Inês Barbedo

O Sistema Criptográfico RSA: Ataques e Variantes

Departamento de Matemática Pura
Faculdade de Ciências da Universidade do Porto
Setembro 2003

TM

Inês Barbedo

O Sistema Criptográfico RSA: Ataques e Variantes

Tese submetida à Faculdade de Ciências da Universidade do Porto para obtenção do grau de Mestre em Matemática - Fundamentos e Aplicações

Departamento de Matemática Pura
Faculdade de Ciências da Universidade do Porto
Setembro 2003

Dissertação realizada sob supervisão do

Prof. Doutor António Machiavelo,

Professor Auxiliar do

Departamento de Matemática Pura da

Faculdade de Ciências da Universidade do Porto

Agradecimentos

Agradeço em especial ao Professor Machiavelo por ter aceite orientar este trabalho e pela constante disponibilidade. À minha escola, ESTGM-IPB, o tempo tão precioso. Aos meus pais e irmão o constante apoio, incentivo e paciência. Àqueles que, por fazerem parte da minha vida, estiveram sempre por perto.

Introdução

Criptografia é a ciência de manter secretos os segredos.

Existem dois tipos de sistemas criptográficos: os *simétricos* e os *assimétricos*. Os sistemas criptográficos simétricos usam uma mesma chave, a *chave secreta*, para encriptar e desencriptar mensagens enquanto que os sistemas criptográficos assimétricos usam um par de chaves: a *chave pública* do conhecimento de todos para encriptar a mensagem e a *chave privada* só do conhecimento do receptor da mensagem que serve para desencriptar. Por isso, os sistemas criptográficos simétricos também se chamam *sistemas criptográficos de chave secreta*, e os assimétricos *sistemas de chave pública*.

Os sistemas criptográficos de chave simétrica têm um problema: como transmitir a chave secreta ao emissor? Se a conseguimos transmitir de forma segura então não precisamos do sistema criptográfico em questão! Basta usar a forma segura usada para transmitir a chave! A necessidade do receptor e do emissor partilharem informação secreta não segura é eliminada nos sistemas criptográficos de chave pública, pois é o receptor que gera o par de chaves.

A ideia de sistema criptográfico de chave pública é introduzida em 1976 por Diffie e Hellman [DH76]. É uma ideia simples mas que vem revolucionar a criptografia. Quem quiser comunicar de forma segura, encripta a mensagem com a chave pública do receptor e só ele consegue desencriptá-la com a respectiva chave privada, que não precisa nem deve ser partilhada com ninguém. Para além de estabelecer comunicações seguras, os sistemas criptográficos de chave pública têm por objectivo autenticar através de uma assinatura digital a informação transmitida.

Apesar de terem sido Diffie e Hellman a introduzir o conceito, não apresentaram nenhum exemplo. São Rivest, Shamir e Adleman que, em Fevereiro de 1978 [RSA78], apresentam o primeiro sistema criptográfico de chave pública: *o sistema criptográfico RSA*.

O sistema criptográfico RSA utiliza resultados da Teoria dos Números como o Teorema de Euler e a resolução de congruências módulo um número N igual ao produto de dois números primos, e baseia-se no princípio de que actualmente é fácil multiplicar números primos, mas pode ser difícil factorizar um número. As suas aplicações são muitas no mundo electrónico como, por exemplo, garantir a privacidade e autenticidade do correio electrónico, a segurança do comércio electrónico ou do acesso a contas bancárias. Com o crescente número de utilizadores e o desenvolvimento dos computadores utilizados, crescem também as tentativas de, pessoas estranhas aos sistemas, descobrir palavras passe, quebrar códigos ou interceptar comunicações, o que obriga a que os gestores de sistemas criptográficos estejam constantemente atentos, actualizem as suas chaves e desenvolvam métodos para garantirem a eficácia desses sistemas.

O objectivo deste trabalho é fazer uma abordagem ao sistema RSA, o sistema criptográfico de chave pública mais utilizado nos dias de hoje.

Começamos por introduzir alguns tópicos da teoria da complexidade no capítulo 1, teoria esta que nos permite estimar a eficiência de algoritmos em termos de tempo de execução e recursos necessários. É introduzida a notação do "Big-O" e as suas propriedades. São descritos o algoritmo de Euclides e um algoritmo para a exponenciação modular para os quais se determina a complexidade. Este capítulo será importante para os que se seguem, pois permite ter um modo de estimar o tempo de execução de algoritmos.

No capítulo 2 introduz-se o sistema criptográfico RSA, descrevendo como são geradas as chaves pública e privada, como se encripta e desencripta mensagens e como são geradas as assinaturas digitais imprescindíveis à autenticação. Inclui-se também um

resultado que mostra que se duas pessoas têm chaves públicas diferentes com o mesmo módulo N, facilmente determinam a factorização de N e portanto a chave privada. Este resultado é importante pois alerta para um erro cometido por alguns, como por exemplo, pelos próprios criadores do RSA, Rivest, Shamir e Adleman ao propôr um exemplo lúdico de aplicação do sistema criptográfico RSA: o "Mental Poker" [Wad81].

Segue-se, no capítulo 3, a descrição de alguns ataques ao sistema RSA. Começa-se por dar um exemplo de um ataque no caso em que se usa um expoente privado pequeno. Enuncia-se o Teorema de Coppersmith que permite determinar de forma eficiente raízes inteiras pequenas de polinómios de coeficientes inteiros módulo N. Este teorema é utilizado em vários ataques ao sistema criptográfico RSA de que iremos apresentar dois exemplos.

No capítulo 4 são introduzidas três variantes do sistema RSA original. A primeira variante, denomidada "Batch" RSA, propõe-se fazer várias desencriptações pelo custo de uma, tornando assim o processo de desencriptação mais rápido. A segunda variante, RSA com múltiplos factores, propõe uma mudança estrutural: mudar a estrutura do módulo para o produto de vários factores primos em vez de dois ou manter os dois primos mas, sendo um deles elevado a uma potência maior que 1. Esta variante que altera a estrutura do módulo tem por objectivo não só acelerar os processos de encriptação e desencriptação mas também proteger o sistema de alguns ataques. A terceira e última variante pretende tornar o processo mais rápido, reequilibrando o esforço efectuado pela encriptação e pela desencriptação, e proteger o sistema de ataques a expoentes públicos pequenos.

Este trabalho termina com dois apêndices. O primeiro, o apêndice A, descreve uma forma de formatar as mensagens a que se vai aplicar o sistema criptográfico RSA. O apêndice B descreve explicitamente o inverso do isomorfismo canónico de $\mathbb{Z}_p^* \times \mathbb{Z}_q^* \rightarrow \mathbb{Z}_N^*$ que será usado no trabalho, e enuncia e prova o Teorema Chinês dos Restos, teorema muito presente em todo este trabalho.

Conteúdo

Capítulo 1

Complexidade

Neste capítulo iremos abordar alguns tópicos da teoria da complexidade, teoria esta que nos permite estimar a eficiência de algoritmos em termos de tempo de execução e recursos necessários. É introduzida a notação do "Big-O" e as suas propriedades. São descritos o algoritmo de Euclides e um algoritmo para a exponenciação modular para os quais se determina a complexidade. Este capítulo será importante para os que se seguem pois introduz uma maneira de estimar o tempo de execução de um algoritmo.

1.1 Tempo de execução de um algoritmo

Quando se pensa num algoritmo não basta que ele resolva um problema. Aliás, existem normalmente vários algoritmos para resolver um mesmo problema. Um algoritmo será melhor ou mais eficiente que outro se utilizar menos recursos, tempo de execução e espaço de memória, para resolver o mesmo problema. Para poder comparar dois ou mais algoritmos que resolvem o mesmo problema e poder optar pelo melhor, deverá existir alguma forma objectiva de os analisar.

A *complexidade* de um algoritmo é uma estimativa do tempo necessário à execução desse algoritmo expressa em função das operações fundamentais e do tamanho dos

dados de entrada (*input*). As operações fundamentais são aquelas que aparecem em cada passo de um algoritmo como, por exemplo, a adição, multiplicação, subtracção ou divisão de números.

Um inteiro n escrito numa base b é representado na forma $(d_{k-1}d_{k-2}\cdots d_1d_0)_b$ onde os d´s são os *dígitos*, i.é, números entre 0 e $b-1$ se $b<10$ e as letras ordenadas do alfabeto se $b \geq 10$. Assim diz-se que $n = d_{k-1}b^{k-1} + d_{k-2}b^{k-2} + \cdots + d_1b + d_0$, com $d_{k-1} \neq 0$, tem *k-dígitos* ou *tamanho* k na base b. Portanto um número n que satisfaz $b^{k-1} \leq n < b^k$ tem k-dígitos na base b, ou seja[1],

$$k = \lfloor \log_b n \rfloor + 1 = \left\lfloor \frac{\log n}{\log b} \right\rfloor + 1,$$

dá o número de dígitos na base b. Podemos omitir a referência à base b sempre que se estiver na base usual decimal ($b = 10$), ou quando esta se subentenda do contexto. Por exemplo, no caso binário ($b = 2$) chamaremos bits aos dígitos, introduzindo assim no texto a base binária em que se está a trabalhar.

Vamos agora ver como funcionam algumas operações em binário para em seguida definir o que é uma operação elementar, determinar a complexidade de um algoritmo, estimando o seu tempo de execução.

Exemplo 1.1.1. Adição de dois números em binário

$$\begin{array}{cccccccc} & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ + & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

Suponhamos que estamos a adicionar dois números com no máximo k-bits cada. Se algum dos números tiver menos bits que o outro serão acrescentados 0´s à esquerda do número com menos bits ficando ambos com o mesmo tamanho. Analisemos com detalhe como se procede para efectuar a adição. O que se faz é repetir k vezes os seguintes passos:

[1] $\lfloor x \rfloor$ denota o maior inteiro menor ou igual a x e $\lceil x \rceil$ o menor inteiro maior ou igual a x

1. Olhar para o bit de cima e para o bit de baixo verificando ainda se "veio 1" (*carry*) da operação anterior.

2. Se os bits são ambos 0 e não "veio 1", então escrever 0 no resultado e continuar.

3. Se ou (a) os bits são ambos 0 e "veio 1", ou (b) um dos bits é 0 e o outro 1, e não "veio 1", então escrever 1 no resultado e continuar.

4. Se ou (a) um dos bits é 1 e o outro 0 e "veio 1" ou (b) ambos os bits são 1 e não "veio 1", então escrever 0 no resultado, colocar "vai 1" na coluna à esquerda e continuar.

5. Se ambos os bits são 1 e existe "veio 1", então escrever 1 no resultado, colocar "vai 1" na coluna à esquerda e continuar.

Aplicar estes procedimentos uma vez diz-se aplicar uma *operação elementar (bit operation)*. Adicionar dois números com k-bits requer assim k operações elementares.

Todas as tarefas, mesmo algumas bem complicadas, podem ser reduzidas a operações elementares. Portanto, estimar o tempo que um computador leva a executar um algoritmo será proporcional ao número de operações elementares a executar. O tempo que uma operação elementar leva a ser executada depende da velocidade de processamento do computador que está a ser utilizado, sendo nos dias de hoje da ordem dos 10^{-9} segundos (1 nanosegundo). Quando se fala em estimar o "tempo" de um algoritmo o que de facto queremos dizer é que estamos a estimar o número de operações elementares que o algoritmo executa.

O tempo necessário à adição de dois números é assim igual ao maior dos tamanhos dos números, o que podemos escrever da seguinte forma

$$\text{Tempo estimado para}(k\text{-bit}+l\text{-bit}) = \max(k, l).$$

Vamos agora estimar o tempo necessário ao produto de dois números binários um com k-bits e outro com l-bits.

Exemplo 1.1.2. Multiplicação de dois números escritos em binário.

$$
\begin{array}{cccccccccc}
 & & & & 1 & 1 & 1 & 0 & 1 \\
 & & & & \times & 1 & 1 & 0 & 1 \\
\hline
 & & & & 1 & 1 & 1 & 0 & 1 \\
 & & & 1 & 1 & 1 & 0 & 1 & 0 & \\
 & & 1 & 1 & 1 & 0 & 1 & & & \\
\hline
1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 &
\end{array}
$$

Suponhamos que é usado o procedimento seguido no exemplo anterior para multiplicar o número n com k-bits pelo número m com l-bits e vamos assumir, sem perda de generalidade, que $k \geq l$. O que se obtém são no máximo l linhas, pois por cada bit igual a 0 em m escreve-se menos uma linha, e onde cada linha é uma cópia de n deslocada um dígito para a esquerda, i.é, acrescida de um 0 no extremo direito. Assim em cada linha teremos inteiros com no máximo $(k+l)$-bits.

$$
\begin{array}{rl}
 & k\text{-bits } (n) \\
\times & l\text{-bits } (m) \\
\hline
 & \cdots \qquad \Big\} \leq l \text{ linhas} \\
 & \underbrace{\qquad\qquad} \\
 & \leq k+l\text{-bits}
\end{array}
$$

Com vista a contar as operações elementares envolvidas nesta multiplicação, vamos supôr que é apenas efectuada uma adição de cada vez, ou seja, adicionamos a primeira linha com a segunda, depois adicionamos a terceira linha com o resultado obtido da adição das duas primeiras linhas e por diante até não haver mais linhas. Teremos assim que efectuar $l-1$ adições. Como em cada adição são executadas no máximo $k+l$ operações elementares o número total dessas operações será, no máximo

$$
\begin{aligned}
&\text{Tempo estimado para } (k\text{-bits} \times l\text{-bits}) \leq \\
&\qquad \leq (l \text{ adições}) \times (k+l \text{ op. elementares por adição}) \\
&\qquad \leq l(k+l) \leq 2kl \text{ , porque } l \leq k.
\end{aligned}
$$

A estimativa de tempo que é feita em termos de operações elementares dá-nos um limite superior para o número de operações elementares necessárias ao algoritmo, procurando em geral obter uma estimativa simples e fácil de se trabalhar. Por exemplo, na multiplicação o número de linhas diferentes de zero a considerar para as adições será menor que $l-1$ e se l tiver muitos zeros essa diferença será significativa. Ou ainda, poderiamos ter usado uma técnica um pouco mais complicada que aquela aqui usada para multiplicar dois números. Prova-se que (ver [Rie85, p. 348-356]), usando outras técnicas para multiplicar dois números com k-bits, o tempo estimado é proporcional a $(k \log k \log\log k)$ operações elementares, para k suficientemente grande.

1.2 A notação do "Big O"

Definição 1.2.1. *Sejam $T(n)$ e $f(n)$ funções reais positivas de variável inteira positiva. Se existem uma constante C e uma constante n_0 tais que $T(n) \leq Cf(n)$ para todo $n \geq n_0$, então escreve-se $T(n) = O(f(n))$ e diz-se que T tem complexidade dominada por $O(f(n))$.*

Esta definição extende-se de forma óbvia ao caso de T e f serem funções com mais que uma variável.

Na prática é muito difícil prever com algum rigor o tempo de execução, $T(n)$, de um algoritmo. O que a notação do "big-O" introduz é uma análise assimptótica ($n \to \infty$) e uma majoração dessa estimativa. Note-se que a igualdade $T(n) = O(f(n))$ não é verdadeiramente uma equação porque $O(f(n))$ não representa uma só função $f(n)$ mas sim um conjunto de funções tais que $T(n) \leq Cf(n)$, para alguma constante C. Assim, o sinal "=" nesta notação significa efectivamente inclusão, "$\subseteq$".

Voltemos aos nossos exemplos anteriores da adição e multiplicação de dois números para exprimir o tempo estimado nesta nova notação. Recordemos que se n tem k-bits e m tem l-bits, então $k \leq \frac{\log n}{\log 2} + 1$ e $l \leq \frac{\log m}{\log 2} + 1$. Assim, escrevemos que a adição tem complexidade $O(\max(\log m, \log n))$ e a multiplicação $O(\log m \log n)$.

Podemos ter vários tipos de complexidade, entre outros:

- constante: $O(1)$;
- logarítmica: $O(\log n)$;
- linear: $O(n)$;
- polinomial: $O(n^c)$, com $c \in \mathbb{N}$;
- exponencial: $O(a^n)$, com $a > 1$.

Para melhor entender como é que a notação do "big-O" exprime a estimativa do tempo de execução de um dado algoritmo, vejamos um exemplo.

Exemplo 1.2.2. Suponhamos que um algoritmo é executado em tempo cúbico sobre o tamanho do dado de entrada (*input*) N, i.é, $O(n^3)$, onde n é o tamanho de N. Vamos supor que N tem 1024 bits e que a unidade de tempo do computador que temos à disposição, ou seja, o tempo que demora a executar uma operação elementar (*bit operation*), é o nanosegundo (10^{-9} do segundo). Como,

$$\frac{1024^3}{10^9} \approx 1,07.$$

este algoritmo demora (supondo que a constante implícita no "O" é 1) $\approx 1,07$ segundos! Se este algoritmo tivesse tempo $O(2^n)$ onde n continua a ser o tamanho do módulo N em binário, demoraria muitos milénios a ser executado. Para ter um termo de comparação se N tivesse 64-bits em vez dos 1024, sendo portanto 8 vezes mais pequeno, o referido algoritmo demoraria ≈ 584 anos!

Este exemplo serve também para perceber que algoritmos com complexidade exponencial têm pouca utilidade prática. Normalmente os algoritmos desejados, porque são os "eficientes", são os algoritmos polinomiais, sendo assim fundamental na teoria de algoritmos e ciências da computação a seguinte definição.

Definição 1.2.3. *Um algoritmo que solucione um problema envolvendo os inteiros $n_1, n_2, ..., n_r$ com $k_1, k_2, ..., k_r$ bits cada, respectivamente, diz-se um algoritmo em tempo polinomial se existem inteiros $d_1, d_2, ..., d_r$ tais que o número de operações elementares necessárias ao algoritmo é $O(k_1^{d_1} k_2^{d_2} \cdots k_r^{d_r})$.*

O seguinte teorema estabelece algumas propriedades básicas da notação do "big-O".

Teorema 1.2.4. *Sejam f e g funções reais positivas de variável natural.*
(i) Se $c \in \mathbb{R}^+$, então $cO(g) = O(g)$.
(ii) $O(f) + O(g) = O(max\{f, g\})$.
(iii) $O(f)O(g) = O(fg)$.

Prova: (i) Se $f = O(g)$, então existe uma constante $k \in \mathbb{R}^+$ tal que $f(n) \leq kg(n)$ para todo o n suficientemente grande. Assim,

$$cf(n) \leq (ck)g(n),$$

donde se obtém $cf = O(g)$, ou seja, $cO(g) = O(g)$.
(ii) Seja $h_1 = O(f)$ e $h_2 = O(g)$, então existem $c_1, c_2 \in \mathbb{R}^+$ tais que $h_1(n) \leq c_1 f(n)$ e $h_2(n) \leq c_2 g(n)$ para n suficientemente grande. Portanto,

$$\begin{aligned} h_1(n) + h_2(n) &\leq c_1 f(n) + c_2 g(n) \\ &\leq 2\max\{c_1 f(n), c_2 g(n)\} \\ &\leq 2\max\{c_1, c_2\} \max\{f, g\} \end{aligned}$$

ou seja, $O(f) + O(g) = O(\max\{f, g\})$.
(iii) Seja $h_1 = O(f)$ e $h_2 = O(g)$, então existem $c_1, c_2 \in \mathbb{R}^+$ tais que $h_1(n) \leq c_1 f(n)$ e $h_2(n) \leq c_2 g(n)$ para n suficientemente grande. Consequentemente,

$$h_1(n)h_2(n) \leq c_1 c_2 f(n)g(n),$$

o que implica que

$$O(f)O(g) = h_1 h_2 = O(fg).$$

□

Observe-se que (i) é um caso particular de (iii), basta tomar $f(n)$ =constante. Além disso, se $f = g$ e aplicarmos repetidas vezes, digamos k vezes, (iii) obtemos $O(f^k) = O(f)^k$.

1.3 Algoritmo de Euclides

O algoritmo de Euclides é uma forma eficiente de determinar o máximo divisor comum entre dois números inteiros e escrever esse máximo divisor comum, como combinação linear inteira dos dois inteiros dados. Vamos estimar o tempo de execução do algoritmo de Euclides.

Proposição 1.3.1. *Sejam a e b dois números inteiros positivos tais que $a \geq b$. O algoritmo de Euclides é um processo finito de complexidade $O(\log b)$.*

Antes de provar a proposição vamos enunciar e provar um Lema que mostra que o algoritmo de Euclides é um processo finito.

Lema 1.3.2. *Se $a \geq b$ e $a = bq + r$ com $0 \leq r < b$ então $r < \frac{a}{2}$.*

Prova: Como $a \geq b$, tem-se que $q \geq 1$, portanto $bq \geq b$ e $a = bq + r \geq b + r$. Daqui resulta que $r \leq a - b$. Somando com b que é maior que r resulta que $2r < a$. □

Provemos então a proposição 1.3.1.

Prova: Aplicando o algoritmo de Euclides obtemos as seguintes igualdades:

$$\begin{aligned} a &= bq_0 + r_0 \\ b &= r_0q_1 + r_1 \\ r_0 &= r_1q_2 + r_2 \\ r_1 &= r_2q_3 + r_3 \\ &\ldots, \end{aligned} \tag{1.1}$$

para alguns inteiros positivos $q_0, q_1, ...$, $0 \leq r_0 < b$ e $0 \leq r_i < r_{i+1}$, com $i \geq 1$. Assim, pelo lema anterior, cujas hipóteses são satisfeitas pelas igualdades (1.1), resulta que:

$$r_1 < \frac{b}{2}$$
$$r_3 < \frac{r_1}{2} < \frac{b}{2^2}$$
$$r_5 < \frac{r_3}{2} < \frac{b}{2^3}$$
$$...$$
$$r_{2t-1} < \frac{b}{2^t}$$

Faça-se $t = \lceil log_2 b \rceil$. Tem-se $2^t \geq b$ e portanto $r_{2t-1} < 1$, ou seja, este resto tem de ser zero. Resulta então que número de passos a efectuar no algoritmo de Euclides até o processo terminar é inferior a $2t$, que é aproximadamente $2log_2 b$ ou seja o algoritmo tem tempo estimado $O(\log b)$, o que conclui a prova do teorema. □

1.4 Exponenciação modular

Uma outra operação que iremos usar várias vezes é a exponenciação modular, i.é, determinar o menor inteiro positivo congruente com $a^r \ mod \ N$, onde r e N são números naturais. Existe uma forma de determinar este número, mais rápida que multiplicar a por ele próprio r vezes e depois reduzir módulo N. A ideia do algoritmo é escrever a potência r em binário, calcular as potências de a da forma a^{2^i} por aplicação sucessiva da operação "elevar ao quadrado" e, por fim, multiplicar aquelas cujos expoentes aparecem na decomposição binária de r.

Vamos descrever o algoritmo e estimar o seu tempo de execução.

Proposição 1.4.1. *Dados os inteiros positivos a, r e N, tais que $a < N$ o tempo estimado para determinar o menor inteiro positivo congruente com $a^r \ mod \ N$ é $O(\log r \log^2 N)$.*

Prova: Comecemos por descrever o algoritmo que determina o menor inteiro positivo congruente com $a^r \ mod \ N$, onde $a < N$.

Dados de entrada (*input*): a, r, N

Resultado (*output*): y

(1) $x := a$, $y := 1$, $s := r$;

(2) Enquanto $s \neq 1$ fazer

$b := s \ (mod \ 2)$

se $b = 1$ então $y := y \cdot x \ (mod \ N)$

$x := x \cdot x \ (mod \ N)$

$s := \lfloor \frac{s}{2} \rfloor$;

(3) *return* y.

Para estimar o tempo que leva a executar o algoritmo, observemos que em cada passo é efectuada uma ou duas multiplicações de números menores que N^2, visto que $a < N$. Logo em cada passo temos tempo dominado por $O(\log N^2 \log N^2) = O(\log^2 N)$. O número de passos do algoritmo é o número de bits de r, portanto $O(\log r)$. Conclui-se assim que o tempo que demora a exponenciação modular é $O(\log r \cdot \log^2 N)$. □

Capítulo 2

O Sistema Criptográfico RSA

Antes de descrever o sistema criptográfico RSA, neste capítulo, começamos por estabelecer terminologia e conceitos básicos de criptografia. Depois introduz-se o sistema RSA, descrevendo como são geradas as chaves pública e privada, como se encripta e desencripta mensagens e como são geradas as assinaturas digitais imprescindíveis à autenticação. Apresenta-se ainda um resultado que mostra que se duas pessoas que têm chaves públicas diferentes com o mesmo módulo, facilmente encontram a chave privada uma da outra. Este resultado é importante pois alerta para um erro cometido por alguns, como por exemplo, os próprios criadores do RSA, Rivest, Shamir e Adleman, ao propôr um exemplo lúdico de aplicação do sistema criptográfico RSA: o "Mental Poker" [Wad81].

2.1 Terminologia e Conceitos Básicos

Criptografia é a arte de criar escritas secretas com o objectivo de transmitir informação confidencial de forma a serem lidas ou interpretadas apenas por pessoas que conheçam o seu significado. Chamaremos ao texto inicial (*plaintext*) *mensagem original*, ao texto final (*cryptotext*) *criptograma*. Ao processo que transforma a mensagem original no criptograma pode chamar-se codificação, cifragem ou encriptação. Optamos por usar

o termo *encriptação* (*encrypt*). Para o processo inverso, ou seja, para a partir do criptograma recuperar a mensagem original, pode usar-se os termos descodificação, decrifragem ou desencriptação. Aquele por que optamos é *desencriptação* (*decrypt*).

Criptanálise (*Criptanalysis*) é o estudo das debilidades dos sistemas criptográficos e quem faz criptoanálise, o *criptoanalista* (*cryptanalist*) é usualmente denominado de inimigo ou intruso, pois é aquele que tenta quebrar o sistema e conseguir desencriptar os criptogramas dirigidos a terceiros.

O sistema criptográfico RSA, criado em 1978 por Ronald Rivest, Adi Shamir e Leonard Adleman [RSA78], baseia-se na assimetria que existe entre a simplicidade de multiplicar quaisquer dois números primos e a dificuldade, mediante algumas condições que veremos adiante, de dado um inteiro o factorizar em números primos.

O sistema RSA diz-se um sistema criptográfico de chave pública pois neste sistema, contrariamente aos sistemas criptográficos de chave simétrica, em que o receptor e o emissor têm de chegar a acordo sobre a chave a utilizar, tendo de usar algum meio de comunicação para chegar a esse acordo, cada utilizador gera um par de chaves: uma *chave pública* que é do conhecimento de todos, e uma outra associada à primeira que só o próprio conhece, a *chave privada*. A chave pública é usada para encriptar a mensagem original e a chave privada para desencriptar o criptograma.

Os objectivos da criptografia não se limitam à confidencialidade. Devem também permitir ao receptor verificar se uma dada mensagem foi alterada ou danificada durante a sua transmissão, quem é o seu emissor e garantir que o emissor não poderá mais tarde negar o envio da mensagem.

O RSA é o sistema criptográfico de chave pública mais usado nos dias de hoje, não só na transmissão ou armazenamento de informação, mas também em transações e comércio electrónico.

2.2 Descrição do código RSA

Sejam p e q dois primos distintos "grandes", onde grandes se refere ao número de dígitos de cada número, sendo actualmente sugerido números com 512 bits (155 dígitos). Determina-se $N = pq$ e $\varphi(N) = (p-1)(q-1)$. A N chama-se *módulo* e φ é a função de Euler[1].

Considera-se e tal que $(e, \varphi(N)) = 1$ e, usando o algoritmo de Euclides, determina-se d tal que $ed \equiv 1 \ (mod\ \varphi(N))$, ou seja, $d \equiv e^{-1} \ (mod\ \varphi(N))$. A e chama-se *expoente de encriptação* e a d *expoente de desencriptação.* O par $\langle N, e\rangle$ é a *chave pública* porque pode ser divulgada publicamente, e o par $\langle N, d\rangle$ a *chave privada* que só o receptor do criptograma conhece e pode usar para desencriptar o criptograma. Assim, aos expoentes de encriptação e e de desencriptação d também se chama *expoente público* e *expoente privado*, respectivamente.

Seja X a mensagem inicial. Para encriptar a mensagem X usando a chave pública $\langle N, e\rangle$, teremos primeiro de a formatar (ver apêndice A) em blocos de inteiros M tais que $M < N$ e $(M, N) = 1$, pois se M e N não fossem primos entre si ficaria exposta a factorização de N. O processo de encriptação é aplicado a M determinando-se $C \equiv M^e \ (mod\ N)$. Para desencriptar o criptograma C, com vista a recuperar a mensagem M, basta calcular $C^d \ (mod\ N)$. De facto,

$$C^d \equiv M^{ed} \equiv M^{ed-1}\, M \equiv M \ (mod\ N)$$

onde a última igualdade se justifica com o Teorema de Euler [2].

Como $(e, \varphi(N)) = 1$, a cada chave pública corresponde apenas uma única chave privada. Temos portanto, a garantia de que a cada mensagem M corresponde apenas um criptograma C, não havendo qualquer possibilidade de ambiguidade na desencriptação. Finalmente a mensagem original X obtém-se de M aplicando o algoritmo recíproco do que formatou a mensagem inicial X em M.

[1] $\varphi(N)$=número de inteiros positivos menores ou iguais a N que são primos com N.

[2] **Teorema de Euler** [Ang95, p. 107] Se n é um inteiro positivo e $a \in \mathbb{Z}$ são tais que $(a, n) = 1$, então $a^{\varphi(n)} \equiv 1 \ (mod\ n)$.

A escolha de p e de q deve ser feita com alguns cuidados tendo em vista dificultar a factorização de N. Devem ser gerados aleatoriamente seguindo, por exemplo, o seguinte processo: começa-se por gerar um número aleatório grande, x; se x for par então x passa a ser o seu sucessor, $x + 1$; aplica-se um teste de primalidade a x. Se x não for primo então consideramos o número ímpar seguinte, $x + 2$, ao qual aplicamos de novo o teste, e por diante até se obter o primo pretendido. Outra coisa a ter em atenção é que p e q devem ser primos relativamente próximos (mas não demasiadamente próximos porque senão podem ser descobertos pelo método de factorização de Fermat [Rie85, pp. 153-155]). Se o módulo N tem n-bits então os primos p e q devem ter $\frac{n}{2}$-bits cada, i.é, ter o mesmo tamanho (mas com os primeiros bits diferentes, por exemplo), pois quanto maior for um factor primo, mais difícil é encontrá-lo.

A segurança das chaves e o tamanho que devem ter, prende-se com aquilo a que se destinam e depende da rapidez e eficiência do "hardware" e "software" disponíveis no mercado. Em Fevereiro de 1978, quando foi apresentado o sistema criptográfico RSA [RSA78], era aconselhado que as chaves tivessem 200-bits (tamanho do módulo N). Nos dias de hoje [RSA03, /technotes] (ano 2003) se quisermos um sistema de alta segurança para um servidor, deverá usar-se uma chave com 2048-bits, se for um utilizador de um sistema que tem necessidade de arquivar documentos durante algum tempo, deve-se utilizar uma chave com 768 ou 1024 bits. Se for apenas para proteger uma transmissão de informação, como por exemplo, uma mensagem de correio electrónico que quando chegar ao destino será arquivada ou apagada, são então aconselhadas chaves com 512 bits pois quanto maiores forem as chaves mais demorado fica o sistema. Em Agosto de 1999 foi decomposto um número com 512-bits (155 dígitos), denominado RSA-155. Apesar disso, não deixou de ser este o tamanho aconselhado para algum tipo de chaves, pois para além da sua factorização ter demorado 7 meses e envolvido muitos meios, utilizou só métodos de factorização já existentes e foi necessário em grosso modo o tempo já previsto à priori.

Há formas de acelerar a encriptação e desencriptação. Algumas delas serão abordadas

no capítulo 4, mas um exemplo de uma forma de tornar a encriptação mais rápida, é escolher um expoente de encriptação e que quando escrito em binário tenha muitos zeros, como por exemplo os primos 3, 17 ou $2^{16}+1$, e usar a exponenciação modular. O processo torna-se mais rápido porque, por exemplo no caso de $e = 2^{16}+1 = 65537$, o que se faz, tal como descrito na secção 1.4, é começar por determinar o menor inteiro positivo tal que $M^{2^{16}} \ (mod\ N) = ((((M^2)^2)^2)^2)^2 \ (mod\ N)$. Por fim multiplica-se o resultado por M e reduz-se $mod\ N$. Portanto a encriptação com este expoente público requer apenas 17 multiplicações o que a torna um processo mais rápido que a desencriptação. Como e não deve, por razões de segurança que veremos no capítulo 3, ser demasiado pequeno, utiliza-se muitas vezes este expoente público, $e = 2^{16}+1$. Para tornar a desencriptação mais rápida pode usar-se o Teorema Chinês dos Restos, pois o receptor conhece os factores p e q do módulo N. Consideremos o isomorfismo canónico (ver apêndice B)

$$\begin{array}{ccc} \Psi : \mathbb{Z}_N^* & \longrightarrow & \mathbb{Z}_p^* \times \mathbb{Z}_q^* \\ x & \mapsto & (x, x). \end{array}$$

Fazendo $c^d = \Psi^{-1}(\Psi(c^d)) = \Psi^{-1}(c^d, c^d)$, calculam-se $c^d \ mod\ p$ e $c^d \ mod\ q$ em $\mathbb{Z}_p$ e em $\mathbb{Z}_q$, respectivamente, em vez de c^d em $\mathbb{Z}_N$, e depois usa-se o Teorema Chinês dos Restos, trabalhando assim com números que têm metade dos bits de N.

Uma outra questão importante na escolha de chaves para o sistema criptográfico RSA é assegurar-nos que chaves diferentes têm módulos diferentes.

Por exemplo, imaginemos uma rede de comunicação com $i > 1$ utilizadores. O gestor da rede atribuí um par e_i, d_i a cada utilizador ficando assim cada um com as suas chaves pública e privada $\langle N, e_i \rangle$ e $\langle N, d_i \rangle$, respectivamente. Aparentemente desde que cada um não revelasse a sua chave privada, a rede seria segura. Mas, o teorema seguinte mostra que expôr um expoente privado d_i de uma chave $\langle N, d_i \rangle$ é equivalente a factorizar N. Portanto mesmo que não tenha sido fornecida a factorização de N a nenhum dos i utilizadores, eles conseguiriam factorizar N o que tornaria a rede totalmente insegura.

Teorema 2.2.1. *Seja $\langle N, e\rangle$ uma chave pública RSA. Dada a chave privada d, podemos factorizar o módulo $N = pq$ usando um algoritmo probabilístico de tempo esperado $O(\log^3 N)$, com probabilidade de sucesso maior ou igual a $\frac{1}{2}$. Reciprocamente, dada a factorização de N podemos determinar d.*

Prova: O facto que dada a factorização de N se pode determinar d, resulta da própria construção do sistema RSA.

Provemos agora que tendo d se consegue factorizar N. Dado d, seja $k = ed - 1$. Pela definição de d e e sabe-se que $\varphi(N)|k$. Como $\varphi(N)$ é par pode-se escrever $k = 2^t r$ onde r é ímpar e $t \geq 1$. Tem-se que $g^k = 1$ para todo o $g \in \mathbb{Z}_N^*$ porque pelo teorema de Euler, como $(g, N) = 1$ e $k = \varphi(N)a$ para algum $a \in \mathbb{N}$,

$$g^k \equiv \left(g^{\varphi(N)}\right)^a \equiv 1^a \ (mod\ N) \equiv 1 \ (mod\ N).$$

Em $\mathbb{Z}_N$ existem 4 raízes quadradas da unidade, pois pelo Teorema Chinês dos Restos,

$$x^2 \equiv 1 \ (mod\ N) \Leftrightarrow \begin{cases} x^2 \equiv 1 \ (mod\ p) \\ x^2 \equiv 1 \ (mod\ q) \end{cases}$$

$$\Leftrightarrow \begin{cases} x \equiv 1 \ mod\ p \\ x \equiv -1 \ mod\ q \end{cases} \vee \begin{cases} x \equiv -1 \ mod\ p \\ x \equiv 1 \ mod\ q \end{cases} \vee x \equiv \pm 1 \ (mod\ pq).$$

Se determinarmos uma das duas raízes não triviais, ou seja, x tal que

$$x \equiv 1 \ mod\ p \ \wedge \ x \equiv -1 \ mod\ q \quad \text{ou} \quad x \equiv -1 \ mod\ p \ \wedge \ x \equiv 1 \ mod\ q$$

fica exposta a factorização de N pois ou $(x - 1, N) = p$ ou $(x - 1, N) = q$.

Assim, o objectivo é encontrar uma raiz quadrada não trivial de 1 $(mod\ N)$ usando o facto de $g^k = 1$ e portanto $g^{2^l r}$, $l = 0, 1, \cdots, t - 1$ são possíveis candidatos. Para isso vamos provar que:

> Se g é escolhido aleatoriamente em $\mathbb{Z}_N^*$, então a probabilidade de g ser tal que um dos elementos da sequência $g^{k/2}; g^{k/4}; ...; g^{k/2^t}\ mod\ N$ é uma raiz quadrada não trivial da unidade é $\frac{1}{2}$.

Consideremos o conjunto

$$G = \{g \in \mathbb{Z}_N^* : g^{2^l r} \text{ é uma raiz quadrada da unidade diferente de } \pm 1, \text{ para algum } l \in \{0, 1, ..., t-1\}\}.$$

O que queremos provar é que $\#G \geq \frac{\varphi(N)}{2}$.

Como já vimos $g^{2^l r}$ é uma raiz quadrada da unidade não trivial se

$$\begin{cases} g^{2^l r} \equiv 1 \ (mod\ p) \\ g^{2^l r} \equiv -1 \ (mod\ q) \end{cases} \tag{2.1}$$

ou

$$\begin{cases} g^{2^l r} \equiv -1 \ (mod\ p) \\ g^{2^l r} \equiv 1 \ (mod\ q). \end{cases} \tag{2.2}$$

Consideremos o isomorfismo canónico $\mathbb{Z}_N^* \longrightarrow \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ (ver apêndice B) e sejam α e β geradores dos grupos cíclicos $\mathbb{Z}_p^*$ e $\mathbb{Z}_q^*$, respectivamente. Via este isomorfismo g corresponde a um elemento $(\alpha^i, \beta^j) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ e portanto $(\alpha^{2^l ri}, \beta^{2^l rj}) = \left(\left[g^{2^l r}\right]_p, \left[g^{2^l r}\right]_q\right) = (\pm 1, \mp 1)$. Assim,

$$\#G = \#\{(i,j) : (\alpha^{2^l ri}, \beta^{2^l rj}) = ([1]_p, [-1]_q) \text{ ou } (\alpha^{2^l ri}, \beta^{2^l rj}) = ([-1]_p, [1]_q) \text{ para algum } l \in \{0, 1, ..., t-1\}\}.$$

Podemos pois reescrever (2.1) e (2.2) da seguinte forma

$$\begin{cases} 2^l ri \equiv 0 \ (mod\ p-1) \\ 2^l rj \equiv \frac{q-1}{2} \ (mod\ q-1) \end{cases} \tag{2.1'}$$

ou

$$\begin{cases} 2^l ri \equiv \frac{p-1}{2} \ (mod\ p-1) \\ 2^l rj \equiv 0 \ (mod\ q-1) \end{cases} \tag{2.2'}$$

e contar, para cada $l \in \{0, 1, ..., t-1\}$ os pares (i, j) que satisfazem estas condições.

Relembremos que $k = ed - 1 = 2^t r$ com r ímpar e $t \geq 1$. Sejam $p - 1 = 2^a u$ e

$q-1 = 2^b v$ onde $2 \nmid uv$ e vamos assumir, sem perda de generalidade, que $1 \leq a \leq b$. Observe-se que $uv|r$.

Por um resultado elementar[3] da Teoria dos Números sabemos que $2^l ri \equiv 0 \ (mod\ p-1)$ tem solução se e só se $(2^l r, p-1)|p-1$. Como

$$(2^l r, p-1) = (2^l r, 2^a u) = u2^{min\{l,a\}}$$

e $u2^{min\{l,a\}}|2^a u$, a congruência tem sempre solução e além disso, pelo mesmo resultado, sabemos que o número de soluções i é $u2^{min\{l,a\}}$. De forma análoga concluí-se que o número de soluções para j na segunda congruência em (2.1´) é

$$(2^l r, q-1) = (2^l r, 2^b v) = v2^{min\{l,b\}}$$

quando $2^{min\{l,b\}} \mid 2^{b-1}$, portanto $l < b$, ou seja, o número de soluções para j é $v2^l$ quando $l \leq b-1$ e zero se $l \geq b$.

De modo análogo, de (2.2') obtém-se o número de soluções para i na primeira congruência:

$$(2^l r, p-1) = (2^l r, 2^a u) = u2^{min\{l,a\}}$$

quando $2^{min\{l,a\}} \mid 2^{a-1}$, portanto $l < a$, ou seja, o número de soluções para i é $u2^l$ quando $l < a$ e zero se $l \geq a$.

O número de soluções para j na segunda congruência é

$$(2^l r, q-1) = (2^l r, 2^b v) = v2^{min\{l,b\}}.$$

Como $a \leq b$ e a primeira congruência só tem solução quando $l < a$, o número de soluções da segunda é, neste caso, $v2^l$.

Em resumo, dado l, tem-se:

- (2.1´) tem solução sse $l < b$ e o número de soluções é $uv2^{\min\{l,a\}} \cdot 2^l$;

[3] **Teorema** [LeV96, p. 58] A congruência $ax \equiv b \pmod{m}$ com $(a,m) = d$ tem solução se e só se $d \mid b$. Além disso, se $d|b$, o número de soluções é d.

- (2.2′) tem solução sse $l < a$ e o número de soluções é $uv2^{\min\{l,a\}} \cdot 2^l = uv2^{2l} = uv4^l$.

Para cada l a solução (i, j) é única (injectividade do isomorfismo), portanto na sequência $g^r, g^{2r}, ..., g^{2^{t-1}r}$ só pode existir uma raiz quadrada da unidade não trivial para cada l. O número de pares (i, j) procurados, é assim, o produto do número de soluções i pelo número de soluções j, obtidas em cada sistema.
No sistema (2.1′) existem $uv2^{\min\{l,a\}}2^l$ soluções para $l = 0, 1, ..., a, ..., b-1$. Como $a \leq b$ o número de soluções para cada $l = 0, 1, ..., a, ..., b-1$ é:

$$u \cdot v, 2u \cdot 2v, 2^2u \cdot 2^2v, \cdots, 2^au \cdot 2^av, 2^au \cdot 2^{a+1}v, \cdots, 2^au \cdot 2^{b-1}v$$

portanto o número total é

$$\begin{aligned} uv \cdot (1 + 4 + \cdots + 4^a) + uv \cdot 4^a(2 + \cdots + 2^{b-a}) &= uv\tfrac{1-4^{a+1}}{1-4} + uv \cdot 4^a \cdot 2\tfrac{1-2^{b-a-1}}{1-2} \\ &= uv \cdot \left[\tfrac{4^{a+1}-1}{3} + 2^{a+b} - 2^{2a+1}\right]. \end{aligned}$$

No sistema (2.2′) existem soluções (i, j) para $l = 0, 1, \cdots, a-1$ portanto

$$u \cdot v, u2 \cdot v2, u2^2 \cdot v2^2, \cdots, u2^{a-1} \cdot v2^{a-1}$$

donde

$$uv \cdot (1 + 4 + \cdots + 4^{a-1}) = uv \cdot \frac{4^a - 1}{4 - 1}.$$

Resulta então que o número total de soluções (i, j) com $a \leq b$ é:

$$\begin{aligned} uv \cdot \left(\tfrac{4^{a+1}-1}{3} + 2^{a+b} - 2 \cdot 4^a + \tfrac{4^a-1}{3}\right) &= \tfrac{p-1}{2^a}\,\tfrac{q-1}{2^b}\left[\tfrac{4^a(4-6+1)-2}{3} + 2^{a+b}\right] \\ &= \varphi(N)\left[1 - \tfrac{2+4^a}{3 \cdot 2^{a+b}}\right] \\ &\geq \varphi(N)\left(1 - \tfrac{2+4^a}{3 \cdot 4^a}\right), \text{ porque } a < b \\ &= \varphi(N)\left(1 - \tfrac{2}{3 \cdot 4^a} - \tfrac{1}{3}\right) \\ &\geq \varphi(N)\left(\tfrac{2}{3} - \tfrac{1}{6}\right), \text{ porque } a \geq 1 \\ &= \tfrac{\varphi(N)}{2} \end{aligned}$$

como se queria provar.

Fica assim provado que a probabilidade de g ser tal que um dos elementos da sequência $g^{k/2}; g^{k/4}; ...; g^{k/2^t} \ mod \ N$ é uma raíz quadrada não trivial da unidade, é maior ou igual a $\frac{1}{2}$.

Falta ver que todos os elementos da sequência podem ser eficientemente determinados em tempo $O(\log^3 N)$. Pela proposição1.4.1 o calculo de $g^{2^t r}$ tem complexidade dominada por $O(\log(2^t r)\log^2 N)$ (ao calcular $g^{2^t r}$ passa-se por todos os elementos da sequência $g^r, g^{2r}, g^{2^2 r}, \cdots, g^{2^t r}$). Observe-se que $2^t r = k$ e $k|\varphi(N)$ logo $k \leq \varphi(N)$. Como $\varphi(N)$ tem tamanho próximo de N, $\log(k) \leq \log(N)$ e portanto $O(\log(k)) = O(\log N)$, donde se conclui que o algoritmo tem complexidade $O(\log N \cdot \log^2 N) = O(\log^3 N)$. □

2.3 Assinaturas Digitais

Pode-se ao enviar uma mensagem querer também assiná-la, tal como se assina um documento ou um cheque. O sistema RSA pode também ser usado para esse fim, sendo possível associar a uma mensagem uma *assinatura digital* que não é mais que um número inteiro que garante a origem da mensagem.

Para assinar uma mensagem M o emissor começa por usar a sua chave privada, digamos $\langle N_E, d_E \rangle$, para determinar

$$A \equiv M^{d_E} \ (mod \ N_E).$$

Em seguida o emissor usa a chave pública do receptor $\langle N_R, e_R \rangle$ para encriptar a mensagem assinada A, fazendo

$$C \equiv A^{e_R} \ (mod \ N_R)$$

e envia C ao receptor. O receptor pode então desencriptar o criptograma C usando a sua chave privada $\langle N_R, d_R \rangle$ e verificar a origem da mensagem usando a chave pública

do emissor $\langle N_E, e_E \rangle$. Ou seja, o que receptor tem que fazer é determinar

$$C^{d_R} \ (mod\ N_R) \equiv A \quad \text{e} \quad A^{e_E} \ (mod\ N_E) \equiv M.$$

Devemos ter em atenção que, porque os módulos não são iguais, não podemos trocar a ordem das coisas tendo mesmo de em primeiro lugar desencriptar a mensagem e só depois verificar a origem da mensagem, i.é, verificar a assinatura. Outra questão que se coloca por estarem a ser usados módulos diferentes é que se $N_E < N_R$ não há problemas mas se $N_E > N_R$ parte da mensagem será perdida!

Existem várias formas de ultrapassar o problema:

- ter em atenção o tamanho de N_R e N_E que são públicos, começar o processo pela assinatura, como acima descrito, se $N_E < N_R$ ou iniciar pela encriptação e só depois assinar, caso contrário;
- o emissor ter o cuidado de redefinir o tamanho dos blocos em que se divide a mensagem depois de assinada;
- assegurar uma lista de chaves de tal forma que todos os utilizadores do RSA usem as chaves com módulo mais pequeno para assinar mensagens e as de módulo maior para encriptar mensagens assinadas.

Existem outras formas de assinaturas digitais. Quisemos apenas mostrar como pode o RSA ser usado não só para encriptação de mensagens mas também para garantir a origem das mensagens, cumprindo assim os vários objectivos da criptografia já atrás referidos: codificação, autenticação e possibilidade de verificar a origem das mensagens enviadas.

Capítulo 3

Descrição de Ataques ao RSA

Neste capítulo descrevem-se alguns ataques ao sistema RSA que se aplicam quando são utilizados expoentes públicos ou privados pequenos. Começa-se por dar um exemplo de um ataque no caso em que se usa um expoente privado pequeno. Enuncia-se o Teorema de Coppersmith que permite determinar de forma eficiente raízes inteiras pequenas de polinómios de coeficientes inteiros módulo N. Este teorema é utilizado em vários ataques ao sistema criptográfico RSA, quando se usa expoente público pequeno, e de que apresentamos dois exemplos.

3.1 Expoente Privado Pequeno

Numa tentativa de tornar o processo de desencriptação mais rápido podemos cair na tentação de escolher um expoente privado d "pequeno". O que o seguinte teorema de M. Wiener mostra é que a escolha de d "pequeno" pode levar à ruptura total da segurança do sistema criptográfico RSA, ou seja, consegue-se determinar o expoente privado d e consequentemente a factorização de N.

Teorema 3.1.1 (M.Wiener). *Seja $N = pq$, com $q < p < 2q$. Seja $d < \frac{1}{3}N^{\frac{1}{4}}$. Dada a chave pública RSA $\langle N, e\rangle$ tal que $ed \equiv 1 \ (mod\ \varphi(N))$ consegue-se determinar d em*

tempo $O(\log N)$.

Prova: A prova baseia-se na teoria de aproximação por fracções contínuas.
Seja $\langle N, e\rangle$ uma chave pública RSA tal que $ed \equiv 1 \ (mod\ \varphi(N))$. Então existe $k \in \mathbb{N}$ tal que $ed - k\varphi(N) = 1$. Ora, $ed - k\varphi(N) = 1 \Leftrightarrow \frac{ed}{\varphi(N)} - k = \frac{1}{\varphi(N)}$ e portanto podemos escrever

$$\left|\frac{e}{\varphi(N)} - \frac{k}{d}\right| = \frac{1}{d\varphi(N)}. \tag{3.1}$$

Em princípio $\varphi(N)$ é um número "grande" o que torna $\frac{k}{d}$ numa boa aproximação de $\frac{e}{\varphi(N)}$.
Não conhecemos $\varphi(N)$, mas

$$\varphi(N) = (p-1)(q-1) = pq - p - q + 1 = N - (p + q - 1)$$

e podemos usar N para conseguir uma aproximação de $\varphi(N)$. Como por hipótese $q < p < 2q$, tem-se

$$q^2 < pq \ \text{ e } \ p^2 < 2pq, \text{ portanto } q < \sqrt{N} \ \text{ e } \ p < \sqrt{2N}.$$

Logo

$$p + q - 1 < p + q < \sqrt{2}\sqrt{N} + \sqrt{N} = (\sqrt{2} + 1)\sqrt{N} < 3\sqrt{N}.$$

Tem-se assim uma aproximação de $\varphi(N)$ por N

$$|N - \varphi(N)| = |p + q - 1| < 3\sqrt{N}.$$

Como vimos $\frac{e}{\varphi(N)}$ é uma boa aproximação de $\frac{k}{d}$, mas $\frac{e}{\varphi(N)}$ é desconhecido. A ideia para obter k e d é, em vez de usar a aproximação de $\frac{k}{d}$ por $\frac{e}{\varphi(N)}$ apresentada em (3.1), aproximar $\frac{k}{d}$ por $\frac{e}{N}$, pois $\frac{e}{N}$ é conhecido e próximo de $\frac{e}{\varphi(N)}$, e depois usar o facto de as fracções continuas serem um bom meio para obter boas aproximações. Tem-se

$$\begin{aligned}
\left|\frac{e}{N} - \frac{k}{d}\right| &= \left|\frac{ed - kN}{Nd}\right| \\
&= \left|\frac{ed - k\varphi(N) - kN + k\varphi(N)}{Nd}\right| = \left|\frac{1 - k(N - \varphi(N))}{Nd}\right| \\
&= \frac{k(N - \varphi(N)) - 1}{Nd} < \frac{k(N - \varphi(N))}{Nd} \\
&< \frac{3k\sqrt{N}}{Nd} = \frac{3k}{\sqrt{N}d}.
\end{aligned}$$

Observe-se que $k\varphi(N) = ed-1 < ed$, $e < \varphi(N)$ por construção da chave pública RSA, e por hipótese $d < \frac{1}{3}N^{\frac{1}{4}}$ portanto

$$k\varphi(N) < ed < \varphi(N)d \text{ o que implica } k < d < \frac{1}{3}N^{\frac{1}{4}}$$

obtendo-se assim

$$\begin{aligned}\left|\frac{e}{N} - \frac{k}{d}\right| &< \frac{3k}{\sqrt{N}d} \\ &< \frac{3 \cdot \frac{1}{3}N^{\frac{1}{4}}}{d \cdot \sqrt{N}} = \frac{N^{\frac{1}{4}}}{d \cdot N^{\frac{1}{2}}} \\ &= \frac{1}{d \cdot N^{\frac{1}{4}}} = \frac{1}{3} \cdot \frac{1}{d} \cdot \frac{3}{N^{\frac{1}{4}}} \\ &< \frac{1}{3d^2} < \frac{1}{2d^2}.\end{aligned}$$

Como $ed - k\varphi(N) = 1$, $\frac{k}{d}$ é uma fracção racional irredutível e, além disso, satisfaz a desigualdade $\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{1}{2d^2}$. Portanto, por um resultado clássico da teoria de aproximação por facções contínuas[1] $\frac{k}{d}$ é uma convergente de $\frac{e}{N}$.

Vamos agora descrever como se determinam as convergentes de $\frac{e}{N}$. Aplicando o algoritmo de Euclides a e e N, existem inteiros $q_i \geq 0$ e $0 \leq r_i < r_{i+1}$ com $i \geq 0$ e $0 \leq r_0 \leq e$, obtêm-se as seguintes igualdades

$$\begin{aligned} e &= Nq_0 + r_0 \\ N &= r_0q_1 + r_1 \\ r_0 &= r_1q_2 + r_2 \\ r_1 &= r_2q_3 + r_3 \\ &\ldots \end{aligned} \tag{3.2}$$

portanto

[1] **Teorema** [Khi97, p. 30] Toda a fracção racional irredutível $\frac{p}{q}$ que satisfaz a desigualdade $\left|x - \frac{p}{q}\right| < \frac{1}{2q^2}$ é uma convergente de x.

$$\begin{aligned}
\frac{e}{N} &= \frac{Nq_0+r_0}{r_0q_1+r_1} \\
&= \frac{e}{eq_1+r_1}, \text{ porque } e < \varphi(N) < N \text{ logo } q_0 = 0 \text{ e } r_0 = e \\
&= \frac{1}{q_1+\frac{r_1}{e}} \\
&= \frac{1}{q_1+\frac{r_1}{r_1q_2+r_2}} \\
&= \frac{1}{q_1+\frac{1}{q_2+\frac{r_2}{r_1}}} \\
&= \ldots
\end{aligned}$$

o que dá um desenvolvimento em fracção contínua simples (finita)

$$\frac{e}{N} = \frac{1}{q_1 + \frac{1}{q_2+\frac{1}{q_3+\frac{1}{\ldots+q_n}}}}.$$

Se representarmos o desenvolvimento em fracção contínua de $\frac{e}{N}$ por $(q_0, q_1, ..., q_n)$, então $\frac{k}{d} = (q_0, q_1, ..., q_m)$, para algum $m \in \mathbb{N}$ em virtude de $\frac{k}{d}$ ser uma convergente de $\frac{e}{N}$.

Para completar a prova falta ainda ver que por este processo se obtém d de forma eficiente em tempo $O(\log N)$. Como vimos as convergentes de $\frac{e}{N}$ determinam-se aplicando o algoritmo de Euclides. Pela proposição 1.3.1 sabemos que o algoritmo de Euclides tem tempo estimado $O(\log N)$. Como $\frac{k}{d}$ é uma dessas convergentes e é uma fracção irredutível, fica determinado d em tempo dominado por $O(\log N)$, o que conclui a prova do teorema. □

3.2 Expoente Público Pequeno

Tal como para acelerar o processo de desencriptação se pode cair na tentação de escolher um expoente privado "pequeno", no processo de encriptação podemos querer escolher um expoente público pequeno na tentativa de tornar a encriptação mais rápida. O menor valor possível para o expoente público e é 3, mas com vista a prevenir eventuais ataques, como os que descrevemos nesta secção, o valor sugerido é $2^{16} + 1$. Este número tem a particularidade de ser uma potência de dois acrescida de

uma unidade, o que permite fazer a exponenciação modular $M^e \ mod \ N$, onde M é a mensagem a encriptar e N é o módulo RSA, de uma forma um pouco mais rápida, como já foi referido no capítulo 2.

Contrariamente ao que acontece nos ataques que têm por objectivo obter o expoente privado e portanto a factorização de N, os ataques ao expoente público estão longe de levar à ruptura total do sistema RSA pois não chegam a factorizar o módulo N. O objectivo destes ataques é normalmente desencriptar uma mensagem sem precisar de descobrir a chave privada. Alguns dos ataques mais poderosos ao expoente público usam o Teorema de Coppersmith, que iremos enunciar e provar antes de dar exemplo desses ataques.

3.2.1 Teorema de Coppersmith

Seja $f(x)$ um polinómio de grau k numa variável,

$$f(x) = x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0.$$

Vamos assumir que $f(x)$ é um polinómio mónico irredutível e N um número composto difícil de factorizar. O teorema de Coppersmith fornece um método eficiente de, em certas condições, determinar todas as raízes pequenas de $f \ mod \ N$. No método só é relevante o facto de $f(x)$ ser mónico e as restantes propriedades de N e f irão ser importantes quando estivermos a aplicar o teorema no ambiente exigido pelo sistema criptográfico RSA.

Teorema 3.2.1 (Coppersmith). *Sejam N um inteiro do qual não se conhece a factorização e $f(x) \in \mathbb{Z}[x]$ um polinómio mónico de grau k. Então, consegue-se encontrar todos os inteiros $|x_0| < \frac{1}{\sqrt{2}}N^{1/k}$ tais que $f(x_0) \equiv 0 \ (mod \ N)$, em tempo polinomial sobre $\log N$ e k.*

Coppersmith, em [Cop97], descreve e prova este método eficiente de determinar todas as raízes "pequenas" inteiras módulo N, do polinómio $f(x)$. Ainda em 1997,

Howgrave-Graham apresenta um método mais simples, para determinar essas mesmas raízes. No ano seguinte, em 1998, na sua tese de doutoramento [HG98] mostra que os dois métodos são equivalentes, depois de efectuados alguns ajustes ao método apresentado por Coppersmith.

Para provar o teorema é necessário introduzir alguns conceitos e resultados preliminares.

Definição 3.2.2. *Seja n um inteiro positivo. Um subconjunto L do espaço vectorial $\mathbb{R}^n$ diz-se um* ***reticulado*** *se existe um conjunto de vectores de $\mathbb{R}^n$ linearmente independentes $\{b_1, b_2, ..., b_n\}$ tais que*

$$L = \sum_{i=1}^{n} \mathbb{Z} b_i = \left\{ \sum_{i=1}^{n} c_i b_i : \; c_i \in \mathbb{Z} \text{ e } 1 \leq \text{i} \leq \text{n} \right\}.$$

Se assim acontecer, dizemos que $\{b_1, b_2, ..., b_n\}$ é uma ***base*** *para L e que o reticulado tem* ***dimensão*** *n.*

Um reticulado pode ser visto como o conjunto das combinações lineares inteiras dos vectores linha $\{b_1, b_2, ..., b_n\}$ de uma matriz M, que formam uma base para o reticulado. Assim, podemos definir o determinante do reticulado como sendo $\det(L) = |\det(b_1, \cdots, b_n)| = |\det(M)|$. Em particular, $\det(M) \neq 0$. Note-se que este determinante não depende da base escolhida porque se tivermos duas bases para L, as respectivas matrizes de mudança de base têm entradas no conjunto $\mathbb{Z}$ dos números inteiros e são inversas uma da outra. Por isso os seus determinantes são números inteiros cujo produto é um, donde se conclui que os determinates são ± 1 e assim o seu módulo é 1.

Utilizando o processo de ortogonalização de Gram-Schmidt à base $\{b_1, b_2, ..., b_n\}$, obtém-se uma base ortogonal $\{b_1^*, b_2^*, ..., b_n^*\}$. Os vectores b_i^* com $1 \leq i \leq n$ e os números reais μ_{ij}, $1 \leq j < i \leq n$ são indutivamente definidos por

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*,$$

$$\mu_{ij} = \frac{b_i \bullet b_j^*}{b_j^* \bullet b_j^*}$$

onde $\bullet$ denota o produto interno em $\mathbb{R}^n$. Assim, $\det(L) = |(b_1^*, b_2^*, \cdots, b_n^*)|$ e como os vectores b_i^* são ortogonais entre si dois a dois, $\det(L) = \prod_{i=1}^n \|b_i^*\|$.

Em [LLL82], Lenstra, Lenstra Jr. e Lovász introduzem o seguinte conceito de base reduzida.

Definição 3.2.3. *Uma base* $\{b_1, b_2, ..., b_n\}$ *do reticulado* L *diz-se* ***reduzida*** *se*

$$|\mu_{ij}| \leq \frac{1}{2}, \text{ para } 1 \leq j < i \leq n$$

e

$$\|b_i^* + \mu_{ii-1} b_{i-1}^*\|^2 \geq \frac{3}{4}\|b_{i-1}^*\|^2, \text{ para } 1 < i \leq n,$$

onde $\|\cdot\|$ *denota a norma euclideana.*

A seguinte proposição estabelece algumas propriedades das bases reduzidas.

Proposição 3.2.4. *Seja* $\{b_1, b_2, \cdots, b_n\}$ *uma base reduzida para o reticulado* L *em* $\mathbb{R}^n$, *e seja* $\{b_1^*, b_2^*, \cdots, b_n^*\}$ *a base obtida quando aplicado o processo de ortogonalização de Gram-Schmidt a essa base. Tem-se:*

(i) $\|b_j\|^2 \leq 2^{i-1}\|b_i^*\|^2$, *para* $1 \leq j \leq i \leq n$;

(ii) $\det(L) \leq \prod_{i=1}^n \|b_i\| \leq 2^{n(n-1)/4} \det(L)$;

(iii) $\|b_1\| \leq 2^{(n-1)/4} \det(L)^{1/n}$.

Prova: Da definição de base reduzida temos que

$$\|b_i^*\|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right) \|b_{i-1}^*\|^2 \geq \frac{1}{2}\|b_{i-1}^*\|^2,$$

para $1 < i \leq n$. Portanto, por indução resulta que

$$\|b_j^*\|^2 \leq 2^{i-j}\|b_i^*\|^2, \text{ para } 1 \leq j \leq i \leq n.$$

Como $b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$, b_i^* e b_j^* são ortogonais, $i \neq j$ e $|\mu_{i,j}| \leq 1/2$, resulta que

$$\begin{aligned} \|b_i\|^2 &= \|b_i^*\| + \sum_{j=1}^{i-1} \mu_{i,j}^2 \|b_j^*\|^2 \\ &\leq \|b_i^*\| + \sum_{j=1}^{i-1} \frac{1}{4} 2^{i-j} \|b_i^*\|^2 \\ &= \left(1 + \frac{1}{4}(2^i - 2)\right) \|b_i^*\|^2 \\ &\leq 2^{i-1} \|b_i^*\|^2. \end{aligned}$$

Assim, $\|b_j\|^2 \leq 2^{j-1}\|b_j^*\|^2 \leq 2^{i-1}\|b_i^*\|^2$ para $1 \leq j \leq i \leq n$ o que prova (i).
Sabemos que $\det(L) = \prod_{i=1}^{n} \|b_i^*\|$ e $\|b_i^*\| \leq \|b_i\| \leq 2^{(i-1)/2}\|b_i^*\|$ donde se conclui (ii).
Para provar (iii) basta fazer $j = 1$ em (i) e multiplicar as desigualdades obtidas para $i = 1, \cdots, n$. □

No mesmo artigo [LLL82] é apresentado um algoritmo que determina uma base reduzida $\{b_1, b_2, ..., b_n\}$ para um reticulado L em tempo[2] $O(n^4 \log R)$, onde R é um número real maior que 2 tal que $|b_i|^2 \leq R$ para $1 \leq i \leq n$. Como "output" esse algoritmo fornece uma matriz B, cujas linhas são os vectores da base reduzida, que se obtém da matriz M, operando sobre as suas linhas.

Estamos agora em condições de provar o teorema de Coppersmith seguindo o método apresentado por Howgrave-Graham.

Prova: Seja $f(x) \in \mathbb{Z}[x]$ um polinómio mónico de grau k. Seja X um número natural tal que $|x| \leq X$, a determinar convenientemente adiante. Pretende-se determinar todas as raízes inteiras pequenas, x_0, de $f(x) \equiv 0 \ (mod\ N)$.
Para algum $h \geq 2$ defina-se a matriz triângular inferior $M = (m_{i,j})$ com hk linhas e hk colunas, onde $m_{i,j} = a_{i,j} X^{j-1}$ para $i = 1, 2, \cdots, hk$ e $j = 1, \cdots, i$, e os $a_{i,j}$ são os

[2]Se usarmos os algoritmos leccionados na escola primária para as operações elementares o tempo estimado é $O(n^6(\log R)^3)$, tal como foi descrito no capítulo 1, conseguindo-se um tempo estimado por $O(n^4 \log R)$ quando são utilizadas técnicas mais eficientes para multiplicar.

coeficientes de x^{j-1} na seguinte expressão:

$$q_{u,v}(x) = N^{(h-1-v)}x^u\,(f(x))^v = a_{i,1} + a_{i,2}x + \cdots + a_{i,i}x^{i-1} \tag{3.3}$$

com $v = \lfloor (i-1)/k \rfloor$ e $u = (i-1) - kv$, ou seja, $i - 1 = kv + u$ sendo v o quociente da divisão de $i-1$ por k e u o resto da mesma divisão. Note-se que $q_{u,v}(x_0) \equiv 0\ (mod\ N^{h-1})$ para todo o $u, v \geq 0$, pois

$$\begin{aligned} f(x_0) \equiv 0\ (mod\ N) &\Leftrightarrow (f(x_0))^v \equiv 0\ (mod\ N^v) \\ &\Leftrightarrow x_0^u(f(x_0))^v \equiv 0\ (mod\ N^v) \\ &\Leftrightarrow N^{h-1-v}x^u(f(x_0))^v \equiv 0\ (mod\ N^{h-1}). \end{aligned}$$

Como M é uma matriz triângular, o seu determinante é o produto dos elementos da diagonal, que são os coeficientes dos termos de maior grau em cada um dos polinómios $q_{u,v}$. Logo

$$\begin{aligned} \det(M) &= \prod_{j=1}^{hk} N^{h-1-v}X^{j-1} \\ &= N^{\sum_{j=1}^{hk} h-1-\lfloor \frac{j-1}{k} \rfloor} X^{\sum_{j=1}^{hk}(j-1)} \\ &= N^{hk(h-1)/2}X^{hk(hk-1)/2} \end{aligned}$$

porque

$$\sum_{j=1}^{hk} \left\lfloor \frac{j-1}{k} \right\rfloor = \sum_{j=k+1}^{hk} \left\lfloor \frac{j-1}{k} \right\rfloor = \sum_{j=k}^{hk-1} \left\lfloor \frac{j}{k} \right\rfloor = k \cdot 1 + k \cdot 2 + \cdots + k \cdot (h-1)$$

uma vez que

$$\left\lfloor \frac{j}{k} \right\rfloor = \begin{cases} 1, & k \leq j < 2k \\ 2, & 2k \leq j < 3k \\ \vdots & \\ h-1, & (h-1)k \leq j < hk. \end{cases}$$

Seja B a matriz cujas linhas são a base reduzida que se obtém aplicando o algoritmo de redução LLL às linhas da matriz M, e denotemos o primeiro vector linha de B por b_1. Pela propriedade (iii) da Proposição 3.2.4 resulta que

$$\|b_1\| \leq 2^{(hk-1)/4}X^{(hk-1)/2}N^{(h-1)/2}. \tag{3.4}$$

Observemos que $\|b_1\| \geq \frac{1}{\sqrt{hk}} \sum_{j=1}^{hk} |b_{1,j}|$ pois pela Desigualdade de Chauchy[3]

$$\left(\sum_{j=1}^{hk} |b_{1,j}| \cdot 1\right)^2 \leq \sum_{j=1}^{hk} |b_{1,j}|^2 \sum_{j=1}^{hk} 1 = \sum_{j=1}^{hk} |b_{1,j}|^2 \cdot (hk) = hk \cdot \|b_1\|^2$$

Se, além disso, considerarmos c tal que $b_1 = cM$ obtemos

$$\begin{aligned}
\|b_1\| &\geq \frac{1}{\sqrt{hk}} \sum_{j=1}^{hk} |b_{1,j}| \\
&= \frac{1}{\sqrt{hk}} \left(\left| \sum_{i=1}^{hk} c_i m_{i,1} \right| + \left| \sum_{i=1}^{hk} c_i m_{i,2} \right| + \cdots + \left| \sum_{i=1}^{hk} c_i m_{i,hk} \right| \right) \\
&= \frac{1}{\sqrt{hk}} \left(\left| \sum_{i=1}^{hk} c_i a_{i,1} \right| + \left| \left(\sum_{i=1}^{hk} c_i a_{i,2} \right) X \right| + \cdots + \left| \left(\sum_{i=1}^{hk} c_i a_{i,hk} \right) X^{hk-1} \right| \right) \\
&\geq \frac{1}{\sqrt{hk}} |r(x)| \text{ para todo o } |x| \leq X, \qquad (3.5)
\end{aligned}$$

onde

$$\begin{aligned}
r(x) &= \sum_{i=1}^{hk} c_i a_{i,1} + \left(\sum_{i=1}^{hk} c_i a_{i,1} \right) x + \cdots + \left(\sum_{i=1}^{hk} c_i a_{i,hk} \right) x^{hk-1} \\
&= c_1 \sum_{j=1}^{hk} a_{1,j} x^{j-1} + c_2 \sum_{j=1}^{hk} a_{2,j} x^{j-1} + \cdots + c_{hk} \sum_{j=1}^{hk} a_{hk,j} x^{j-1} \\
&= c_1 q_{u,v}(x) + c_2 q_{u,v}(x) + \cdots + c_{hk} q_{u,v}(x). \qquad (3.6)
\end{aligned}$$

Assim, $\|b_1\|$ pode ser visto como um "limite superior" para o polinómio $r(x)$ no domínio $|x| \leq X$. Observemos que se $f(x_0) \equiv 0 \ (mod\ N)$ então $r(x_0) = 0 \ (mod\ N^{h-1})$ pois $q_{u,v}(x_0) \equiv 0 \ (mod\ N^{h-1})$, para todo o u, v.

Juntando as relações 3.4 e 3.5 obtidas para b_1 conclui-se que podemos construir um polinómio $r(x)$ que satisfaz $r(x_0) = 0 \ (mod\ N^{h-1})$ sempre que x_0 é tal que $f(x_0) \equiv 0 \ (mod\ N)$ e

$$|r(x)| \leq \left(2^{(hk-1)/4} \sqrt{hk}\right) X^{(hk-1)/2} N^{(h-1)/2}, \text{ para todo o } |x| \leq X.$$

Depois de construído o polinómio $r(x)$, descrito em (3.6), escolhe-se

$$X \leq \left(2^{-1/2} (hk)^{-1/(hk-1)}\right) N^{(h-1)/(hk-1)} \qquad (3.7)$$

[3]**Desigualdade de Cauchy**: $\left(\sum_{k=1}^{n} a_k b_k\right)^2 \leq \sum_{k=1}^{n} a_k^2 \sum_{k=1}^{n} b_k^2$ com a_k e b_k sucessões de números reais quaisquer.

para se ter

$$|r(x)| < N^{h-1} \text{ para todo } |x| \leq X,$$

o que implica que se $r(x_0) \equiv 0 \ (mod\ N^{h-1})$ então $r(x_0) = 0$ sobre os inteiros, para cada x_0 tal que $|x_0| \leq X$. Como facilmente se determinam as raízes inteiras de um polinómio de coeficientes inteiros[4], obtêm-se assim todos os $|x_0| \leq X$ tais que $r(x_0) = 0$. Resta agora verificar se de facto satisfazem $f(x_0) = 0 \ (mod\ N)$.

Note-se ainda que h foi tomado como um inteiro qualquer maior que 1 e quando $h \to \infty$ temos que o majorantes de X em (3.7) tende para $2^{-1/2}N^{1/k}$, portanto podemos escolher qualquer X tal que $X < \frac{1}{\sqrt{2}}N^{1/k}$.

O tempo estimado para determinar todas as raízes "pequenas" de $f(x) \equiv 0 \ (mod\ N)$ é dominado pela implementação do algoritmo LLL que determina a base reduzida sendo, segundo Howgrave-Graham em [HG98, p. 82], $O(h^9k^6 \log^3 N)$. □

Para ilustrar como é aplicado o método vamos dar o exemplo apresentado por Howgrave-Graham em [HG98, p. 77]

Exemplo 3.2.5. Seja $f(x) = x^2 + 14x + 19$. Pretende-se determinar todas as raízes inteiras de $f(x) \equiv 0 \ (mod\ 35)$ considerando $h = 3$ ($k = 2$, pois é o grau do polinómio $f(x)$ e $X = 2$ de forma a verificar a condição 3.7). O método foi provado para o domínio $|x_0| < \frac{1}{\sqrt{2}}N^{1/k}$, mas existem casos, como o exemplo que se segue, em que o método descrito ainda encontra raízes fora do intervalo $\left]-\frac{1}{\sqrt{2}}N^{1/k}, \frac{1}{\sqrt{2}}N^{1/k}\right[$.

[4]Por um resultado elementar: Seja $f(x) = a_nx^n + \cdots + a_0 \in \mathbb{Z}[x]$ e seja $\frac{r}{s}$ uma raiz racional de $f(x)$, com $(r, s) = 1$. Então $r|a_0$ e $s|a_n$.

Portanto basta verificar quais os números racionais $\frac{r}{s}$ tais que $r|a_0$ e $s|a_n$ são de facto raízes do polinómio.

Começamos por construir a matriz quadrada M de ordem 6:

$$M = \begin{pmatrix} 1225 & & & & & \\ 0 & 1225 \times 2 & & & & \\ 665 & 490 \times 2 & 35 \times 2^2 & & & \\ 0 & 665 \times 2 & 490 \times 2^2 & 35 \times 2^3 & & \\ 361 & 532 \times 2 & 234 \times 2^2 & 28 \times 2^3 & 2^4 & \\ 0 & 361 \times 2 & 532 \times 2^2 & 234 \times 2^3 & 28 \times 2^4 & 2^5 \end{pmatrix}$$

Através da aplicação do algoritmo LLL obtém-se a matriz reduzida

$$B = \begin{pmatrix} 3 & 8 \times 2 & -24 \times 2^2 & -8 \times 2^3 & -1 \times 2^4 & 2 \times 2^5 \\ 49 & 50 \times 2 & 0 & 20 \times 2^3 & 0 & 2 \times 2^5 \\ 115 & -83 \times 2 & 4 \times 2^2 & 13 \times 2^3 & 6 \times 2^4 & 2 \times 2^5 \\ 61 & 16 \times 2 & 37 \times 2^2 & -16 \times 2^3 & 3 \times 2^4 & 4 \times 2^5 \\ 21 & -37 \times 2 & -14 \times 2^2 & 2 \times 2^3 & 14 \times 2^4 & -4 \times 2^5 \\ -201 & 4 \times 2 & 33 \times 2^2 & -4 \times 2^3 & -3 \times 2^4 & 1 \times 2^5 \end{pmatrix}$$

onde $B = HM$ e

$$H = \begin{pmatrix} 70 & 46 & -98 & 32 & -57 & 2 \\ 73 & 48 & -104 & 32 & -56 & 2 \\ 55 & 36 & -74 & 27 & -50 & 2 \\ 125 & 82 & -171 & 60 & -109 & 4 \\ -175 & -115 & 254 & -74 & 126 & -4 \\ 41 & 27 & -59 & 18 & -31 & 1 \end{pmatrix}.$$

O polinómio pretendido pode obter-se:

- considerando o vector b_1 (1^a linha da matriz reduzida B) temos os coeficientes do polinómio $r(x)$ desde que cada uma das entradas seja dividida respectivemente por $1, 2, \cdots, 2^5$; assim $r(x) = 2x^5 - x^4 - 8x^3 - 24x^2 + 8x + 3$,
- usando as entradas de $h_1 = (\alpha_i)$, a primeira linha de H, para formar a soma

$$r(x) = \alpha_1 N^2 + \alpha_2 N^2 x + \alpha_3 N f(x) + \alpha_4 N x f(x) + \alpha_5 f^2(x) + \alpha_6 x f^2(x),$$

que é obviamente igual ao polinómio obtido no ponto anterior.

3.2.2 Ataque de Hastad

O ataque que a seguir se descreve, implementado por Hastad, é um primeiro exemplo de aplicação do teorema de Coppersmith em ataques ao sistema criptográfico RSA. Comecemos por dar um exemplo.

Exemplo 3.2.6. Suponhamos que um emissor quer enviar uma mensagem encriptada M para os receptores $R_1, R_2, ..., R_k$. Cada um dos receptores tem a sua chave pública $\langle N_i, e_i \rangle$. Vamos assumir que M é menor que qualquer dos N_i. Naturalmente o emissor encripta a mensagem M com as respectivas chaves públicas e envia o i-ésimo criptograma ao receptor R_i. Um intruso pode interceptar a ligação sem que o emissor se aperceba e coleccionar os k criptogramas.
Para simplificar, suponhamos que todos os expoentes de encriptação e_i são iguais a 3. Vamos mostrar que se $k \geq 3$, o intruso consegue recuperar M, a partir dos criptogramas $C_1, C_2, C_3, ...$, onde

$$C_1 \equiv M^3 \ (mod\ N_1)$$
$$C_2 \equiv M^3 \ (mod\ N_2)$$
$$C_3 \equiv M^3 \ (mod\ N_3).$$

Podemos assumir que $(N_i, N_j) = 1$ para todo o $i \neq j$, pois caso contrário pode-se factorizar algum dos N_i. Assim, aplicando o Teorema Chinês dos Restos a C_1, C_2 e C_3 obtém-se $C' \in \mathbb{Z}_{N_1 N_2 N_3}$ satisfazendo

$$C' \equiv M^3 \ (mod\ N_1 N_2 N_3).$$

Como $M < N_i, \forall i$ temos $M^3 < N_1 N_2 N_3$. Então $C' = M^3$ e portanto $M = \sqrt[3]{C'}$.
Mais geralmente, se os expoentes de encriptação forem todos iguais a e, pode-se recuperar M logo que se tenha $k \geq e$ onde k é o número de criptogramas interceptados. Portanto, este ataque só será bem sucedido se o expoente público e for relativamente pequeno.

Numa tentativa de prevenir este ataque podemos, em vez de encriptar a mensagem M, camuflá-la (*padding*) primeiro com um polinómio e usar expoentes de encriptação diferentes. Uma camuflagem poderá ser, por exemplo, se a mensagem M tem m-bits, aplicar a função linear $f_i(x) = i2^m + M$ à mensagem M, i.é, justapor os "bits" de i à esquerda dos de M, e encriptar $M_i = f_i(M)$.

Assim, o emissor fixaria um polinómio público $f_i \in \mathbb{Z}_{N_i}[x]$ para cada receptor $R_1, ..., R_k$ e transmitiria a encriptação de $f_i(M)$ ao receptor R_i. O que se interceptaria seria então os criptogramas

$$C_i \equiv f_i(M)^{e_i} \bmod N_i, \text{ para } i = 1, 2, ..., k.$$

O próximo teorema mostra que mesmo usando expoentes de encriptação diferentes e este tipo de camuflagem, por aplicação de funções polinomiais às mensagens, não se previne este tipo de ataque quando o número de criptogramas interceptados é suficientemente grande.

Teorema 3.2.7 (Hastad). *Sejam $N_1, N_2, ..., N_k$ inteiros dois a dois primos entre si, $\overline{N} = N_1...N_k$ e $N_{min} = \min_i(N_i)$. Sejam $g_i = f_i^{e_i} - C_i \in \mathbb{Z}_{N_i}[x]$, k polinómios de grau no máximo n, onde $n = \max_{i=1,...,k}\{e_i deg(f_i)\}$ e $f_i \in \mathbb{Z}_{N_i}[x]$ é um polinómio público que serve para camuflar (pad) a mensagem. Suponhamos que existe um único $M < N_{min}$ que satisfaz $g_i(M) \equiv 0$ (mod N_i), $\forall i = 1, ..., k$. Se $k > n$, podemos recuperar M em tempo $O(n^4 \log^3 \overline{N})$.*

Prova: Comecemos por observar que M é a mensagem a encriptar e portanto deverá ter sido formatada de forma a que $M < N_i$, $\forall i$ logo $M < \min_{i=1,...k}(N_i)$. Notemos ainda que todos os polinómios f_i devem ter o coeficiente guia invertível, pois caso contrário, i.é, se para algum i o coeficiente guia não for invertível em $\mathbb{Z}_{N_i}^*$ ficaria exposta a factorização de N_i. Assim todos os polinómios $g_i = f_i^{e_i} - C_i \in \mathbb{Z}_{N_i}[x]$ têm também o coeficiente guia invertível.
Multiplicando cada g_i pela potência de x apropriada, podemos considerar que todos

os polinómios têm grau n. Consideremos então

$$g_i(x) = \sum_{j=0}^{n} a_{ij}x^j \ (mod\ N_i), \text{ onde } a_{in} \text{ é invertível em } \mathbb{Z}_{N_i}.$$

Aplicando o Teorema Chinês dos Restos calculamos os números inteiros u_i $(i = 1, \cdots, k)$ tais que $u_i < \overline{N}$ com $i = 1, ..., k$ e (a demonstração do Teorema Chinês dos Restos incluída no apêndice B fornece um método de calcular tais números):

$$u_i \equiv \begin{cases} 1 \pmod{N_i} \text{ se } & i = j \\ 0 \pmod{N_j} \text{ se } & i \neq j. \end{cases}$$

Seja $G(x) \in \mathbb{Z}_{\overline{N}}[x]$ o polinómio definido por

$$G(x) = \sum_{j=1}^{k} u_j g_j(x).$$

Tem-se então,

$$G(M) \equiv \sum_{j=1}^{k} u_j g_j(M) \equiv g_i(M) \equiv 0 \ (mod\ N_i), \forall i$$

o que implica que $G(M) \equiv 0 \ (mod\ \overline{N})$, porque $(N_i, N_j) = 1$, $\forall i \neq j$.
O coeficiente guia de $G(x)$ é $a_{in} \ (mod\ N_i)$, para todo i porque $\sum_{j=1}^{k} u_j a_{jn} \equiv a_{in} \ (mod\ N_i)$. Como $(a_{in}, N_i) = 1$, $\forall i = 1, \cdots, k$, resulta que o coeficiente guia de $G(x)$ é invertível módulo $\overline{N}$. Multiplicando $G(x)$ pelo inverso do coeficiente guia e reduzindo $mod\ \overline{N}$ obtemos um polinómio mónico $\widetilde{G}(x)$ de grau n tal que $\widetilde{G}(M) \equiv 0 \ (mod\ \overline{N})$ e além disso $M < N_{\min} \leq \overline{N}^{1/k} < \overline{N}^{1/n}$. Estamos nas condições de aplicar o Teorema de Coppersmith e recuperar M, uma raiz inteira do polinómio mónico $\tilde{G}(M) \ (mod\ \overline{N})$.
O tempo estimado para recuperar a mensagem M é dominado pelo tempo de aplicar o Teorema de Coppersmith que é $O(n^4 \log^3 \overline{N})$. □

3.2.3 Ataque a mensagens camufladas que usam o mesmo módulo

Suponhamos que um emissor envia uma mensagem M camuflada (*paded*) a um receptor, onde a camuflagem é feita aleatoriamente acrescentando uns "bits" à mensagem

original, antes de ser encriptada. Suponhamos ainda que um intruso intercepta o criptograma enviado e impede que este chegue ao receptor. O emissor vendo que o receptor não responde à mensagem enviada faz uma nova tentativa. Camufla novamente a mensagem M, encripta com a mesma chave pública e envia o respectivo criptograma, que é novamente interceptado pelo intruso. O intruso tem então em seu poder dois criptogramas correspondentes a duas codificações da mesma mensagem M usando duas camuflagens aleatórias diferentes.

O que o seguinte teorema mostra é que embora o intruso não consiga descobrir a camuflagem, que neste caso supomos ser uma função linear $f_i(x) = 2^m x + r_i$ onde m e cada r_i estão nas condições descritas no teorema seguinte, ele pode recuperar a mensagem inicial. Como se verá no fim desta secção este ataque só é possível se o expoente público e for pequeno, pelo que a prova completa só será feita para o caso em que $e = 3$.

Teorema 3.2.8. *Seja $\langle N, e\rangle$ a chave pública RSA, onde N tem n-bits de comprimento. Sejam $m = [\frac{n-1}{e^2}]$ e $M \in \mathbb{Z}_N^*$ uma mensagem com no máximo (n-m)-bits. Definam-se*

$$M_1 = 2^m M + r_1 \text{ e } M_2 = 2^m M + r_2$$

onde r_1 e r_2 são inteiros distintos e $0 \leq r_1, r_2 < 2^m$. Se conhecermos $\langle N, e\rangle$ e as codificações C_1 e C_2 de M_1 e M_2 (não conhecendo r_1 e r_2) podemos recuperar M em tempo polinomial.

Prova: Sejam $g_1(x, y) = x^e - C_1$ e $g_2(x, y) = (x+y)^e - C_2$ com $g_1, g_2 \in \mathbb{Z}_N$. O número M_1 é uma raíz comum aos polinómios $g_1(x, r_2 - r_1)$ e $g_2(x, r_2 - r_1)$, pois

$$g_1(M_1, r_2 - r_1) = M_1^e - C_1 \equiv 0 \; mod \; N$$

e

$$\begin{aligned} g_2(M_1, r_2 - r_1) &= (M_1 + r_2 - r_1)^e - C_2 = (2^m M + r_1 + r_2 - r_1)^e - C_2 \\ &= (2^m M + r_2)^e - C_2 = M_2^e - C_2 \equiv 0 \; mod \; N \end{aligned}$$

Por outras palavras, $\Delta = r_2 - r_1$ é uma raíz da resultante[5] $res_x(g_1(x,y), g_2(x,y)) \in \mathbb{Z}$. Como g_1 e g_2 são polinómios de grau e sobre a variável x, então

$$h(y) = -res_x(g_1(x,y), g_2(x,y)) = \det(R)$$

onde R é a matriz quadrada de ordem $2e$

$$R = \left(\begin{array}{c|c} D & I_e \\ \hline T & P \end{array}\right),$$

onde

- $D = -C_1 I_e$ é uma matriz diagonal de ordem e, cujas entradas não depende de y;
- I_e é a matriz identidade de ordem e;
- T é uma matriz triângular superior de ordem e cujos elementos da diagonal são $y^e - C_2$;
- P é uma matriz triângular inferior de ordem e cujos elementos da diagonal são iguais a 1.

[5]**Definição** [Spi94, pp. 150-153]: Sejam $f(x) = a_0 + a_1 x + \cdots + a_n x^n$ e $g(x) = b_0 + b_1 x + \cdots + b_m x^m$ polinómios sobre um anel comutativo A tais que $a_n \neq 0$ e $b_m \neq 0$. A resultante de f e g, que é um elemento do anel A e se denota por $res(f,g)$, é o determinante da matriz quadrada de ordem $(m+n)$

$$\begin{array}{l} m \left\{ \right. \\ n \left\{ \right. \end{array} \left(\begin{array}{ccccccccccc} a_0 & a_1 & a_2 & \cdots & \cdots & a_{n-1} & a_n & 0 & \cdots & 0 \\ 0 & a_0 & a_1 & a_2 & \cdots & \cdots & a_{n-1} & a_n & \cdots & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & \cdots & a_0 & a_1 & a_2 & \cdots & \cdots & a_{n-1} & a_n \\ b_0 & b_1 & \cdots & \cdots & b_m & 0 & 0 & \cdots & \cdots & 0 \\ 0 & b_0 & b_1 & \cdots & \cdots & b_m & 0 & \cdots & \cdots & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & \cdots & 0 & b_0 & b_1 & \cdots & \cdots & b_{m-1} & b_m \end{array}\right).$$

Uma propriedade fundamental da resultante é: f, g têm raiz comum $\Leftrightarrow res(f,g) = 0$.

Ora, $\det(R) = |DP - TI_e|$, pelo teorema 3 de [Sil00], e só a matriz M contribui para o determinante com termos que dependem de y, sendo fácil ver que a matriz $DP - T$ tem como entradas polinómios em y, de grau e na diagonal principal e de grau menor fora da diagonal. Portanto $h(y)$ tem grau e^2.

Exemplo 3.2.9. Vejamos o que obtemos em $h(y) = -res_x(g_1(x,y), g_2(x,y))$ quando $e = 3$. Temos

$$g_1(x,y) = x^3 - C_1$$

$$g_2(x,y) = x^3 + 3x^2y + 3xy^2 + y^3 - C_2$$

logo

$$h(y) = -\begin{vmatrix} -C_1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -C_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -C_1 & 0 & 0 & 1 \\ y^3 - C_2 & 3y^2 & 3y & 1 & 0 & 0 \\ 0 & y^3 - C_2 & 3y^2 & 3y & 1 & 0 \\ 0 & 0 & y^3 - C_2 & 3y^2 & 3y & 1 \end{vmatrix}$$

$$= y^9 + 3y^6C_1 - 3y^6C_2 + 21y^3C_2C_1 + 3y^3C_2^2 + 3y^3C_1^2 + C_1^3 - 3C_1^2C_2 + 3C_1C_2^2 - C_2^3.$$

Logo temos um polinómio mónico na variável y de grau 3^2.

Estamos a supor que $0 \leq r_1, r_2 < 2^m$ logo $\Delta = r_2 - r_1 < 2^m$. Além disso, N tem n-bits, resultando que

$$|\Delta| < 2^m < N^{\frac{1}{e^2}}.$$

Assim, $h(\Delta) \equiv 0 \ (mod\ N)$ onde Δ é uma raiz pequena, $\Delta < N^{1/e^2}$, do polinómio mónico de grau e^2, $h(y)$. Estamos em condições de aplicar o teorema de Coppersmith e assim determinar Δ de forma eficiente.

Sabendo Δ podemos reescrever $g_2(x) = (x + \Delta)^e - C_2$ da seguinte forma:

$$g_2(x) = f(x)^e - C_2$$

$f(x) = x + \Delta$ e portanto $M_2 = f(M_1)$. Para concluir a demonstração basta aplicar o Lema de Franklin-Reiter que se apresenta de seguida.

Lema 3.2.10 (Franklin-Reiter). *Seja $\langle N, e\rangle$ uma chave pública RSA. Sejam $M_1, M_2 \in \mathbb{Z}_N^*$ distintos satisfazendo $M_2 = f(M_1)$ mod N para algum polinómio linear $f(x) = ax + b \in \mathbb{Z}_N[x]$ com $b \neq 0$. Então dado $\langle N, e, C_1, C_2, f\rangle$, pode-se recuperar M_1 e M_2 em tempo polinomial.*

Prova: Comecemos por observar que M_1 é raiz do polinómio

$$g_2(x) = f(x)^e - C_2 \in \mathbb{Z}_N[x],$$

pois $C_2 \equiv M_2^e \ mod\ N$, e também é raiz do polinómio

$$g_1(x) = x^e - C_1 \in \mathbb{Z}_N[x].$$

Portanto $x - M_1$ é factor comum aos dois polinómios. Aplicando o algoritmo de Euclides determina-se o máximo divisor comum de g_1 e g_2. Se o máximo divisor comum for linear então está determinado o factor e portanto também M_1 e $M_2 = f(M_1)$.
No caso $e = 3$ temos $g_1(x) = (x - M_1)p(x)$ onde $p(x)$ é um polinómio quadrático irredutível *mod* N porque $x^3 - C_1$ tem uma única raíz, pois caso contrário teriamos para duas mensagens diferentes um mesmo criptograma, o que é garantido não acontecer dado $(e, \varphi(N)) = 1$. Como g_2 não pode dividir g_1, o $m.d.c(g_1, g_2)$ é linear.
Para $e > 3$ ou o $m.d.c.$ é linear procedendo-se da forma acima descrita ou o ataque falha. □

Este ataque só é de facto uma ameaça para e pequeno pois para valores grandes de e o cálculo de $m.d.c.(g_1, g_2)$ e $res(g_1, g_2)$ é proibitivo.

Capítulo 4

Variantes do RSA

Vamos introduzir três variantes do sistema criptográfico RSA original que têm como objectivo principal acelerar a implementação do sistema. A primeira variante, denomidada "Batch" RSA, propõe-se fazer várias desencriptações pelo custo de uma, tornando assim o processo de desencriptação mais rápido. A segunda variante, RSA com múltiplos factores, é a única variante que propõe uma mudança estrutural: mudar a estrutura do módulo para o produto de vários factores primos em vez de dois ou manter os dois primos mas sendo um deles elevado a uma potência maior que 1. Esta variante que altera a estrutura do módulo tem por objectivo não só acelerar os processos de encriptação e desencriptação mas também proteger o sistema de alguns ataques. A terceira e última variante pretende tornar o processo mais rápido, reequilibrando o esforço efectuado pela encriptação e pela desencriptação, e proteger o sistema de ataques a expoentes públicos pequenos.

4.1 "Batch" RSA

O número de utilizadores do "World Web Wide" (WWW) e diversidade de páginas disponíveis na "Web" tem aumentado muito nos últimos anos. Nos dias de hoje é possível através da Internet comunicar, pesquisar, consultar, comprar, pagar contas,

fazer transferências bancárias ou aplicações financeiras. Algumas páginas "web", como por exemplo as de comércio electrónico ou de aplicações financeiras, têm de garantir a privacidade e autenticidade das suas comunicações usando para isso protocolos de segurança. Infelizmente os servidores que protegem as suas comunicações tornam-se muitos mais lentos dos que não usam qualquer tipo de segurança, o que obriga a grandes investimentos em "hardware" de forma a que o tempo de resposta ao utilizador seja aceitável. O que descrevemos nesta secção é um algoritmo que permite acelerar o tempo de resposta ao utilizador sem ser necessário qualquer tipo de "hardware" específico. A ideia base do que iremos descrever é o servidor aguardar até receber b pedidos de acesso por b clientes diferentes e tratá-los como se fosse apenas um.

Suponhamos que o protocolo de segurança utilizado pelo servidor é o sistema criptográfico RSA. O processo de desencriptação do RSA original usa a exponenciação e o produto modulares. Para garantir a segurança do sistema o expoente privado deve ser suficientemente grande, o que torna o processo de desencriptação lento. A variante "Batch" RSA propõe uma forma de acelerar esse processo, fazendo a desencriptação de vários criptogramas pelo custo de um.

Exemplo 4.1.1. Suponhamos que C_1 é o criptograma que se obtém quando encriptada a mensagem M_1 com a chave pública $\langle N, 3\rangle$, e C_2 o criptograma que se obtém da encriptação de M_2 com a chave pública $\langle N, 5\rangle$. Para desencriptar os criptogramas é necessário determinar $C_1^{1/3}\ mod\ N$ e $C_2^{1/5}\ mod\ N$, onde 1/3 e 1/5 representam, neste contexto, os inversos de 3 e 5 $mod\ \varphi(N)$, respectivamente. Se fizermos $A \equiv (C_1^5 \cdot C_2^3)^{1/15}\ (mod\ N)$ obtém-se

$$C_1^{1/3} \equiv \frac{A^{10}}{C_1^3 \cdot C_2^2}\ (mod\ N),\ \text{pois}\ A^{10} \equiv \left[(C_1^5 \cdot C_2^3)^{1/15}\right]^{10} \equiv C_1^{1/3} \cdot C_1^3 \cdot C_2^2\ (mod\ N)$$

e

$$C_2^{1/5} \equiv \frac{A^6}{C_1^2 \cdot C_2}\ (mod\ N),\ \text{pois}\ A^6 \equiv \left[(C_1^5 \cdot C_2^3)^{1/15}\right]^6 \equiv C_1^2 \cdot C_2^{1/5} \cdot C_2\ (mod\ N).$$

Portanto pelo custo de calcular apenas a 15ª raiz $mod\ N$ e mais algumas operações aritméticas consegue-se desencriptar C_1 e C_2. Como calcular a 15ª raiz leva o mesmo tempo que uma desencriptação temos então duas desencriptações pelo custo de uma.

Fiat, em [Fia89], descreve como funciona este método para um número qualquer de utilizadores. O processo de desencriptação com "batch" usa árvores binárias em que cada nó tem sempre dois filhos. Seja N o módulo RSA e b o tamanho do "batch", ou seja, temos b expoentes de encriptação, e_1, e_2, ... e_b, todos primos com $\varphi(N)$ e primos entre si dois a dois. Por exemplo, e_1, e_2, ... e_b podem ser os b primeiros números ímpares que não dividem $\varphi(N)$.

Pelo que vimos nos capítulos anteriores pode parecer imprudente o facto de se estar a usar o mesmo módulo e expoentes públicos pequenos. Note-se que quem gera as chaves é o servidor, portanto tem que ter várias chaves públicas à disposição para que os seus clientes acedam ao sistema em segurança. Só o servidor conhece as respectivas chaves privadas, logo a factorização do módulo. Quanto aos expoentes públicos recordemos que os ataques descritos no capítulo anterior só se aplicam quando estamos a encriptar várias vezes a mesma mensagem, algo que aqui não acontece pois aos b utilizadores diferentes correspondem b mensagens diferentes.

Assim, dados os criptogramas C_1, C_2, ..., C_b o objectivo é gerar as b raízes

$$C_1^{1/e_1} \ (mod\ N),\ C_2^{1/e_2} \ (mod\ N),\ ...,\ C_b^{1/e_b} \ (mod\ N)$$

pelo custo de uma.

O processo divide-se em três passos que passamos a descrever:

1° Passo: Construção do produto (figura 4.1)
Começar por construir uma árvore binária $\mathcal{A}$ cujas folhas (os nós sem filhos) são etiquetadas com os expoentes públicos e_1, e_2, ... e_b e os respectivos criptogramas $C_1, C_2, ..., C_b$. A árvore deverá ser construída das folhas para a raiz, portanto de baixo para cima, de forma a que cada nó tenha exactamente dois filhos e tal que $W = \sum_{i=1}^{b} t_i \log e_i$ seja mínimo, onde t_i denota a profundidade da folha e_i[1].

[1]Em [Knu73, p. 402] Knuth descreve um algoritmo proposto por Huffman para construir uma árvore nestas condições.

Seja $E = \prod_{i=1}^{b} e_i$. Usando a árvore $\mathcal{A}$ obtém-se o produto

$$C \equiv C_1^{E/e_1} C_2^{E/e_2} \ldots C_b^{E/e_b} \ (mod\ N)$$

trabalhando das folhas para a raiz através do seguinte processo recursivo: a cada nó interno associar os produtos $E = E_L \cdot E_R$ e $C \equiv C_L^{E_R} \cdot C_R^{E_L} \ (mod\ N)$ onde E_L e E_R são o produto dos expoentes públicos associados respectivamente aos nós filhos do ramo esquerdo e ramo direito e C_L e C_R são o produto dos criptogramas associados igualmente aos nós filhos do ramo esquerdo e ramo direito, respectivamente. C é simplesmente o resultado associado à raiz. O esquema da figura que se segue mostra como se obtém um nó dos dois nós filhos.

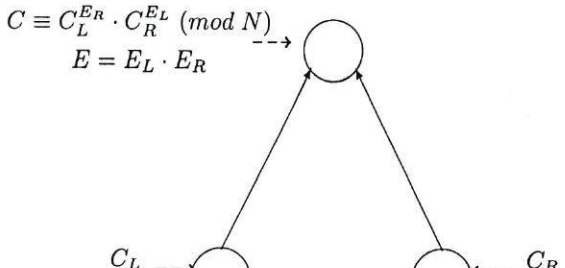

Figura 4.1: Construção do produto

2°Passo: Extracção da raiz

Extrair a E-ésima raiz de C, calculando o inverso de E $(mod\ \varphi(N))$ para obter

$$C^{1/E} \equiv C_1^{1/e_1} \cdot C_2^{1/e_2} \cdots C_b^{1/e_b} \ (mod\ N).$$

3° Passo: Decomposição em factores (figura 4.2)

As raízes pretendidas são os factores de $C^{1/E}$. Portanto, o objectivo é, novamente através de uma árvore binária construída agora da raiz para as folhas (de cima para

baixo) e usando a árvore construída no 1° passo, determinar a decomposição de $C^{1/E}$ nos b factores $C_1^{1/e_1}, C_2^{1/e_2}, \cdots, C_b^{1/e_b}$. Começa-se por decompor $C^{1/E}$ no produto dos dois factores correspondentes aos nós filhos da primeira árvore que estão na mesma posição. Cada um destes nós filhos será novamente factorizado no produto dos dois factores que se encontram na posição correspondente na primeira árvore, até se obter os b números pretendidos. Para isso, seja $k = \left[\frac{b}{2}\right]$ e começamos por determinar X, associado à raiz da árvore tal que:

$$\begin{cases} X \equiv 0 \ (mod\ e_1) \\ X \equiv 0 \ (mod\ e_2) \\ \quad \vdots \\ X \equiv 0 \ (mod\ e_k) \\ X \equiv 1 \ (mod\ e_{k+1}) \\ X \equiv 1 \ (mod\ e_{k+2}) \\ \quad \vdots \\ X \equiv 1 \ (mod\ e_b). \end{cases}$$

Através do Teorema Chinês dos Restos determinamos a solução única $X\ mod\ \prod_{i=1}^{b} e_i$ deste sistema. Usando a notação do 1° passo, o valor de X associado a cada nó filho é a solução única do seguinte sistema:

$$\begin{cases} X \equiv 0 \ (mod\ E_L) \\ X \equiv 1 \ (mod\ E_R). \end{cases}$$

Existem então X_L e X_R tais que

$$\begin{aligned} X &= E_L X_L, \\ X - 1 &= E_R X_R \end{aligned}$$

Calcula-se de seguida a X-ésima potência módulo N de $M \equiv C^{1/E}\ (mod\ N)$ obtendo:

$$\begin{aligned}
\left(C^{1/E}\right)^X &\equiv \left(\prod_{i=1}^{b} C_i^{1/e_i}\right)^X \\
&\equiv \left(\prod_{i=1}^{k} C_i^{1/e_i}\right)^X \left(\prod_{i=k+1}^{b} C_i^{1/e_i}\right)^{X-1} \prod_{i=k+1}^{b} C_i^{1/e_i} \\
&\equiv \left(\prod_{i=1}^{k} C_i^{1/e_i}\right)^{E_L X_L} \left(\prod_{i=k+1}^{b} C_i^{1/e_i}\right)^{E_R X_R} \prod_{i=k+1}^{b} C_i^{1/e_i} \\
&\equiv C_L^{X_L}.C_R^{X_R}.\prod_{i=k+1}^{b} C_i^{1/e_i}\ (mod\ N).
\end{aligned}$$

M e X são os valores associados à raiz da nova árvore. Para obter o produto $M_R = \prod_{i=k+1}^{b} C_i^{1/e_i}$, que é a factorização de M associada ao filho direito, basta calcular $C_L^{X_L}$, $C_R^{X_R}$ e usar a igualdade anterior, ou seja, $M_R \equiv \frac{M^X}{C_R^{X_R} C_L^{X_L}}\ (mod\ N)$. C_L e C_R são os valores associados aos filhos esquerdo e direito correspondentes ao nó da árvore construída no 1º passo que se encontra na mesma posição. Para determinar o outro produto, a factorização associada ao filho do lado esquerdo, basta dividir $C^{1/E}$ por $\prod_{i=k+1}^{b} C_i^{1/e_i}$, i.é, $M_L = \frac{M}{M_R}$. O processo repete-se como esquematizado na figura 4.2 sobre os valores obtidos e é fácil ver que se obtém a factorização pretendida.

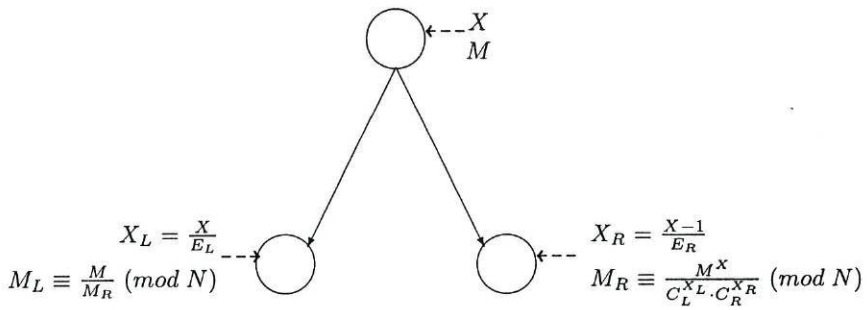

Figura 4.2: Decomposição do produto

Exemplo 4.1.2. Vejamos como são construídas as árvores num exemplo que pretende desencriptar 7 mensagens pelo custo de uma, com expoentes públicos pequenos.

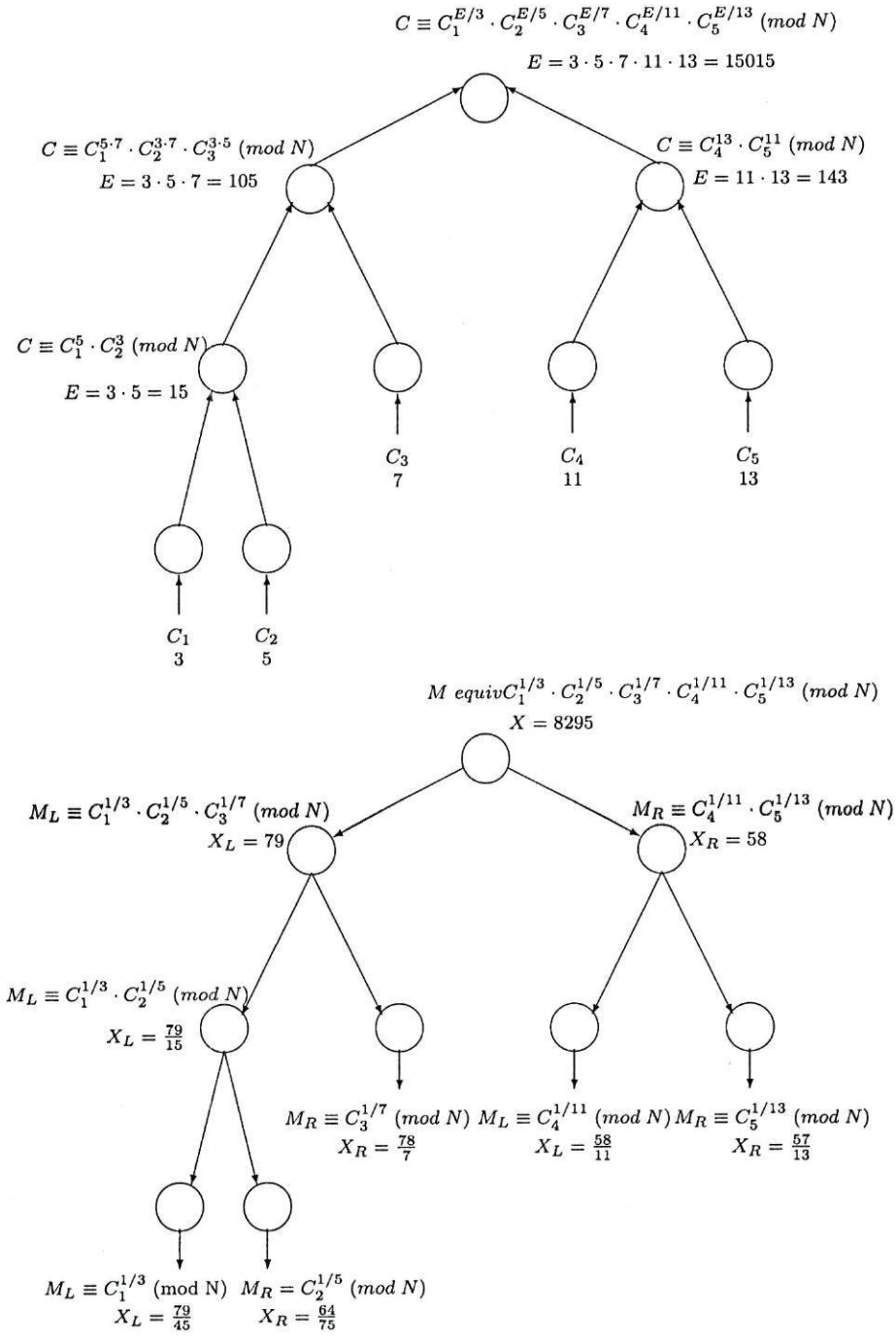

Figura 4.3: Exemplo de árvores construidas quando utilizada a variante "batch"RSA.

Esta técnica de desencriptar b critpogramas de uma só "fornada" (*batching*) só vale a pena quando os expoentes públicos são pequenos, pois caso contrário as operações aritméticas adicionais tornam-se demasiado dispendiosas. Além disso, só podemos usar a desencriptação com "batch" se estivermos a usar o mesmo módulo e expoentes públicos primos entre si. Se as chaves públicas forem iguais o processo não funciona como é mostrado no apêndice A de [SB01] usando resultados de Teoria de Galois.

Esta variante do sistema criptográfico RSA aplica-se, por exemplo, ao protocolo de segurança "Secure Socket Layer" (SSL). Para ter uma ideia de quanto é mais rápido o processo de desencriptação com "batch", apresentamos na seguinte tabela alguns resultados obtidos experimentalmente (com um Intel Pentium III, 700MHz e 256 MB RAM) por Boneh e Shacham em [SB01].

tamanho do "batch"	tamanho do módulo N		
	512	1024	2048
sem "batch"	1,53	8,38	52,96
2	1,22	5,27	29,43
4	0,81	3,18	16,41
8	0,70	2,42	10,81

Tabela 4.1: tempo de desencriptação RSA por utilizador, em milisegundos, em função dos tamanhos do "batch" e da chave.

4.2 RSA com múltiplos factores

Uma outra forma de tornar o processo de desencriptação mais rápido que no RSA original é alterar a estrutura do módulo N. Iremos considerar duas alterações da estrutura do módulo: uma em que N é o produto de dois números primos mas onde um deles é elevado a uma potência, i.é, $N = p^r \cdot q$ e por isso será denominado de RSA multi-potência, e outro em que N é o produto de pelo menos três primos, i.é,

$N = p_1 \cdot p_2 \cdot ... \cdot p_r$, $r \geq 3$, a que chamaremos RSA multi-primos.

4.2.1 RSA multi-potência: $p^r \cdot q$

Comecemos por descrever os processos de encriptação e desencriptação para então ver porque é mais rápida a desencriptação dos criptogramas, relativamente ao RSA original.

Para que o tamanho de $N = p^r \cdot q$ seja n-bits e sabendo que os primos p e q devem ser próximos para ter maior segurança, cada um deles deverá ter $\lfloor \frac{n}{r+1} \rfloor$-bits. Tal como no RSA original, tem-se um expoente público e primo com $\varphi(N)$, para gerar a chave privada determina-se d tal que $ed \equiv 1 \ (mod \ \varphi(N))$ e, além disso, determinam-se os menores inteiros positivos d_p e d_q tais que

$$d_p \equiv d \ (mod \ p-1) \text{ e } d_q \equiv d \ (mod \ q-1).$$

A chave pública é então $\langle N, \ e \rangle$ e a chave privada $\langle p, \ q, \ d_p, \ d_q \rangle$.

O processo de encriptação funciona exactamente como no RSA original, $C \equiv M^e \ (mod \ N)$, mas para desencriptar um criptograma C procede-se de forma um pouco diferente porque a chave privada a usar será da forma $\langle p, q, d_p, d_q \rangle$.

Começa-se por calcular $M_p \equiv C^{d_p} \ mod \ p$ e $M_q \equiv C^{d_q} \ mod \ q$, de onde se obtém $M_p^e \equiv C \ mod \ p$ e $M_q^e \equiv C \ mod \ q$. Partindo de $M_p \ mod \ p$ e usando um método conhecido que iremos descrever a seguir, constrói-se $M_p' \equiv M_p \ (mod \ p)$ tal que $\left(M_p'\right)^e \equiv C \ (mod \ p^r)$ e portanto $M_p' \equiv C^d \ (mod \ p^r)$. Usando o Teorema Chinês dos Restos determina-se $M \in \mathbb{Z}_N$ tal que

$$\begin{cases} M \equiv M_p' \ (mod \ p^r) \\ M \equiv M_q \ (mod \ q) \end{cases}$$

Então $M \equiv C^d \ mod \ N$ que é a desencriptação de C.

Falta então descrever o método pelo qual se obtém M_p' tal que $\left(M_p'\right)^e \equiv C \ (mod \ p^r)$ partindo de $M_p \ (mod \ p)$.

Observemos que $M'_p \ (mod\ p^r)$ pode ser escrito na sua expansão p-ádica

$$M'_p \equiv K_0 + K_1 p + K_2 p^2 + ... + K_{r-1} p^{r-1} \ (mod\ p^r) \tag{4.1}$$

e apliquemos um método conhecido (ver [Vin77, pp. 76-78]) que nos permite determinar uma solução de $x^e \equiv C \ (mod\ p^r)$ conhecendo uma solução de $x^e \equiv C \ (mod\ p^{r-1})$. Sabemos que M_p é uma solução de $x^e \equiv C \ (mod\ p)$ e queremos determinar uma solução de $x^e \equiv C \ (mod\ p^2)$ da forma $M_p + pK_1$ com $K_1 \in \mathbb{Z}$.

Ora,

$$(M_p + pK_1)^e \equiv M_p^e + p \cdot e \cdot K_1 M_p^{e-1} \equiv C \ (mod\ p^2)$$

que é equivalente a

$$\frac{M_p^e - C}{p} + eK_1 M_p^{e-1} \equiv 0 \ (mod\ p).$$

Como $\left(eM_p^{e-1}, p\right) = 1$ a congruência tem solução e esta é única.

O processo repete-se agora com vista a determinar K_2 tal que $((M_p + pK_1) + p^2 K_2)^e \equiv C \ (mod\ p^3)$. De forma semelhante ao feito anteriormente

$$\begin{aligned} & (M_p + pK_1)^e + p^2 \cdot eK_2 (M_p + pK_1)^{e-1} \equiv C \ (mod\ p^3) \\ \Leftrightarrow \quad & \tfrac{(M_p + pK_1)^e - C}{p^2} + eK_2 (M_p + pK_1)^{e-1} \equiv 0 \ (mod\ p), \end{aligned}$$

que como $(e(M_p + pK_1)^{e-1}, p) = 1$ porque $(e, p) = (M_p^{e-1}, p) = 1$ tem solução (única).

Assumindo que já foram determinados os valores $K_1, K_2, ..., K_{i-1}$ tal que $A_{i-1} = M_p + pK_1 + p^2 K_2 + \cdots + p^{i-1} K_{i-1}$ é solução de $x^e \equiv C \ (mod\ p^i)$, queremos determinar uma solução de $x^e \equiv C \ (mod\ p^{i+1})$ da forma $A_{i-1} + p^i K_i$. Temos

$$\begin{aligned} & (A_{i-1} + p^i K_i)^e \equiv C \ (mod\ p^{i+1}) \\ \Leftrightarrow \quad & A_{i-1}^e + p^i \cdot eK_i A_{i-1}^{e-1} \equiv C \ (mod\ p^{i+1}) \\ \Leftrightarrow \quad & \tfrac{A_{i-1}^e - C}{p^i} + eK_i A_{i-1}^{e-1} \equiv 0 \ (mod\ p) \end{aligned}$$

e como $(A_{i-1}^{e-1}, p) = 1$ a congruência tem solução.

O processo repete-se até $i = r - 1$ obtendo-se assim uma solução M'_p de $x^e \equiv C \ (mod\ p^r)$.

Takagi, em [Tak98], resume o processo de desencriptação no seguinte algoritmo, onde $M'_p (mod\ p^r)$ está escrito na forma (4.1):

Dados de entrada (*input*): d, p, q, e, r, C

Resultado (*output*): M

(1) $d_p := [d]_{p-1}$, $d_q := [d]_{q-1}$;

(2) $K_0 := [C^{d_p}]_p$, $M_q := [C^{d_q}]_q$;

(3) $A_0 := K_0$;

Para $i = 1$ até $r - 1$ fazer

$F_i := [A_{i-1}^e]_{p^{i+1}}$;

$E_i := [C - F_i]_{p^{i+1}}$;

$B_i := \frac{E_i}{p^i}$ em $\mathbb{Z}$;

$K_i := [(eF_i)^{-1} A_{i-1} B_i]_p$;

$A_i := A_{i-1} + p^i K_i$ em $\mathbb{Z}$;

(4) $M'_p := A_{r-1}$;

(5) $p_1 := [(p^r)^{-1}]_q$ e $q_1 := [q^{-1}]_{p^r}$;

(6) $M := [q_1 q M'_p + p_1 p^r M_q]_{p^r q}$.

Para verificar que de facto este processo de desencriptação é mais rápido observe-se que no RSA original se fazem duas exponenciações módulo $\left(\frac{n}{2}\right)$-bits (ver secção 1.4). No RSA multi-potência fazem-se também as duas exponenciações mas módulo números com $\left(\frac{n}{r+1}\right)$-bits, passo (2) do algoritmo de Takagi, e como os outros passos são mais rápidos[2] que a exponenciação, a complexidade deste método é dominada por essas duas exponenciações. Se usarmos o algoritmo para a exponenciação modular tal como descrito na secção 1.4 para calcular $x^d \ mod \ p$ ele levará tempo $O(\log d \log^2 p)$. Como d e p são da mesma ordem o tempo será estimado por $O(\log^3 p)$. Assim o RSA multi-potência relativamente ao RSA original será mais rápido aproximadamente

$$\frac{2\left(\frac{n}{2}\right)^3}{2\left(\frac{n}{r+1}\right)^3} = \frac{(r+1)^3}{8}.$$

Portanto quando $r = 2$, para o RSA de 1024 bits a rapidez aumentará aproximadamente 3,38 vezes.

[2]Esses passos envolvem basicamente resolver r-1 congruências lineares (algoritmo de Euclides) e parte-se do princípio que o expoente e deverá ser próximo de uma potência de 2, por exemplo $2^{16}+1$; os passos (5) e (6) não são mais que uma aplicação do Teorema Chinês dos Restos.

Quanto à segurança, esta variante depende tal como no RSA original da dificuldade de factorização do módulo $N = p^r q$. Para manter o RSA com um módulo de 1024 bits, r deverá ser no máximo 2 ficando p e q com 341 bits cada. Se $r = 3$ teríamos p e q com 256 bits cada, o que não é seguro pois estariam dentro dos limites de factorização pelo método das curvas elípticas (Elliptic Curve Method(ECM), actualmente um dos melhores algoritmos de factorização). Boneh, Durfee e Howgrave-Graham em [BDHG99] afirmam que o RSA multi-potência deve ser tratado com cuidado, principalmente para valores grandes de r e mostram que se $r \approx \sqrt{\log p}$, N pode ser factorizado usando um algoritmo de factorização que assimptóticamente é ainda mais rápido que o ECM, para números deste tipo.

4.2.2 RSA multi-primos: $p_1 \cdot p_2 \cdot \ldots \cdot p_r$

Tal como na secção anterior, vamos começar por descrever os processos de encriptação e desencriptação desta variante do RSA.

As chaves públicas e privadas são, tal como no RSA original, $\langle N,\ e\rangle$ e $\langle N,\ d\rangle$, respectivamente, onde $N = p_1 p_2 ... p_r$. Logo $\varphi(N) = \prod_{i=1}^{r}(p_i - 1)$ e se N tem n-bits, cada p_i deverá ter $\lfloor \frac{n}{r} \rfloor$-bits. A encriptação funciona exactamente como no RSA original. A desencriptação faz-se usando o Teorema Chinês dos Restos. Seja $d_{p_i} \equiv d\ (mod\ p_i - 1)$. Para desencriptar um criptograma C começamos por calcular $M_i \equiv C^{d_{p_i}}\ (mod\ p_i)$ para cada $i = 1,\ ...,\ r$. Aplicando o teorema Chinês dos Restos determina-se o único M tal que $M \equiv C^d\ (mod\ N)$.

Também aqui o processo de desencriptação é mais rápido. Os cálculos em que se usa o Teorema Chinês dos Restos (cuja complexidade é igual à do algoritmo de Euclides) são insignificantes quando comparados com as r exponenciações. O RSA original que usa igualmente o Teorema Chinês dos Restos na desencriptação requer duas exponenciações módulo um número com $(\frac{n}{2})$-bits (ver secção 1.4). No caso do RSA multi-primos são feitas r exponenciações módulo um número com $\lfloor \frac{n}{r} \rfloor$-bits. Tal como vimos anteriormente o tempo estimado para cada uma das exponenciações é

cúbico sobre o tamanho do módulo e portanto o aumento de velocidade será

$$\frac{2\left(\frac{n}{2}\right)^3}{r\left(\frac{n}{r}\right)^3} = \frac{r^2}{4}.$$

Para um RSA com módulo de 1024 bits e $r = 3$, esta variante do RSA é aproximadamente 2,25 vezes mais rápida que RSA original.

A segurança desta variante do RSA depende tal como no RSA original, da dificuldade de factorizar $N = p_1 \cdot p_2 \cdot \ldots \cdot p_r$ com $r > 2$. Os algoritmos de factorização não ganham nada com esta nova estrutura do N se tivermos o cuidado, tal como no RSA original, de escolher cada um dos primos p_i suficientemente grande. Por exemplo, se queremos N com 1024 bits, r deverá ser no máximo 3 pois se $r = 4$ cada um dos números primos terá 256 bits, que tal como atrás referido, estão dentro dos limites do método de factorização por curvas elípticas (ECM). Em contrapartida, como para maior parte dos ataques que se conhecem ao RSA é crucial que N seja exactamente o produto de dois primos, ao usarmos mais que dois factores estamos a proteger o sistema desses mesmos ataques.

Hinek, Low e Teske em [HLT02] apresentam um estudo sobre a efectividade dos ataques descritos no artigo [Bon99] e aqui apresentados no capítulo 3, quando aplicados a esta variante e concluem que muitos deles deixam de ser eficazes.

4.3 RSA reequilibrado

No RSA original, a encriptação e a verificação de assinaturas são mais rápidas que a desencriptação e a geração de assinaturas, respectivamente, onde assinar uma mensagem significa encriptá-la com a chave privada do emissor para que o receptor possa desencriptá-la com a chave pública do emissor, verificando assim a sua origem (ver secção 2.3). Em algumas aplicações do RSA o desejado é precisamente o inverso. Por exemplo, um telemóvel que precisa de gerar uma assinatura RSA que vai ser testada num servidor muito mais rápido, seria preferível que assinar fosse mais rápido do que verificar.

Wiener [Wie90] apresentou uma variante do RSA com o objectivo de reequilibrar (*rebalance*) a dificuldade de desencriptar e a facilidade da encriptação passando algum do trabalho da desencriptação para a encriptação. Note-se que não podemos simplesmente acelerar a desencriptação usando um expoente privado d pequeno pois se $d < \frac{1}{3}N^{1/4}$ o sistema torna-se inseguro (ataque descrito na secção 3.1). O reequilíbrio consegue-se usando um d grande mas tal que $d \ mod \ p-1$ e $d \ mod \ q-1$ sejam pequenos. Vejamos então como são as chaves pública e privada, que deverão ser geradas com especial cuidado, e como funcionam os processos de encriptação e desencriptação desta variante, RSA reequilibrado.

Sejam n e k dois inteiros tais que $k \leq \frac{n}{2}$. Geram-se dois primos distintos p e q com $\left(\frac{n}{2}\right)$-bits cada e tais que $m.d.c.(p-1,\ q-1) = 2$ e $N = pq$. Depois escolhem-se aleatoriamente d_p e d_q, com k-bits cada, tais que $(d_p, p-1) = 1$ e $(d_q, q-1) = 1$. Determina-se d tal que

$$\begin{cases} d \equiv d_p \ (mod\ p-1) \\ d \equiv d_q \ (mod\ q-1). \end{cases}$$

Para resolver o sistema seria natural usar o Teorema Chinês dos Restos, o que não é possível pois $p-1$ e $q-1$ não são primos entre si. Mas $\frac{p-1}{2}$ e $\frac{q-1}{2}$ são dois números primos entre si e podemos aplicar o Teorema Chinês dos Restos para determinar d' tal que

$$\begin{cases} d' \equiv \frac{d_p-1}{2} \ \left(mod\ \frac{p-1}{2}\right) \\ d' \equiv \frac{d_q-1}{2} \ \left(mod\ \frac{q-1}{2}\right) \end{cases} \text{ ou seja } \begin{cases} 2d'+1 \equiv d_p \ (mod\ p-1) \\ 2d'+1 \equiv d_q \ (mod\ q-1) \end{cases}.$$

Seja então $d = 2d'+1$. Assim, d é grande e d_p e d_q, que são os menores inteiros positivos tais que $d_p \equiv d \ (mod\ p-1)$ e $d_q \equiv d \ (mod\ q-1)$, respectivamente, são pequenos, portanto a chave privada pretendida é $\langle p, q, d_p, d_q \rangle$. O expoente público e é calculado em último lugar: é o inverso módulo $\varphi(N)$ (observe-se que como $(d_p, p-1) = 1$ e $(d_q, q-1) = 1$, $(d, p-1) = 1$ e $(d, q-1) = 1$). A chave pública é então $\langle e, N \rangle$ e a chave privada $\langle p, q, d_p, d_q \rangle$.

A encriptação é idêntica à do RSA original sendo a única diferença o tamanho do expoente e que passa a ser bastante maior. A desencriptação de um criptograma C

é um pouco diferente uma vez que a chave privada é $\langle p, q, d_p, d_q \rangle$. Começa-se por calcular

$$\begin{cases} M_p \equiv C^{d_p} \ mod \ p \\ M_q \equiv C^{d_q} \ mod \ q \end{cases}$$

e depois, usando o Teorema Chinês dos Restos, determina-se a mensagem original $M \in \mathbb{Z}_N$ tal que

$$\begin{cases} M \equiv M_p \ mod \ p \\ M \equiv M_q \ mod \ q \end{cases}$$

e porque $d \equiv d_p \ (mod \ p-1)$ e $d \equiv d_q \ (mod \ q-1)$

$$\begin{cases} M \equiv C^d \ mod \ p \\ M \equiv C^d \ mod \ q \end{cases} \Rightarrow M \equiv C^d \ mod \ N.$$

Para comparar o desempenho desta variante com o RSA original observemos que no RSA original o tempo de execução é dominado por duas exponenciações módulo números com $\left(\frac{n}{2}\right)$-bits cada. Nesta variante as operações mais demoradas são as duas primeiras exponenciações da desencriptação onde cada um dos expoentes tem k-bits. Como a exponenciação leva tempo linear sobre o tamanho dos expoentes, o processo é acelerado em $\frac{n/2}{k}$. Assim para um módulo N com 1024 bits e $k = 160$ temos um aumento na rapidez de execução de aproximandamente $3,2$.

Esta variante tem por objectivo, para além de acelerar o processo de desencriptação, proteger o sistema RSA contra ataques a sistemas criptográficos RSA que usam expoentes privados pequenos descrito na secção 3.1. Para tal escolhe-se um d grande mas usa-se uma chave privada $\langle p, q, d_p, d_q \rangle$, onde d_p e d_q são números pequenos, com k-bits e $k \leq n/2$ onde n é o tamanho da chave, nas condições acima descritas para desencriptar uma mensagem (ou assinar). No entanto, Boneh e Shacham em [BS02] chamam a atenção para o facto de que tendo uma chave privada $\langle p, q, d_p, d_q \rangle$ nessas condições, com $d_p < d_q$, então pode-se determinar d em tempo $O(\log d_p \sqrt{d_p})$. A escolha de $k \geq 160$ tem a ver com este facto, pois se d_p e d_q têm pelo menos 160-bits cada, $\log d_p \sqrt{d_p} = 160\sqrt{2^{160}} \geq 2^{80}$, o que torna o cálculo de d impraticável (actualmente).

Apêndice A

PKCS#1

Para encriptar ou assinar uma mensagem usando o sistema criptográfico RSA a mensagem tem primeiro de ser devidamente formatada. O texto original é de alguma forma transformado num número binário X, por exemplo usando o código ASCII. A sequência de bits (*bit string*) X é divida em partes de um dado tamanho. A cada uma dessas partes é aplicado o PKCS#1 (*Public Key Cryptography Standard*) que é um conjunto de normas para camuflar (*padding*) e formatar a mensagem antes de a encriptar ou assinar. Estas normas foram desenvolvidas pelo RSA Laboratories [RSA03] e alguns dos seus clientes (por exemplo, Apple, Microsoft, MIT) de sistemas de segurança de dados, no início dos anos 90. As regras de camuflagem são ligeiramente diferentes nos casos de encriptar e assinar, devido ao tipo de ataques que têm por objectivo proteger.

A formatação dos blocos tem a seguinte estrutura

00	BT	PS	00	D

em que

- o tamanho total do bloco é igual ao tamanho do módulo RSA que se está a usar escrito em "bytes", vamos supôr k-bytes;

- D são os dados a ser encriptados ou assinados, com no máximo $(k-11)$-bytes; devemos ter em atenção que o primeiro dígito de D tem que ser diferente de zero para que não haja ambiguidade quanto ao seu tamanho;

- PS (*padding string*) é uma sequência de bytes de tamanho $[k-3-$(tamanho de D em bytes)]-bytes e serve para camuflar a mensagem;

- BT (*block type*) é um byte que escrito em hexadecimal é 00 ou 01; se BT=00 então PS tem todos os bytes iguais a zero, caso que se usa quando se pretende encriptar a mensagem, e se BT=01 então PS tem todos os bytes iguais a ff[1], caso que se usa quando se pretende assinar;

- 00 é um "byte"escrito em hexadecimal que assegura que o bloco quando interpretado como um valor inteiro é menor que o módulo RSA.

O PKCS#1 é apenas um exemplo de formatação, que segundo Menezes [MvOV97, p. 656] incorpora duas versões anteriores de PKCS. Existe uma série de outras versões para o PKCS cada uma delas com as suas expecificações como se poderá ver em [MvOV97, p. 656] e na página do RSA Laboratories [RSA03, /pkcs].

[1]Na notação hexadecimal usam-se 16 símbolos, utilizando-se os dez algarismos da notação decimal seguidos das primeiras 5 letras do alfabeto: 0, 1, $\cdots$, 9, a, b, c, d, e, f; portanto na base 16, ff= $15 \times 16 + 15 = 265$, na base 10.

Apêndice B

$\mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$

Consideremos $\mathbb{Z}_m$ o conjunto das classes de equivalência ou classes residuais módulo m, onde m é um inteiro positivo. Um elemento $a \in \mathbb{Z}_m$ será representado por $[a]_m$. Note-se que a relação de congruência é uma relação de equivalência porque tem as seguintes propriedades:

- $a \equiv a \,(mod\, m)$, $\forall a \in \mathbb{Z}$;
- $a \equiv b \,(mod\, m) \Leftrightarrow b \equiv a \,(mod\, m)$, $\forall a, b \in \mathbb{Z}$;
- $a \equiv b \,(mod\, m) \wedge b \equiv c \,(mod\, m) \Rightarrow a \equiv c \,(mod\, m)$, $\forall a, b, c \in \mathbb{Z}$.

Além disso:

$$a \equiv a' \,(mod\, m) \wedge b \equiv b' \,(mod\, m) \Rightarrow \begin{cases} a \pm b \equiv a' \pm b' \,(mod\, m) \\ ab \equiv a'b' \,(mod\, m) \end{cases}, \ \forall a, a', b, b' \in \mathbb{Z}.$$

Das propriedades das congruências concluí-se que $\mathbb{Z}_m$ é um anel comutativo. Se p for primo, $\mathbb{Z}_p$ é corpo e o grupo multiplicativo $\mathbb{Z}_p^*$ dos elementos não nulos de $\mathbb{Z}_p$ é cíclico e existe $g \in \mathbb{Z}$ tal que g, g^2, ..., g^{p-1} são exactamente todos os elementos de $\mathbb{Z}_p$. A g chama-se gerador ou raiz primitiva de $\mathbb{Z}_p$ [LeV96, pp. 79-84].

Designemos por $\mathbb{Z}_m^*$ o grupo dos elementos invertíveis de $\mathbb{Z}_m$ e por $\varphi(m)$ a ordem de $\mathbb{Z}_m^*$. Observe-se que se $z \in \mathbb{Z}_m^* \Leftrightarrow zx = 1$ é solúvel em $\mathbb{Z}_m$, $zx \equiv 1 \ (mod\ m)$ tem solução que sabemos só acontecer se e só se $(z, m) = 1$.

É fácil verificar que, sendo $N = pq$,

$$\begin{array}{ccc} \mathbb{Z}_N^* & \rightarrow & \mathbb{Z}_p^* \times \mathbb{Z}_q^* \\ x & \mapsto & (x, x). \end{array} \tag{B.1}$$

é um isomorfismo. Pretende-se aqui exibir o seu inverso e para tal vamos usar o Teorema Chinês dos Restos.

Teorema B.0.1 (Teorema Chinês dos Restos). *Sejam $n_1, n_2, ..., n_k$ inteiros positivos primos entre si dois a dois e sejam $r_1, r_2, ..., r_k$ inteiros quaisquer. Então as congruências*

$$\begin{cases} x \equiv r_1 \ (mod\ n_1) \\ x \equiv r_2 \ (mod\ n_2) \\ \quad \vdots \\ x \equiv r_k \ (mod\ n_k) \end{cases}$$

têm uma solução comum que é única módulo $N = n_1 \cdot ... \cdot n_k$.

Prova: Seja $N = \prod_{i=1}^{k} n_i$. Tem-se evidentemente $(\frac{N}{n_i}, n_i) = 1, \forall i \in \{1, 2, ..., k\}$. Daqui resulta que existem inteiros t_i tais que $\frac{N}{n_i} t_i \equiv 1 \ (mod\ n_i)$. Como $\frac{N}{n_i} t_i \equiv 0 \ (mod\ n_j)$, com $j \neq i$, o inteiro

$$x_0 = \sum_{i=1}^{k} \frac{N}{n_i} t_i r_i$$

é solução de cada uma das congruências consideradas, visto que

$$\begin{aligned} x_0 &= \tfrac{N}{n_1} t_1 r_1 + \tfrac{N}{n_2} t_2 r_2 + \ ... \ + \tfrac{N}{n_1} t_k r_k \\ &\equiv \tfrac{N}{n_i} t_i r_i \ (mod\ n_i), \text{ pois } \tfrac{N}{n_j} \equiv 0 \ (mod\ n_i), \text{ se } j \neq i \\ &\equiv r_i \ (mod\ n_i), \ \forall i \in \{1, 2, ..., k\}, \text{ porque } \tfrac{N}{n_i} t_i \equiv 1 \ (mod\ n_i). \end{aligned}$$

Vejamos que a solução é única módulo N. Com efeito, se x_0 e x_1 são soluções de todas as congruências $x \equiv r_i \ (mod\ n_i), \ \forall i \in \{1, 2, ..., k\}$ tem-se então $x_0 \equiv x_1 \ (mod\ n_i), \forall i \in$

$\{1, 2, ..., k\}$ e como os n_i são primos entre si dois a dois, resulta que $x_0 \equiv x_1 \ (mod\ N)$.
□

A pré-imagem de $(y, z) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ pelo isomorfismo (B.1) é portanto dada por

$$[x]_N \equiv [y]_p t_1 q + [z]_q t_2 p \ (mod\ N),$$

onde $t_1 q \equiv 1\ mod\ p$ e $t_2 p \equiv 1\ mod\ q$. Vejamos um exemplo.

Exemplo B.0.2. Consideremos os seguintes grupos

$$\mathbb{Z}_3^* = \{1, 2\}$$
$$\mathbb{Z}_5^* = \{1, 2, 3, 4\}$$
$$\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$$

Vamos ver como é que, dado um elemento de $\mathbb{Z}_3^* \times \mathbb{Z}_5^*$, se obtém o elemento de $\mathbb{Z}_{15}^*$ que lhe corresponde via o isomorfismo (B.1). Sejam $r_1 \in \mathbb{Z}_3^*$ e $r_2 \in \mathbb{Z}_5^*$. O que queremos é determinar $z \in \mathbb{Z}_{15}^*$ tal que

$$\begin{cases} z \equiv r_1 \ (mod\ p) \\ z \equiv r_2 \ (mod\ q) \end{cases}.$$

Sejam $n_1 = 3$ e $n_2 = 5$ para que $N = 15 = n_1 n_2$. Calcula-se $\frac{N}{n_1} = \frac{15}{3} = 5$ e $5t_i \equiv 1\ (mod\ 3) \Rightarrow t_1 = 2$ e $\frac{N}{n_2} = \frac{15}{5} = 3$ e $3t_2 \equiv 1\ (mod\ 5) \Rightarrow t_2 = 2$. Assim,

$$z \equiv r_1 t_1 \frac{N}{n_1} + r_2 t_2 \frac{N}{n_2}\ (mod\ 15) \equiv 10 r_1 + 6 r_2\ (mod\ 15).$$

Portanto cada um dos 8 elementos $\mathbb{Z}_{15}^*$ é a pré-imagem de um elemento $(y, z) \in \mathbb{Z}_3^* \times \mathbb{Z}_5^*$, obtido da forma acima descrita. Na tabela seguinte explicita-se o isomorfismo inverso do isomorfismo (B.1) neste caso concreto.

$\mathbb{Z}_3^* \times \mathbb{Z}_5^*$	$\mathbb{Z}_{15}^*$
(1,1)	$10 \times 1 + 6 \times 1 = 10 + 6 \equiv 16 \ (mod\ 15) = 1$
(2,1)	$10 \times 2 + 6 \times 1 = 20 + 6 \equiv 26 \ (mod\ 15) = 11$
(1,2)	$10 \times 1 + 6 \times 2 = 10 + 12 \equiv 22 \ (mod\ 15) = 7$
(2,2)	$10 \times 2 + 6 \times 2 = 20 + 12 \equiv 32 \ (mod\ 15) = 2$
(1,3)	$10 \times 1 + 6 \times 3 = 10 + 18 \equiv 28 \ (mod\ 15) = 13$
(2,3)	$10 \times 2 + 6 \times 3 = 20 + 18 \equiv 38 \ (mod\ 15) = 8$
(1,4)	$10 \times 1 + 6 \times 4 = 10 + 24 \equiv 34 \ (mod\ 15) = 4$
(2,4)	$10 \times 2 + 6 \times 4 = 20 + 24 \equiv 44 \ (mod\ 15) = 14$

Como o cálculo da solução do sistema de congruências no Teorema Chinês dos Restos envolve apenas o algoritmo de Euclides e umas multiplicações e adições, a sua complexidade é a mesma do algoritmo de Euclides.

Referências

[And94] George E. Andrews. *Number Theory.* Dover, New York, 1994.

[Ang95] W. S. Anglin. *The Queen of Mathematics: an introdution to number theory.* Kluwer Academic Publishers, 1995.

[BD00] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $n^{0,292}$. *IEEE Transactions on Informattion Theory*, 46(4):1339–1349, Jul. 2000.

[BDHG99] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $n = p^r q$ for large r. In *Proceedings of Crypto'99*, volume 1666, pages 326–337. LNCS, Springer-Verlag, 1999.

[Bon99] Dan Boneh. Twenty years of attacks the RSA cryptosystem. *Notices of the AMS*, pages 203–213, February 1999.

[BS02] Dan Boneh and Hovav Shacham. I. Fast variants of RSA, II. How to encrypt properly with RSA, III. Composite-residuosity based cryptography: An overview. *RSA Laboratories CryptoBytes*, 5(1), Winter/Spring 2002.

[Coh93] H. Cohen. *A Course in Computational Algebraic Number Theory.*, volume 138 of *Graduate Texts in Mathematics.* Springer-Verlag, 1993.

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *Journal of CRYPTOLOGY*, (10):233–260, 1997.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Informattion Theory*, 22(6):644–654, Nov. 1976.

[DK02] Hans Delfs and Helmut Knebl. *Introdution to Cryptography.* Springer-Verlag, 2002.

[Fia89] Amos Fiat. Batch RSA. In G. Brassard, editor, *Proceedings of Crypto '89*, volume 435, pages 175–185. LNCS, Springer-Verlag, 1989.

[GKP95] Ronald Graham, Donald Knuth, and Oren Patashnik. *Concrete Mathematics.* Addison-Wesley, 2nd edition, 1995.

[HG98] Howgrave-Graham. *Computational Mathematics Inspired by RSA.* PhD thesis, University of Bath, 1998.

[HLT02] M. Hinek, Mo Low, and E. Teske. On some attacks on multi-prime RSA. In *Advances in Cryptology- Proceedings of SAC 2002.* LNCS, to appear, 2002.

[JS98] Antoine Joux and Jacques Stern. Lattice reduction: A toolbox for the cryptanalyst. *Journal of CRYPTOLOGY*, (11):161–185, 1998.

[Khi97] A. Ya Khinchin. *Continued Fractions.* Dover, 3rd edition, 1997.

[Knu73] Donald Knuth. *The Art of Computer Programming.* Addison-Wesley, 2nd edition, 1973.

[Kob87] Neal Koblitz. *A Course in Number Theory and Cryptography.*, volume 114 of *Graduate Texts in Mathematics.* Springer-Verlag, 1987.

[Kob98] Neal Koblitz. *Algebraic Aspects of Cryptography.* Graduate Texts in Mathematics. Springer-Verlag, 1998.

[Lab03] RSA Laboratories. *Frequently Asked Question About Today Cryptography.* http://www.rsasecurity.com/rsalabs/faq, 2003.

[LeV96] William J. LeVeque. *Fundamentals of Number Theory.* Dover, New York, 1996.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[Lov86] László Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity.* SIAM Publications, 1986.

[LV01] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptography key sizes. *Journal of CRYPTOLOGY*, (14):255–293, August 2001.

[Mol01] Richard A. Mollin. *An Introdution to Cryptography.* Chapman&Hall/CRC, 2001.

[Mor93] José Morgado. *Teoria Elementar dos Números.* Departamento de Matemática Pura da FCUP, 1993.

[MvOV97] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[Rie85] Hans Riesel. *Prime Numbers and Computer Methods for Factorization.* Birkhäuser, 2nd edition, 1985.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978.

[RSA03] *RSA Laboratories.* http://www.rsasecurity.com/rsalabs/, 2003.

[Sal96] Arto Salomaa. *Public-Key Cryptography.* Springer-Verlag, 2nd edition, 1996.

[SB01] H. Shacham and D. Boneh. Improving SSL handshake performance via batching. In D. Naccache, editor, *Proceedings of RSA 2001*, volume 2020, pages 28–43. LNCS, Springer-Verlag, 2001.

[Sha95] Adi Shamir. RSA for paranoids. *RSA Laboratories CryptoBytes*, 1(3), Autumn 1995.

[Sil00] John R. Silvester. Determinants of block matrices. *Maths Gazette*, 84:460–467, 2000.

[Spi94] Karlheinz Spindler. *Abstract Algebra with Applications*, volume 2. Marcel Dekker, 1994.

[Tak97] T. Takagi. Fast RSA-type cryptosystem using n-adic expansion. In *Advances in Cryptology- Crypto'97*, volume 1294, pages 372–384. LNCS, Springer-Verlag, 1997.

[Tak98] T. Takagi. Fast RSA-type cryptosystem modulo $p^k q$. In *Proceedings of Crypto'98*, volume 1462, pages 318–326. LNCS, Springer-Verlag, 1998.

[Vin77] Ivan Vinogradov. *Fundamentos de la Teoria de los Numeros.* Editorial Mir, 1977.

[Wad81] Wadsworth, editor. *The Mathematical Gardner: Mental Poker*, pages 37–43. David Klarner, 1981.

[Wie90] M. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Informattion Theory*, 36(3):553–558, May 1990.