Criptoanálise

Pedro Quaresma
Departamento de Matemática, Universidade de Coimbra
3001-454 COIMBRA, PORTUGAL
pedro@mat.uc.pt

Augusto Pinho
Núcleo de Estágio Pedagógico
Lic. Matemática, F.C.T.U.C.
Escola B. 2, 3 c/ Sec. José Falcão, Miranda do Corvo
gustopinho@iol.pt

24 de Julho de 2007

1 Introdução

A capacidade de comunicar está na base da definição de qualquer sociedade humana, a necessidade de esconder a informação que um dado grupo troca entre si dos outros diferentes grupos vem logo a seguir, concomitante com esta vem a necessidade de descobrir os segredos que os outros pretendem guardar. Os sistemas criptográficos surgiram então como forma de garantir a confidencialidade da informação, a criptoanálise surgiu como forma de "quebrar" essa mesma confidencialidade.

Um sistema criptográfico é um conjunto de técnicas que permitem tornar incompreensível uma dada mensagem, de modo que só o verdadeiro destinatário da mesma a consiga decifrar, obtendo dessa forma o texto original [4, 6, 7].

A criptoanálise pelo seu lado desenvolve as técnicas capazes de "quebrar" as diferentes cifras de forma a conseguir recuperar o texto original, mesmo que parcialmente, a partir do texto cifrado.

É importante notar que a criptoanálise não é o mesmo que a decifração de uma mensagem cifrada, neste último caso está-se perante um processo "normal" de, dado um texto cifrado, obter o texto original, isto é, o decifrador pertence ao mesmo grupo que o cifrador e tem a informação necessária para, por um processo simétrico ao da cifração, obter a mensagem original a partir da mensagem cifrada. No caso da criptoanálise está-se perante um elemento de um grupo contrário que, ou não tem nenhum conhecimento sobre o processo de cifração, ou tem um conhecimento parcial do mesmo.

No presente artigo vamos abordar algumas das técnicas disponíveis ao criptoanalista, nomeadamente o estudo da frequência relativa das letras ou grupos de letras numa dada língua natural, e os algoritmos de factorização de números primos. A cada método de criptografia está associado um ou mais métodos de criptoanálise. No caso presente trata-se dos sistemas de substituição mono-alfabéticos e o método RSA [1, 4, 6, 7].

2 Criptoanálise

Como já foi dito o criptoanalista tenta obter o texto original a partir do texto cifrado, não tendo para tal o conhecimento do processo exacto usado para a cifração, em particular não tendo o conhecimento da chave secreta necessária para a decifração.

Quais são então os objecto e técnicas do criptoanalista? Para já o(s) texto(s) cifrado(s), eventualmente o conhecimento do tipo de cifração usada, e também o conhecimento, mesmo que parcelar, do(s) texto(s) original(is). As estórias de espionagem estão cheias de casos em que uma dada informação é "entregue" ao inimigo só para provocar uma mensagem cifrada cujo texto original se poderia adivinhar, ou então de golpes de sorte que resultam na obtenção de informações importantes sobre o processo de cifração usada pelo inimigo.

Estes são alguns dos objectos que o criptoanalista tem ao seu dispor, vejamos agora algumas das técnicas de que ele se pode socorrer. No que se segue vamos assumir que temos o conhecimento do tipo de cifração usado e de um, ou mais, textos cifrados.

2.1 Sistemas de Substituição Mono-alfabéticos

Os sistemas de substituição mono-alfabéticos são caracterizados por serem sistemas em que cada letra do texto original é substituída por uma outra letra no texto cifrado, sendo que ambos os textos usam o mesmo alfabeto de base. Um exemplo de um sistema desse tipo é o assim designado, *Cifra de Júlio César*. Neste sistema cada letra é substituída por uma outra três posições mais à direita no alfabeto romano. Este é um sistema de substituição mono-alfabético aditivo [6, 7] muito simples de quebrar. A designação aditivo advêm do modo como é feita a substituição, neste caso preciso, por adição (deslocação para a direita) de três posições à posição original da letra no alfabeto. Considerando que o alfabeto tem 26 caracteres distintos então existem somente 25 cifras distintas, como tal um "ataque directo" por tentativa e erro é possível, permitindo "quebrar" a cifra rapidamente.

Nem todos os sistemas de substituição mono-alfabéticos são assim tão fáceis de quebrar, no entanto a sua própria essência vai permitir um outro tipo de "ataque", sendo este aplicável a todos os sistemas deste tipo que possamos inventar.

Como foi dito atrás cada letra do alfabeto é substituída por uma outra, quer isto dizer que, num dado texto, todos os 'a's são substituídos por, por exemplo, 't's, isto é pode-se dizer que todos os 'a's continuam no texto cifrado, eles estão somente mascarados de 't's. Será que isso é importante? Será que desse facto podemos extrair alguma informação que nos permita decifrar o texto cifrado?

A resposta a estas duas questões é afirmativa em ambos os casos. Para uma dada língua natural pode-se verificar que cada letra tem uma frequência própria, por exemplo para o Português temos (não se tem em conta os caracteres acentuados) [6, 7]:

A	12,71%	B	0,81%	C	4,16%	D	5,52%	E	11,99%
F	1,34%	G	1,32%	H	0,74%	I	7,18%	J	0,21%
K	0,00%	L	3,23%	M	4,48%	N	5,24%	O	11,32%
P	3,07%	Q	1,41%	R	6,47%	S	7,99%	T	5,31%
U	3,44%	V	1,36%	W	0,02%	X	0,28%	Y	0,02%
Z	0,37%								

É possível, dados estes valores, agrupar as letras em grupos bem definidos, das mais frequentes para as menos frequentes:

1º	A, E, O
2º	S, R, I
3º	N, D, M, U, T, C
4º	L, P, V, G, H, Q, B, F
5º	Z, J, X, K, W, Y

É ainda possível obter as frequências relativas de alguns *digramas* (pares de letras) e de *trigramas* (ternos), assim como as letras de início de palavras, as de fim de palavra, e as palavras mais frequentes.

Toda esta informação permite "quebrar" uma cifra deste tipo tendo somente o conhecimento do texto cifrado, obviamente que quanto maior o texto mais fácil se torna a análise. Por outro lado este estudo não elimina por completo um processo de tentativa e erro, o que faz é reduzir o número de tentativas a efectuar antes de se ser bem sucedido.

Tentemos então "quebrar" a cifra usada para criar o texto `D FKDYH WHP GH VHU PDQWLGD VHFUHWD`, sabendo de antemão que foi usado um sistema de cifração mono-alfabético, aditivo.

Fazendo o cálculo das frequências relativas temos:

D	17,9%	F	7,1%	G	7,1%	H	21,4%
K	3,6%	L	3,6%	P	7,1%	Q	3,6%
U	7,1%	V	7,1%	W	10,7%	Y	3,6%

Temos então que o "D", com 17,9%, e o "H", com 21,4% fazem com certeza parte do grupo de letras de alta frequência ("A", "E", e "O"). Dado que estamos a analisar uma frase pequena o estudo de frequência só nos fornece uma aproximação, um ponto de partida, para começarmos o processo de tentativa e erro.

Voltando ao nosso exemplo temos: "D" e "H" como candidatos para "A", tentemos primeiro a chave 3 ("A" $\rightsquigarrow$ "D"):

A	B	C	D	E	F	G	H	I	J	K	L	M
D	E	F	G	H	I	J	K	L	M	N	O	P
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Q	R	S	T	U	V	W	X	Y	Z	A	B	C

obtém-se então:

```
A CHAVE TEM DE SER MANTIDA SECRETA
```

Ou seja, conseguimos quebrar a cifra!

Com o advento dos computadores em que todo este tratamento pode ser executado a velocidades estonteantes, facilmente se conclui que este tipo de método deixou de ser seguro.

Saltando um pouco na evolução dos sistemas criptográficos, vamos agora analisar os sistemas que baseiam o processo de cifração em problemas computacionais difíceis, isto é

problemas que só são resolvidos computacionalmente à custa de enormes custos em tempo gasto e/ou espaço, ou dito doutra forma, problemas cuja resolução é computacionalmente muito difícil seja em termos de tempo computacional despendido, seja em termos do volume de memória computacional necessária para a sua resolução, ou ambos.

2.2 Problemas Computacionais Difíceis

A ideia por de trás de alguns dos sistemas de criptografia da actualidade é a de explorar problemas que se crê serem computacionalmente difíceis. Um problema é dito computacionalmente difícil quando, por um lado a sua solução computacional exigir muitos recursos computacionais, seja no tempo necessário para o resolver, seja na quantidade de memória (RAM) exigida para a sua resolução (ou ambos). Por outro lado num problema deste tipo, a um pequeno aumento na dimensão do problema, vai corresponder um grande aumento nos recursos computacionais necessários para a sua resolução. Temos então que se se basear um sistema criptográfico num problema deste tipo, de tal forma que a quebra da cifra implique a resolução do problema, então teremos que: a quebra da cifra será sempre computacionalmente muito difícil; a um pequeno aumento da dimensão do problema (por exemplo a escolha de uma chave de cifra melhor), vai corresponder a um grande aumento da dificuldade da quebra da cifra.

Entre outros problemas deste tipo temos [6]:

- RSA - dado um $n \in \mathbb{N}, n = pq$, com p e q primos, factorizar n nos seus factores primos.
- El Gamal - problema do logaritmo discreto. Dado um grupo cíclico finito G, dado um gerador $g \in G$ e um elemento $x \in G$, determinar o único $i \in \mathbb{Z}_G$ tal que $x = g^i$;
- Knapsack - dado uma mochila com um dado volume, e dados n objectos com volumes diferentes, descobrir um sub-conjunto de elementos que encha de forma exacta a mochila.

Estes métodos têm em comum o facto de ser possível construir um sistema criptográfico em que a cifração é feita de forma eficiente, mas em que a "quebra" da cifra se crê ser computacionalmente muito difícil.

De seguida vamos apresentar algumas das técnicas de criptoanálise disponíveis para o sistema RSA. Em [6] o leitor poderá encontrar todos os detalhes referentes a este, a aos restantes sistemas acima referidos.

2.3 O sistema RSA

O sistema RSA é um sistema assimétrico, isto é, possui uma chave pública (e, n), e uma chave privada (d, n), em que [4, 6]:

- $n = pq$, com p e q, números primos;
- $1 < e < \varphi(n)$, em que $\varphi(n)$, a função de Euler, é igual ao número de co-primos de n;
- $d\,e = 1 (\mathrm{mod}\ \varphi(n))$.

Sabendo a chave pública, isto é, (e, n), basta factorizar n no produto de números primos p, e q, para depois facilmente se obter d, ou seja a chave secreta (d, n). O problema é que a factorização de um número nos seus factores primos é um problema computacionalmente difícil [3, 5, 6].

Vejamos de seguida alguns dos métodos de criptoanálise para o sistema RSA. Iremos acabar esta secção com um estudo comparativo entre eles.

2.3.1 Método das Divisões

Este é o *método da força bruta*, ou seja, vai-se tentar a divisão sucessiva por todos os números primos até $\lfloor\sqrt{n}\rfloor$, ou até que a solução seja encontrada.

Por exemplo em *Octave*[1] ter-se-ia[2]:

```
## Método da "força bruta"
##
## -> n, o inteiro a factorizar
## <- resp, o primeiro factor primo
function resp=Divisoes(n)
  ## cálculo do limite
  limite=floor(sqrt(n));
  ## vector com todo os primos até ao limite
  c=crivoEratostenes(limite);
  ## número de primos obtidos
  nprimos=columns(c);
  i=0;
  ## ciclo de teste, desde o primeiro até ao penúltimo primo
  do
    i=i+1;
    if(mod(n,c(i))==0) # (mod - resto da divisão inteira)
      resp=c(i);
    endif
  until (i>=nprimos || mod(n,c(i))==0)
  ## teste para o último primo
  if (i>=nprimos && mod(n,c(nprimos))==0)
    resp=c(nprimos);
  endif
endfunction
```

Uma dos pontos fracos desta aproximação é a necessidade que se tem de gerar os números primos até um determinado m. Infelizmente, a despeito do esforço das melhores mentes matemáticas ao longo dos séculos, não é actualmente conhecida nenhuma fórmula para a geração de números primos. É então necessário recorrer a um método que construa a

[1] Octave, `www.octave.org` é um sistema de programação numérica de distribuição gratuita, compatível com o *MatLab*.

[2] Os algoritmos estão disponíveis em: `http://www.mat.uc.pt/~pedro/cientificos/Cripto`

lista de todos os números primos até um dado limite, por exemplo o *Crivo de Eratóstenes* [3]. Tentando descrever sumariamente o algoritmo[2] tem-se: começa-se por gerar um vector com todos os inteiros de 2 a n; 2 é primo; aplica-se de seguida o "crivo 2" à lista retirando desse modo o 2 e todos os seus múltiplos; 3 o primeiro dos inteiro que sobraram é primo, então podemos agora aplicar o "crivo 3" à lista. É fácil de ver que se repetirmos o processo até ao fim da lista original, os elementos que não forem eliminados pelas sucessivas aplicações dos crivos, constituem a lista dos números primos de 2 a n.

A necessidade de criar a lista de todos os inteiros de 2 a n, e as inúmeras vezes ($n-1$ para ser preciso) que é necessário percorrer essa lista para aplicar os sucessivos crivos leva a que, a utilização do *Crivo de Eratóstenes* acrescenta um peso muito significativo ao algoritmo tanto temporalmente como espacialmente.

Uma alternativa é a utilização de uma fórmula que gere todos os números primos sucessivos, além de outros não primos também. A utilização de uma tal fórmula torna o ciclo de tentativas de divisão menos eficiente, no entanto os ganhos, tanto espacialmente como temporalmente, obtidos pela não utilização do *Crivo de Eratóstenes* compensam essas perdas. Podemos, por exemplo, utilizar a fórmula $p = 6k \pm 1$ [5].

Em *Octave* ter-se-ia:

```
## Utilização da fórmula 6k+1 e 6k-1 para determinar números primos.
##
## -> n
## <- resp, o primeiro factor primo
function resp=Divisoes1(n)
  limite=floor(sqrt(n));
  ## testa para 2 e para 3 (mod - resto da divisão inteira)
  if (mod(n,2)==0) resp=2; endif
  if (mod(n,3)==0) resp=3; endif
  ## de 6*1-1=5, 6*1-1+2=6*1+1 até ao limite
  p=5;
  while (p<=limite && resp==0)
    ## testa 6k-1 e 6k-1+2=6k+1
    if (mod(n,p)==0) resp=p; endif
    if (mod(n,p+2)==0) resp=p+2; endif
    p=p+6;
  endwhile
endfunction
```

Depois de apresentarmos mais alguns métodos alternativos iremos comparar a eficiência temporal de todos eles, tentando dessa forma responder à questão de se saber se é viável, com estes programas, quebrar a cifra RSA.

2.3.2 Método de Euclides

Este método ganha o seu nome da utilização do algoritmo de Euclides[2] para o cálculo do máximo divisor comum de dois inteiros [2]. Este algoritmo é muito eficiente e pode-nos ajudar a obter um dos factores primos de n. Basta multiplicar todos os números primos

entre 2 e $\lfloor\sqrt{n}\rfloor$, calculando de seguida o $m.d.c$ entre esse produto e n, de forma a obter o factor primo desejado.

Esta aproximação apresenta dois problemas: primeiro a necessidade de gerar a lista de todos os primos até um dado limite, como já dissemos essa tarefa é pesada tanto temporalmente como espacialmente; o outro problema advém da representação computacional do número que se obtém da multiplicação dos números primos, rapidamente o número obtido da multiplicação ultrapassa a capacidade de representação da maioria das linguagens de programação existentes.

Para obstar a este último problema pode-se dividir a multiplicação em várias multiplicações parcelares.

Para tal procede-se do seguinte modo:

- Começa-se por definir os conjuntos auxiliares:

  $R = \{r_1, \ldots, r_m\}$ representado r_i um limite inferior $(r_1 = 1, r_i < r_{i+1})$
  $S = \{s_1, \ldots, s_m\}$ representado s_i um limite superior $(s_i < s_{i+1}, s_{m-1} < \lfloor\sqrt{n}\rfloor \leq s_m)$

- Para cada par r_i e s_i, multiplicam-se todos os números primos entre estes dois limites, $P_i = \prod_{r_i \leq p_j \leq s_i} p_j$;

- Para cada um dos P_i calcula-se o $\text{mdc}(P_i, n) = a_i$.

- Se $a_i \neq 1$, então a_i é o factor primo de n que se pretende obter.

Vejamos um exemplo: seja $n = 1457$, $\lfloor\sqrt{n}\rfloor = 38$, e façamos as somas parcelares de 10 em 10.

$$R = \{1, 11, 21, 31\} \qquad S = \{10, 20, 30, 40\}$$

$$\begin{aligned}
P_1 &= \prod_{1 \leq p_j \leq 10} p_j = 2 \times 3 \times 5 \times 7 = 210 && \text{mdc}(210, 1457) = 1 \\
P_2 &= \prod_{11 \leq p_j \leq 20} p_j = 11 \times 13 \times 17 \times 19 = 46189 && \text{mdc}(46189, 1457) = 1 \\
P_3 &= \prod_{21 \leq p_j \leq 30} p_j = 23 \times 29 = 667 && \text{mdc}(667, 1457) = 1 \\
P_4 &= \prod_{31 \leq p_j \leq 40} p_j = 31 \times 37 = 1147 && \text{mdc}(1147, 1457) = 31
\end{aligned}$$

Temos então 31 como um dos factores primos de 1457.

Em *Octave*:

```
## Método de Euclides
##
## -> n
## <- resp, o primeiro factor primo
function resp=Euclides(n)
  ## cálculo do limite
  limite=floor(sqrt(n));
```

```
  ## vector com todo os primos até ao limite
  c=crivoEratostenes(limite);
  ## número de primos obtidos
  nprimos=columns(c);
  ## inicialização das variáveis que controlam a construção dos
  ## conjuntos auxiliares (de 10 em 10)
  passo=10;k=1;i=1;si=passo;
  ## inicializa o vector dos produtos só com 1s (elemento neutro
  ## da multiplicação)
  prods=ones(1,limite);
  ## cálculo dos productórios (guardados no vector "prods")
  while (i<=nprimos)
    if (c(i)>si)
      si=si+passo;
      k=k+1;
    endif
    prods(k)=prods(k)*c(i);
    i=i+1;
  endwhile
  m=1;i=1;
  ## cálculo do mdc entre os productórios e n
  while (i<=k && m==1)
    m=mdc(prods(i),n);
    if (m!=1)
      resp=m;
    endif
    i=i+1;
  endwhile
endfunction
```

Novamente põe-se a questão de gerar os números primos até um determinado limite. Além desta limitação os produtos de números primos vão rapidamente levantar problemas de representação numérica, isto é, mesmo que a representação dos números primos por si só não levante problemas a sua multiplicação, mesmo que em grupos reduzidos, vai rapidamente levantar problemas de representação na gama finita disponível nos tipos de dados das linguagens de programação mais usuais.

2.3.3 Método de Fermat

O método de Fermat consiste em encontrar dois inteiros a e b que permitam representar o número natural n como a diferença de dois quadrados:

$$n = a^2 - b^2 \quad \Leftrightarrow \quad n = (a-b)(a+b)$$

Teorema 1 *Qualquer inteiro n ímpar maior do que* 1 *pode ser escrito como a diferença de dois quadrados*

Demonstração:

Seja $n = pq$, com $p \geq q$ (no caso de n ser primo considera-se $n = n \times 1$)

Por hipótese n é ímpar, então p e q também o são, logo:

$$\tfrac{p+q}{2} \text{ e } \tfrac{p-q}{2} \text{ são inteiros}$$

mas então temos:

$$\begin{aligned}
(\frac{p+q}{2})^2 - (\frac{p-q}{2})^2 &= \frac{p^2+2pq+q^2}{4} - \frac{p^2-2pq+q^2}{4} \\
&= \frac{p^2+2pq+q^2-p^2+2pq-q^2}{4} \\
&= \frac{4pq}{4} \\
&= pq \\
&= n
\end{aligned}$$

ou seja

$$n = (\tfrac{p+q}{2})^2 - (\tfrac{p-q}{2})^2$$

□

Para encontrar a e b tais que $n = a^2 - b^2$ procede-se do seguinte modo:

- Dado um inteiro n ímpar começamos por tomar

$$a = \lfloor \sqrt{n} \rfloor + 1.$$

- Se $b = \sqrt{a^2 - n}$ é um inteiro, obtém-se o pretendido;
- Caso contrário, incrementamos a de uma unidade até que b seja um inteiro.

Por exemplo: seja $n = 2027651281$; $a = \lfloor \sqrt{n} \rfloor + 1 = 45030$ é o primeiro valor a testar.

	a	b		a	b
1º	45030	$\sqrt{45030^2 - 2027651281} = 222{,}75$	**7º**	45036	$\sqrt{45036^2 - 2027651281} = 768{,}12$
2º	45031	$\sqrt{45031^2 - 2027651281} = 373{,}73$	**8º**	45037	$\sqrt{45037^2 - 2027651281} = 824{,}67$
3º	45032	$\sqrt{45032^2 - 2027651281} = 479{,}31$	**9º**	45038	$\sqrt{45038^2 - 2027651281} = 877{,}58$
4º	45033	$\sqrt{45033^2 - 2027651281} = 565{,}51$	**10º**	45049	$\sqrt{45039^2 - 2027651281} = 927{,}49$
5º	45034	$\sqrt{45034^2 - 2027651281} = 640{,}21$	**11º**	45040	$\sqrt{45040^2 - 2027651281} = 974{,}84$
6º	45035	$\sqrt{45035^2 - 2027651281} = 707{,}06$	**12º**	45041	$\sqrt{45041^2 - 2027651281} = 1020$

Concluindo:

$$n = 45041^2 - 1020^2, \quad n = 46061 \times 44021$$

No caso do algoritmo de Fermat prova-se que, quanto maior for a diferença entre p e q, maior é o número de tentativas necessárias para obter um primeiro valor inteiro para a raíz [5]. Ou seja o método de Fermat leva-nos a escolher, para a nossa cifra, p e q primos distantes entre si.

Em *Octave*:

```
## Método de Fermat
##
## -> n
## <- resp=[a,b]
function resp=Fermat(n)
  ## começa-se por tomar a = (maior inteiro contido em raíz
  ## quadrada de n) + 1
  a=floor(sqrt(n))+1;
  ## se a é um quadrado então a resposta é [a,a]
  if (issqr(m))
    resp=[a,a];
  else # senão
    do
      b=a^2-n;
      ## se b=raíz quadrada(a^2-n) é um inteiro então a resposta
      ## é [a,b]
      if (issqr(b))
        resp=[a,sqrt(b)];
      endif
      ## senão incrementa-se a de uma unidade e recomeça-se
      a=a+1;
    until (issqr(b)) # até que b seja um inteiro
  endif
endfunction
```

2.3.4 Estudo Comparativo dos Vários Métodos

Será que já temos as ferramentas necessárias para quebrar a cifra RSA? Teoricamente a resposta é positiva, existem métodos construtivos de factorização de números primos os quais podem ser usados para obter o que se pretende. A questão é então de índole prática, isto é, será que já temos as ferramentas computacionais necessárias para quebrar, em tempo útil, a cifra RSA?

Estando o estudo teórico da complexidade dos algoritmos apresentados fora do âmbito do presente texto, vamos somente apresentar os resultados de um estudo comparativo dos vários métodos baseado num conjunto de testes de execução.

Todos os valores respeitam a testes efectuados sob as mesmas condições computacionais: sistema GNU/Linux 2.6.8; Intel Pentium 4 a 3,0GHz; 2GiB RAM; Octave 2.1.69. Os tempos referem-se ao tempo gasto pelo processador tal como nos é dado pelo sistema operativo.

n	Factores		Divisões	Divisões1	Euclides	Fermat
1457	47	31	0,009s	0,002s	0,015s	0,001s
13199	197	67	0,023s	0,005s	0,030s	0,002s
281161	3559	79	0,077s	0,006s	0,092s	0,218s
701123	3559	197	0,129s	0,016s	0,169s	0,179s
23420707	41017	571	0,699s	0,047s	0,839s	2,728s
488754769	110503	4423	3,343s	0,361s	4,477s	6,093s
2027651281	46061	41017	8,705s	3,611s	19,575s	0,004s
103955963689	47188363	2203	49,050s	0,179s	51,891s	3926,6s
128228613281	58206361	2203	55,016s	0,180s	58,180s	17175,0s
210528952589	95564663	2203	72,176s	0,182s	75,888s	7953,8s
2746662891777043	47188363	58206361	—	3861,6s	—	50,333s
4509540007616669	47188363	95564663	—	3857,9s	—	712,73s

Olhando para o quadro apresentado podemos concluir que, dado uma escolha apropriada dos factores primos, a cifra RSA está segura. Vejamos mais em pormenor:

- os métodos *Divisões* e *Euclides* não só vêem os seus tempos de execução subir de forma significativa com o crescimento da grandeza relativa do factores primos, como deixam de ser capazes de resolver o problema a partir de valores relativamente baixos de $n = p \times q$. A necessidade de gerar os números primos até $\lfloor \sqrt{n} \rfloor$ e, no caso do método de *Euclides*, a multiplicação de números de grande dimensão, estão na origem destas limitações.

- o tempo gasto pelo método *Divisões1* está directamente relacionado com o valor do primeiro factor primo, para valores elevados deste é um método pouco eficiente.

- o método de *Fermat* vê os seus tempos de execução crescer de modo muito significativo com o crescimento da dimensão do factores primos. É importante notar no entanto que há excepções a esse comportamento genérico, olhando com atenção verifica-se que, quando os factores primos estão próximos entre si, o método de *Fermat* é muito eficiente.

Concluímos então que para assegurar a segurança da cifra RSA os factores primos têm de ser distantes um do outro e o n tem de ter uma dimensão acima dos 20 dígitos.

Na verdade a dimensão de n tem de ser bastante mais elevada. Devido a algoritmos mais eficientes do que os aqui apresentados a cifra RSA utilizando valores de n com 129, 155, e mesmo 576 dígitos foi já quebrada. A cifra RSA actualmente usa valores de n com 1024, ou mais dígitos [4, 6].

3 A Quebra da Cifra RSA

Num anterior artigo da Gazeta de Matemática colocamos ao leitor um pequeno desafio [4]. Será que já estamos em condições de o resolver?

O texto cifrado é:

```
359394 185904 0 231105 382481 474195 382481 10935 75745 382481 185904 0 201637
382481 302441 522545 270765 382481 185904 0 185904 382481 265174 79985 0 365807
292080 66056 261188 75745 382481 371293 60839 185904 185904 265174 185904 0
90175 75745 75745 382481 185904 270765 522545 10935 66056 474195
```

sabe-se que foi usado o método RSA, com uma cifração letra a letra (caracteres *ASCII* entre ' ' e '~' que correspondem a, ' '=0, '!'=1,...), e que a chave pública usada foi $(5, 561971)$.

Qual será então o texto original?

Temos agora as ferramentas necessárias para fazer a criptoanálise deste texto. Na verdade a dimensão de $n = p \times q = 561971$ indica-nos que, qualquer um dos métodos aqui apresentados será capaz de obter os factores primos. Assim é:

```
octave:1> Fermat(561971)
tempoCPU = 0.001s
ans = 750   23

octave:2> Divisoes(561971)
tempoCPU = 0.060s
ans = 727

octave:3> Euclides(561971)
tempoCPU = 0.217s
ans = 727
```

Os dois factores primos são então 727 e 773. Dado a sua proximidade o algoritmo de Fermat foi o mais eficaz, não necessitando mais do que 0,001s para obter a factorização.

Utilizando os algoritmos do método RSA temos como chave pública $(5, 561971)$ e como chave privada $(224189, 561971)$. Posto isto podemos obter o texto original por aplicação directa do método RSA.

As Palavras Magicas sao: Quebra-Ossos Irrascivel

Esta frase é a tradução para o Português (descontando a falta de acentos), da frase "The magic words are squeamish ossifrage", a qual constituía o desafio original dos autores do sistema RSA. A cifra original, de chave pública $(9007, n)$ sendo que n é um dado número inteiro com 129 dígitos, foi quebrada utilizando o método do *Crivo Quadrático*, método esse que foi inventado já tendo como objectivo a criptoanálise do sistema RSA [1, 6].

4 Conclusões

A história da criptografia e da criptoanálise é então uma história de passos sucessivos, uma autêntica guerra travada no interior de gabinetes. Desde os métodos simples (mecânicos) da época pré-computadores, aos métodos mais complexos que exploram a capacidade dos computadores para "baralhar" a informação, até aos métodos que exploram os limites teóricos dos computadores, com a utilização dos problemas computacionais difíceis, estamos perante um processo constante em que para um dado avanço na criptografia se segue um avanço

na criptoanálise e vice-versa. Talvez que um próximo passo seja dado nas fronteiras da matéria, explorando a criptografia quântica, com a certeza de que a criptoanálise quântica se seguirá de imediato a esse passo, num processo infindável, até ao momento que não haja necessidade de comunicar de forma secreta [6].

Referências

[1] D. Atkins, M. Graff, A. Lenstra, and P. Leyland. The magic words are squeamish ossifrage. In *ASIACRYPT 1994*, pages 263–277, 1994.

[2] Donald E. Knuth. *The Art of Computer Programming*, volume 1, Fundamental Algorithms. Addisson-Wesley Publishing Company, Reading, USA, 2nd edition, 1973.

[3] Donald E. Knuth. *The Art of Computer Programming*, volume 2, Seminumerical algorithms. Addisson-Wesley Publishing Company, Reading, USA, 2nd edition, 1981.

[4] Pedro Quaresma and Elsa Lopes. Criptografia. *Gazeta de Matemática*, aceita para publicação.

[5] Hans Riesel. *Prime Numbers and Computer Methods for Factorization*, volume 126 of *Progress in Mathematics*. Birkhäuser, 2nd edition, 1994.

[6] Richard Spillman. *Classical and Contemporary Cryptology*. Prentice Hall, 2005.

[7] Viktoria Tkotz. *CRIPTOGRAFIA - Segredos Embalados para Viagem*. NOVATEC Editora, São Paulo, Brasil, 2005.