 1979.

[24] H. W. Lenstra. Integer programming with a fixed number of variables. *University of Amsterdam*, TR 81–03, April 1981.

[25] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Bocaraton, fifth edition, August 1997.

[26] G. Miller. Riemann hypothesis and tests of primality. *Journal of computer and system sciences*, 13:300–317, 1976.

[27] J. Morgado. Teoria elementar dos números. Departamento de Matemática Pura da Universidade do Porto.

[28] V. Neves. Introdução à teoria de números. Departamento de Matemática da Universidade de Aveiro, Maio 2001.

[29] J. von Newmann. Various thecniques for use in connection with random digits. *Von Newmann Collected Works*, pages 768–770, 1963.

[30] J. Plumstead. Inferring a sequence generated by a linear congruence. *Proceedings 23rd IEEE FOCS*, pages 153–159, 1982.

[31] C. Pomerance. Recent developments in primality testing. *The Mathematical Intelligencer*, 3:97–105, 1981.

[32] C. Pomerance. *Cryptology and Computational Number Theory*, volume 42. AMS, Providence, 1990. Procedings of symposia in applied mathematics.

[33] C. Pomerance. A tale of two sieves. `http://www.ams.org/notices/199612/pomerance.pdf`, 1998.

[34] C. Pomerance, J. Selfriedge, and S. Wagstaff. The pseudoprimes to $25 \cdot 10^9$. *Math. Comput.*, 35:1003–1026, 1980.

[35] H. Riesel. *Prime Numbers and Computer Methods for Factorization.* Birkhäuser, Boston, 1994.

[36] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public–key cryptosystems. Technical Report 82, MIT/LCS/TM, 1977.

[37] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, New York, second edition, 1996.

[38] M. Schroeder. *Number Theory in Science and Communication.* Springer-Verlag, Berlin, 1986.

[39] A. Shamir. On the generation of criptographically strong pseudo–random sequences. *Springer*, 544–550, 1981.

[40] A. Shamir. A polynomial time algorithm for breaking the basic merkle–hellman cryptosystem. *23rd IEEE FOCS*, 145–152, 1982.

[41] C. E. Shannon. A mathematical theory of comunication. *Bell Syst. Thec. J.*, 27:379–423 and 623–656, 1948.

[42] R. Solovay and V. Strassen. A fast Monte Carlo test for primality. *SIAM J. Comp.*, 6:84–85, 1977.

[43] D. Stinson. *Cryptography : Theory and Practice.* CRC Press, Boca Raton, 1995.

[44] A. Yao. Theory and application of trapdoor functions. *Proceedings of the 23rd IEEE*, pages 80–91, IEEE 1982.