# Propriedades Multiplicativas

Seja $m_1$ e $m_2$ duas mensagens em texto claro, e sejam $c_1$ e $c_2$ as respectivas mensagens encriptadas. Veja-se que:

$$(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \pmod{n}$$

Por outras palavras, o texto cifrado correspondente ao texto claro $m = m_1 m_2 \mod n$, é $c = c_1 c_2 \mod n$; isto é muitas vezes referido como a *propriedades homomórfica da cifra RSA*. Esta observação leva ao seguinte ataque adaptativo texto cifrado escolhido.





# Propriedades Multiplicativas

Suponha-se que um adversário activo pretende decifrar um texto cifrado $c = m^e \mod n$ destinado para $A$.
Suponha-se também que esse adversário tem a capacidade de «forçar» $A$ a decifrar um dado texto cifrado arbitrário (que não o próprio $c$).

O adversário pode disfarçar $c$ seleccionando um inteiro aleatório $x \in \mathbb{Z}_n^*$ e calculando $\overline{c} = cx^e \mod n$.

Após a apresentação de $\overline{c}$, $A$ vai calcular para o adversário $\overline{m} = (\overline{c})^d \mod n$. Dado que

$$\overline{m} \equiv (\overline{c})^d \equiv c^d (x^e)^d \equiv mx \pmod{n}$$

o adversário pode então calcular $m = \overline{m} x^{-1} \mod n$.





# Propriedades Multiplicativas

Este ataque adaptativo de texto cifrado escolhido deve ser evitado na prática através da imposição de algumas restrições estruturais nas mensagens.

Se um texto cifrado $c$ é decifrado dando origem a um texto claro que não tenha essa estrutura então $c$ deve ser rejeitado por ser fraudulento.

Se a mensagem $m$ tem esta estrutura (escolhida de forma cuidadosa), então, com uma probabilidade alta $mx \mod n$ não terá essa mesma estrutura para um qualquer $x \in \mathbb{Z}_n^*$.

Consequentemente o ataque adaptativo de texto cifrado escolhido descrito anteriormente falha dado que $A$ não irá decifrar $\overline{c}$ para o adversário.





# Mensagens não Encriptáveis

## Definição (Mensagem não Encriptável)

*Uma mensagem $m$, $0 \leq m \leq n-1$, na cifra RSA é dita* não encriptável *se o resultado da encriptação é a própria mensagem; isto é*

$$m^e \equiv m \pmod{n}$$

.

Existem sempre mensagens que são não encriptáveis, por exemplo $m = 0$, $m = 1$, e $m = n - 1$. O número de mensagens não encriptável é de:

$$[1 + \text{mdc}(e-1, p-1)] \cdot [1 + \text{mdc}(e-1, q-1)]$$

Dado que $e-1$, $p-1$ e $q-1$ são todos pares, o número de mensagens não encriptável é sempre maior ou igual a 9.

Se $p$ e $q$ são primos escolhidos de forma aleatória, e se $e$ é também aleatório (ou se $e$ é escolhido de forma a ser pequeno), então a proporção de mensagem não encriptáveis será, de uma forma geral, negligenciável, e como tal as mensagens não encriptáveis não constituem um problema de segurança para a cifra RSA.

# O Desafio RSA (1991-2007)

| Número RSA | Dig. Dec. | Dig. Bin. | Prémio (USD) | Em | Por |
|---|---|---|---|---|---|
| RSA-100 | 100 | 330 | $1.000 | 4/1991 | Arjen K. Lenstra |
| RSA-110 | 110 | 364 | $4.429 | 4/1992 | Arjen K. Lenstra and M.S. Manasse |
| RSA-120 | 120 | 397 | $5.898 | 6/1993 | T. Denny et al. |
| RSA-129 | 129 | 426 | $100 | 4/1994 | Arjen K. Lenstra et al. |
| RSA-130 | 130 | 430 | $14.527 | 10/4/1996 | Arjen K. Lenstra et al. |
| RSA-140 | 140 | 463 | $17.226 | 2/2/1999 | Herman J. J. te Riele et al. |
| RSA-150 | 150 | 496 | — | 16/4/2004 | Kazumaro Aoki et al. |
| RSA-155 | 155 | 512 | $9.383 | 22/8/1999 | Herman J. J. te Riele et al. |
| RSA-160 | 160 | 530 | — | 1/4/2003 | Jens Franke et al., University of Bonn |
| RSA-200 | 200 | 663 | — | 9/5/2005 | Jens Franke et al., University of Bonn |
| RSA-576 | 174 | 576 | $10.000 | 3/12/2003 | Jens Franke et al., University of Bonn |
| RSA-640 | 193 | 640 | $20.000 | 2/11/2005 | Jens Franke et al., University of Bonn |
| RSA-704 | 212 | 704 | $30.000 | — | — |
| RSA-768 | 232 | 768 | $50.000 | — | — |
| RSA-896 | 270 | 896 | $75.000 | — | — |
| RSA-1024 | 309 | 1024 | $100.000 | — | — |
| RSA-1536 | 463 | 1536 | $150.000 | — | — |
| RSA-2048 | 617 | 2048 | $200.000 | — | — |

▸ Implementação





# Limitações

Em C, assim como a generalidade das linguagens de programação, o tipo `int`, implementação dos números inteiros, tem uma gama de variação limitada.

Em C (`unsigned`, de zero até):

- unsigned int — dois «bytes» — 65.535
- unsigned long — quatro «bytes» — 2.147.483.647
- unsigned long long (C99) — oito «bytes» — 18.446.744.073.709.551.615

Insuficientes para garantir a segurança da Cifra RSA ($\geq 704$bits).





# Inteiros de Gama (quase) Infinita

**GMP — GNU Multiple Precision Arithmetic Library**

O que é a biblioteca GMP?

O GMP é uma biblioteca livre para a aritmética de precisão e gama de variação arbitrárias, implementa inteiros com sinal, racionais, e reais.

Não tem limites fixos para a precisão ou gama de variação, que não sejam os impostos pelas limitações em memória do sistema computacional que se está a usar.

A biblioteca GMP possuí um conjunto muito rico de funções, sendo que cada uma delas possuí um interface normalizado.

As principais aplicações da biblioteca GMP são, sistemas criptográficos, segurança da Rede, sistemas algébricos, etc.

http://gmplib.org/





# GMP — Breves Notas

- Para se poder trabalhar com a biblioteca GMP é necessário incluir o ficheiro de cabeçalhos apropriado:

```
#include <gmp.h>
```

- Na fase de compilação é necessário fazer a ligação à biblioteca `libgmp`:

```
gcc exemplo.c -lgmp
```

Makefile:

```
CC = gcc
FLAGS = -lgmp
   ⋮
rsa: funcoesRSA.o chavesRSA.o
        $(CC) rsa.c funcoesRSA.c chavesRSA.c $(FLAGS) -o rsa
```

- O tipo para os inteiros de gama de variação infinita é `mpz_t`, por exemplo:

```
mpz_t n;
```

# GMP — Classes de Funções

Existem 6 classes de funções na biblioteca GMP. As mais relevantes (para o fim em vista) são:

- Funções para a aritmética inteira: os nomes começam por `mpz_`; o tipo associado é `mpz_t`; existem cerca de 150 funções nesta classe;
- Funções de baixo nível eficientes para inteiros: os nomes começam por `mpn_`; o tipo associado é uma matriz de `mp_limb_t`; Existem cerca de 30 funções nesta classe;
- Miscelânea (geração de números pseudo-aleatórios, ...);

Por analogia com a instrução de atribuição as funções na biblioteca GMP têm, em geral, os parâmetros de saída, antes dos argumentos de entrada.





# GMP — Inicialização

Antes de se usar uma variável GMP é necessário inicializá-la. Por exemplo:

```
void exemplo (void) {
 mpz_t n;
 int i;
 mpz_init (n);
 for (i = 1; i < 100; i++) {
  mpz_mul (n, ...);
  mpz_fdiv_q (n, ...);
  ...
 }
 mpz_clear (n);
}
```





# GMP — Convenções sobre Parâmetros

As variáveis GMP quando usadas como parâmetros de funções são efectivamente **passadas por referência** («*call-by-reference*»), isto dado que é o ponteiro que é passado no mecanismo usual de passagem por valor do C. Se se pretender forçar a utilização de um parâmetro de entrada (e somente de entrada) pode-se designar o referido parâmetro como **const** para provocar um erro ou um aviso por parte do compilador, caso se tente modificar o mesmo.

Quando uma função definida pelo utilizador tem como resultado um elemento do um dos tipos GMP deve definir um parâmetro de saída que obtêm o referido resultado, isto da mesma forma como as funções internas o fazem. A utilização da instrução `return` para um objecto dos tipos BMP não vai devolver o objecto mas sim um ponteiro para o mesmo, o que mais que provavelmente, não é o que se pretende.





# GMP — Exemplo

Exemplo de uma função que manipula objectos BMP.

```
#include <gmp.h>

void exemplo(mpz_t resultado, const mpz_t parametro, unsigned long n) {
 unsigned long i;

 mpz_mul_ui(resultado, parametro, n);
 for (i = 1; i < n; i++)
   mpz_add_ui(resultado, resultado, i*7);
}

int main (void) {
 mpz_t r, n;
 mpz_init (r);
 mpz_init_set_str(n, "123456", 0);
 exemplo(r, n, 20L);
 gmp_printf("%Zd\n", r);
 return 0;
}
```

- `mpz_mul_ui(producto, factor_grande, factor_pequeno)`, multiplica um inteiro «mpz» por um inteiro sem sinal, colocando o resultado em `produto`
- `mpz_add_ui(soma, parcela_grande, parcela_pequena)`, soma um inteiro «mpz» por um inteiro sem sinal, colocando o resultado em `soma`.

# Criptoanálise da Cifra RSA — Bibliografia


- Pedro Quaresma e Augusto Pinho, «Criptoanálise», Gazeta de Matemática, Gazeta de Matemática 157, 22–31, Abril de 2009, SPM, Lisboa.
- Hans Riesel. Prime Numbers and Computer Methods for Factorization, volume 126 of Progress in Mathematics. Birkhäuser, 2nd edition, 1994.
- Richard Crandall e Carl Pomerance, Prime Numbers. A computacional Prespective, Springer, 2nd Edition, 2005.
- D. Atkins, M. Graff, A. Lenstra, and P. Leyland. The magic words are squeamish ossifrage. In ASIACRYPT 1994, pages 263–277, 1994.






# Criptoanálise da Cifra RSA

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.





# Criptoanálise da Cifra RSA

RSA - dado um $n \in \mathbb{N}, n = pq$, com $p$ e $q$ primos, factorizar $n$ nos seus factores primos.

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.





# 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.
Uma dos pontos fracos desta aproximação é a necessidade que se tem de gerar os números primos até um determinado $m$.

- não é actualmente conhecida nenhuma fórmula para a geração de números primos.
- *Crivo de Eratóstenes*.

A necessidade de criar a lista de todos os inteiros de $2$ a $n$, e as inúmeras vezes ($\lfloor \sqrt{n} \rfloor$ 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.

# Fórmula $p = 6k \pm 1$

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

$$p = 6k \pm 1$$

.





# Método de Euclides

Este método ganha o seu nome da utilização do algoritmo de Euclides para o cálculo do máximo divisor comum de dois inteiros.

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:

- *Crivo de Eratóstenes*.
- 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, ou então utilizar a biblioteca GMP.





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





# Método de Fermat – Continuação

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.

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.

Ou seja o método de Fermat leva-nos a escolher, para a nossa cifra, $p$ e $q$ primos distantes entre si.

# Método de Fermat – Exemplo

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:

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





# Estudo Comparativo dos Vários Métodos

Fazendo um estudo comparativo dos vários métodos temos:

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

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.





# Criptoanálise RSA

Para evitar a utilização de algoritmos de factorização conhecidos, no método criptográfico RSA, $p$ e $q$ devem ser escolhidos satisfazendo algumas condições.

| Algoritmo | Restrição sobre $p$ e $q$ |
|---|---|
| Divisões (Crivo de Eratóstenes) | $p$ e $q$ devem ser «grandes» |
| Fermat | $\lvert p - q \rvert$ deve ser «grande» |
| Pollard $p-1$ | $p-1$ e $q-1$ devem ser múltiplos de, pelo menos, uma potência «grande» de um número primo |
| Curvas elípticas | $p$ e $q$ devem ter aproximadamente o mesmo número de bits |
| Crivo quadrático | $p$ e $q$ devem ter, pelo menos, 1024 bits cada |





# ElGamal — Geração de Chaves

A segurança da cifra ElGamal está baseada na intractibilidade do problema do logaritmo discreto.
A geração de chaves segue o padrão de geração de chaves das cifras de chave pública, cada entidade gera um par de chaves, e concordam numa forma de distribuir as chaves públicas.

## Geração de Chaves

Cada entidade $A$ deve proceder do seguinte modo:

1. Gera de forma aleatório um primo de grande dimensão, e $g$, uma raiz primitiva módulo $p$ (algoritmo H4.84).

### Definição (Raiz Primitiva)

*$g$ é uma raiz primitiva módulo $n$ se todo número $a$ coprimo com $n$ for congruente com uma potência de $g$ módulo $n$.*
*Ou seja, $g$ é uma raiz primitiva módulo $n$ se para cada inteiro $a$ coprimo com $n$, existe um inteiro $k$ tal que $g^k \equiv a \pmod{n}$. Esse valor $k$ é chamado de índice ou logaritmo discreto de $a$ para a base $g$ módulo $n$.*