UNIVERSIDADE D
COIMBRA

José Songo

Criptografia RSA

Projecto de Investigação em Matemática Aplicada e Computação no âmbito do Mestrado em Matemática orientado pelo Professor Doutor Pedro Quaresma

Janeiro de 2024

Criptogtafia de Chave Pública

José Songo

UNIVERSIDADE D
COIMBRA

Mestrado em Matemática

Master in Mathematics

Janeiro 2019 / January 2024

Agradecimentos

Agradeço ao professor Pedro Quaresma, que aceitou orientar o meu projeto, revelando uma especial atenção às minhas ideias. Os seus conselhos e sugestões bem como a valorização do trabalho desenvolvido foram determinantes para alcançar este resultado.

Resumo

A criptografia RSA revolucionou a segurança digital com o seu procedimento de chaves pública e privada, em contraponto com as cifras de chaves simétricas cuja gestão das chaves é muito difícil de gerir em ambientes com muitos intervenientes. A criptoanálise, por sua vez, desempenha um papel fundamental na avaliação e no aprimoramento da segurança dos sistemas criptográficos, buscando constantemente formas de tornar as comunicações digitais mais seguras e protegidas contra ameaças cibernéticas.

Neste texto apresentam-se as implementações dos diferentes métodos descritos no trabalho de seminário, nomeadamente, os algoritmos para os métodos de deslocamento simples, linear e RSA.

No final, foi feito um estudo comparativo dos métodos acima mencionados.

Conteúdo

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Neste texto vão-se apresentar as diferentes implementações efetuadas dos métodos apresentados no trabalho de seminário. Os algoritmos implementados estão em linguagem "*c++*" por instrução do orientador. Por motivos de eficiência consideraram-se apenas as letras de a-z. Além disso, uma vez que os algoritmos são subprogramas, ou seja, para serem usados noutros programas, como o correio eletrónico, optou-se por utilizar os argumentos na linha de comando. Por último, escolheu-se ler e escrever em ficheiro pois esses algoritmos são usados noutros programas onde têm de receber e enviar textos em ficheiros.

O texto segue as notações e terminologias apresentadas no trabalho de seminário [2].

Capítulo 2

Criptografia Clássica

Relativamente à criptografia de deslocamento (cifras pré-computacionais), cifras simples caracterizadas pelo deslocamento que estabelecem nas letras do alfabeto, vão ser implementadas as cifras de deslocamento simples e linear. Para isto, primeiramente, é realizada uma pré-codificação aonde os letras do nosso alfabeto irão ser convertidas em números. Em seguida, executados códigos elaborados em *C++* com vista a implementar a encriptação, desencriptação e o quebrar da cifra.

Pré-codificação

Vamos supor, por simplificação, que a mensagem original contém apenas letras minúsculas e espaços entre palavras. Esta primeira etapa vamos designar por **pré-codificação**.

Na codificação as letras vão ser convertidas em números de acordo com a tabela seguinte (ver Tabela 2.1):

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Tabela 2.1 Tabela de Conversão

Por exemplo, a frase “criptografia”. Inicialmente vai ser codificada no conjunto de algarismos:

2 17 8 15 19 14 6 17 0 5 8 0

Ao aplicarmos a cifra deslocamento linear, com chave 5, obtém-se:

7 22 13 20 24 19 11 23 5 10 15 5

Finalmente, convertendo novamente para letras vamos obter

hwnuytlxfkpf

2.1 Cifra de Deslocamento Simples

A abordagem da cifra de deslocamento simples é uma cifra clássica, onde a segurança de uma mensagem é assegurada através da substituição de cada letra por outra situada a um número fixo

de posições à "frente" no alfabeto. Este método, também conhecido por ser um caso particular do deslocamento linear, apresenta uma técnica direta e importante na história da criptografia, baseada na ideia de variações sistemática das letras para assegurar a confidencialidade das informações. No âmbito prático, exploramos não apenas o processo de encriptação e desencriptação, mas também a criptoanálise, revelando como a busca exaustiva no espaço das chaves permite a quebra da cifra e a recuperação do texto original. Este estudo detalhado revela os princípios e vulnerabilidades desse método clássico, ressaltando a importância de procedimentos mais robustos na proteção de dados sensíveis nos tempos modernos [2, Cap.4].

Para desenvolver a cifra de deslocamento em *C++*, é necessário criar as funções de encriptação, desencriptação e quebra-cifra. Além disso, algumas funções auxiliares, como a conversão de letras para números e de números para letras, devem ser criadas. Logo, tem-se:

2.1.1 Cifra de Deslocamento Simples—Encriptação

Para esta segunda etapa, precisamos, de conhecer, além do texto claro, o valor da chave (deslocamento) $0 < k < |\mathscr{A}|$. Tem-se:

↪ mensagem a cifrar, chave;

encriptar número a número através da função de encriptação;

↩ texto cifrado.

Especificação:

```
int cifraEncriptar(int ,int ,int ); //Encripta a letra
```

Algoritmo 2.1 Cifra de Deslocamento Simples—Especificação de `cifraEncriptar`

Implementação:

```
void encriptar(int k,fstream& fin,fstream& fout){ //Encripta mensagens
    int i=0,aux;
    string linha;
    string nomeMsg;
        while (getline(fin, linha)) {//Obter as linhas do ficheiros
              while(linha[i]!='\0'){
                aux=conversaoDS_LN(linha[i]); //Letras --->numeros
                aux=cifraEncriptar(aux,k,26); //Aplicação de encriptação
                i+=1;
                if(aux >26){
                    aux=aux%26;}
                fout << conversaoDS_NL(aux);}
            fout << endl;
            i=0;}
        fout.close(); //fechar ambos os ficheiros
        fin.close();}
```

Algoritmo 2.2 Encriptar

Um exemplo de utilização desta cifra, vai-se encriptar o seguinte texto:

```
a  criptografia classica
deslocamento linear
universidade de coimbra
```

Para cifra de Julho César, $k = 3$, vamos obter o seguinte texto cifrado

```
dNfulswrjudildNfodvvlfdN
ghvorfdphqwrNolqhdu
xqlyhuvlgdghNghNfrlpeud
```

Nota: 'N' corresponde a valores que estão fora do nosso alfabeto. Nesse caso, o espaço cujo valor é 26 com a codificação escolhida ficou de fora do nosso alfabeto, tendo-se optado por uma não encriptação nesses casos.

2.1.2 Cifra de Deslocamento Simples—Desencriptação

Na cifra de deslocamento simples, o valor de k da chave de encriptação corresponde ao mesmo valor para a chave de desencriptação. Tem-se:

Desencriptação

↪ mensagem cifrada, chave;

desencriptar número a número através da função de encriptação;

↩ texto original.

Especificação:

```
int cifraDesencriptar(int, int, int, int);//Desencripta a letra
```

Algoritmo 2.3 Cifra de Deslocamento Simples—Especificação de `cifraDesencriptar`

Implementação:

```
void desencriptar(int k,fstream& fin,fstream& fout){//desencripta mensagens
int i=0,aux;
//string nomeMsg;
string linha;
    while (getline(fin, linha)){//Obter as linhas do ficheiro.txt
          while(linha[i]!='\0'){
            aux=linha[i];
            aux=conversaoDS_LN(linha[i]);
            aux=cifraDesencriptar(k,aux,26);
            while(aux<0){
                aux=aux+26;}
            i+=1;
            fout << conversaoDS_NL(aux);}
       fout << endl;
```

```
        i=0;}
    fout.close();
    fin.close();}
```

Algoritmo 2.4 Desencriptar

Continuando do passo anterior, para $k = 3$, vamos obter o seguinte texto decifrado:

```
aNcriptografiaNclassicaN
deslocamentoNlinear
universidadeNdeNcoimbra
```

Neste caso, 'N' que ficou fora do alfabeto corresponde ao espaço entre as palavras.

2.1.3 Cifra de Deslocamento Simples—Quebrar a Cifra

Relativamente a criptoanálise, para esta cifra efetuou-se um ataque por procura exaustiva no espaço das chaves, ou seja, verificaram-se todas as possíveis chaves no espaço das chaves.

↪ mensagem a cifrada;

desencriptar número a número através da função de "quebra cifra";

↩ texto original.

Especificação:

```
int cifraDesencriptar(int , int ,int,int);//Desencripta a letra
```

Algoritmo 2.5 Cifra de Deslocamento Simples—Especificação de `cifraDesencriptar`

Implementação:

```
void quebraCifra(string nome){
int aux = 0;
string nomeMsg;
string linha;
fstream fd;
for (int k = 1; k <= 26; k++) {
fstream fin(nome, std::ios::in | std::ios::binary);// abrir o ficheiro
if (!fin.is_open()) {
    cerr << "Impossível␣abrir␣o␣arquivo␣" << endl;
    return; // no caso de não conseguir abrir o ficheiro}
aux = aux + 1;
nomeMsg = "MensagemDecifrada" + to_string(aux) + ".txt";
fd.open(nomeMsg, std::ios::out);
if (!fd.is_open()) {
    cerr << "Impossível␣criar␣o␣arquivo␣" << nomeMsg << endl;
    return;
}
desencriptar(k, fin, fd);
```

```
fd.close();
//fin.clear(); // Clear the stream state
fin.seekg(0); // apontar para o inicio
fin.close();}}
```

Algoritmo 2.6 Quebrar a cifra

Portanto geraram-se 27 ficheiros do tipo texto (.txt) dos quais apresentam-se, em seguida, os 4 primeiros resultados :

```
dqfulswrjudildqfodvvlfd
ghvorfdphqwrqolqhdu
xqlyhuvlgdghqghqfrlpeud

cpetkrvqitchkcpencuukec
fgunqecogpvqpnkpgct
wpkxgtukfcfgpfgpeqkodtc

bodsjquphsbgjbodmbttjdb
eftmpdbnfoupomjofbs
vojwfstjebefoefodpjncsb

ancriptografianclassica
deslocamentonlinear
universidadendencoimbra
```

Assim, através da análise dos textos anteriores, é evidente que o último deles, escrito em uma linguagem percetível, corresponde ao nosso texto original.

Iremos de seguida repetir os procedimentos descritos, mas agora para o deslocamento linear. Será fácil de ver que a cifra descrita acima é uma cifra de deslocamento linear, para o caso particular em que o valor da componente multiplicativa da chave, a, é 1.

2.2 Cifra de Deslocamento Linear

Esse método, também conhecida como cifra afim, procede através de uma função matemática que move cada letra da mensagem original de acordo com uma chave de encriptação (a,b). Este procedimento é fundamental para garantir a proteção das informações transmitidas, ao substituir as letras por outras correspondentes na mensagem cifrada.

O propósito deste estudo é explorar em detalhes os processos de encriptação e desencriptação nessa cifra. Além disso, será abordada a criptoanálise, um método aplicado para quebrar a cifra por meio de ataques de força bruta, testando todas as possíveis chaves no espaço das chaves.

Seguidamente, analisaremos os resultados de tentativas de quebra da cifra, identificando a abordagem bem-sucedida para recuperar a mensagem original. Essa análise minuciosa permite compreender a funcionalidade e a fragilidade desse método clássico de criptografia diante de ataques de força bruta e sua importância na segurança da informação.

Para implementar a cifra de deslocamento linear precisamos de criar, primeiramente, funções secundárias para calcular o inverso multiplicativo e conversão de letra para número e vice-versa para auxiliar as funções, nomeadamente, as cifras de encriptação e desencriptação, que por sua vez, completam as funções de encriptação, desencriptação e quebra-cifra.

2.2.1 Cifra de Deslocamento Linear—Validação da Chave

Testar as modificações da validação

Para utilizar as cifras de deslocamento linear, nomeadamente, as cifras de desencriptação e quebra cifra precisamos, previamente, validar a chave de encriptação. Para isso, tem-se que verificar se a tem inverso multiplicativo modulo n.

Especificação:

```
//Encontrar o inverso multiplicativo
int inversoMultiplicativo(int,int);
```

Algoritmo 2.7 Cifra de Deslocamento Simples—Especificação de `cifraDesencriptar`

```
//calcula o inverso multiplicativo de b mod n
int inversoMultiplicativo(int n,int b){
int  n0=n;
int t0=0;
int t=1;
int q=n0/b;
int r=n0-q*b;
int temp;
while(r > 0){
    temp=t0 -q*t;
    if (temp >=0)
        temp=temp%n;
    else
        temp=n-((-temp)%n);
//Atualização dos valores
t0=t;
t=temp;
n0=b;
b=r;
q=n0/b;
r=n0-q*b;
}
if(b==1)
    return t;
else
    return 0;}
```

Algoritmo 2.8 Validação da chave

2.2.2 Cifra de Deslocamento Linear—Encriptação

Para encriptar a mensagem precisamos saber a chave de encriptação (a, b). Tem-se:

Encriptação

↪ mensagem a cifrar, a, b, os valores para a chave foram validados previamente;

encriptar número a número através da função de encriptação;

↩ texto cifrado.

Especificação:

```
//encripta o texto
void encriptar(int,int,fstream&,fstream&);
```

Algoritmo 2.9 Cifra de Deslocamento Linear—Especificação de `cifraEncriptar`

```
void encriptar(int a, int b,fstream& fin,fstream& fout){//Encripta mensagens
int i=0,aux;
string linha;
string nomeMsg;
while (getline(fin, linha)) {//Obter as linhas do ficheiros
  while(linha[i]!='\0'){aux=conversaoDS_LN(linha[i]);//Letras --> Numeros
    aux=cifraEncriptar(a,b,aux,26);//Aplicação de encriptação
    i+=1;
    if(aux >26){
        aux=aux%26;
    }
    fout << conversaoDS_NL(aux);
}
fout << endl;
i=0;}
fout.close();//fechar ambos os ficheiros
fin.close();}
```

Algoritmo 2.10 Encriptação

Para a chave $(2, 1)$ vamos cifrar o texto seguinte:

```
deslocamento linear
projeto e seminario
portugal e coimbra
```

Obtém-se o seguinte texto cifrado

```
dNfulswrjudildNfodvvlfdN
ghvorfdphqwrNolqhdu
xqlyhuvlgdghNghNfrlpeud
```

2.2.3 Cifra de Deslocamento Linear—Desencriptação

Desencriptação:

↪ mensagem a cifrada, a, b;

validar a chave

encriptar número a número através da função de encriptação;

↩ texto original.

Especificação:

```
//desencripta o texto
void desencriptar(int , int ,fstream& ,fstream& );
```

Algoritmo 2.11 Cifra de Deslocamento Linear—Especificação de `cifraDesencriptar`

```
void desencriptar(int a, int b,fstream& fin,fstream& fout){//desencripta mensagens
int i=0,aux;
//string nomeMsg;
string linha;
    while (getline(fin, linha))  {//Obter as linhas do ficheiro.txt
        cout<<linha;
        cout<<endl<<"De mili ne";
          while(linha[i]!='\0'){
            aux=linha[i];
            aux=conversaoDS_LN(linha[i]);
        aux=cifraDesencriptar(a,b,aux,26);//cifra para desencriptar
            while(aux<0){
                aux=aux+26;}
            i+=1;
            fout << conversaoDS_NL(aux);}
        fout << endl;
        i=0;}
    fout.close();
    fin.close();}
```

Algoritmo 2.12 Desencriptação

Usando a mesma chave utilizada anteriormente para encriptar a mensagem obteve-se o seguinte texto cifrado:

```
aNcriptografiaNclassicaN
deslocamentoNlinear
universidadeNdeNcoimbra
```

2.2.4 Cifra de Deslocamento Linear—Quebrar a cifra

Relativamente a criptoanálise, uma vez que todas as cifras clássicas são suscetíveis a ataque de força bruta, para esta cifra realizou-se um ataque por procura exaustiva no espaço das chaves, ou seja, mais uma vez verificaram-se todas as possíveis chaves, neste caso, válidas no espaço das chave.

↪ mensagem a cifrada;

desencriptar número a número através da função de "quebra cifra";

↩ texto original.

Especificação:

```
//quebra a cifra gerando todas as combinações possíveis do texto cifrado
void quebraCifra(string );
```

Algoritmo 2.13 Cifra de Deslocamento Linear—Especificação de `cifraQuebraCifra`

```
void quebraCifra(string nome) {
int k = 0;
string nomeMsg;
string linha;
fstream fd;
for (int b = 0; b <= 26; b++) {
for (int a = 1; a <= 26; a++) {
    fstream fin(nome, std::ios::in | std::ios::binary); // abrir o ficheiro
    if (!fin.is_open()) {
        cerr << "Impossível␣abrir␣o␣arquivo␣" << endl;
        return; // no caso de não conseguir abrir o ficheiro}
    k = k + 1;
    nomeMsg = "MensagemDecifrada" + to_string(k) + ".txt";
    fd.open(nomeMsg, std::ios::out);
    if (!fd.is_open()) {
        cerr << "Impossível␣criar␣o␣arquivo␣" << nomeMsg << endl;
        return; }
    desencriptar(a, b, fin, fd);
    fd.close();
    //fin.clear(); // Clear the stream state
    fin.seekg(0); // Move file pointer to the beginning of the file
    fin.close();}}
// Close the file outside the loops}
```

Algoritmo 2.14 Quebrar a cifra

```
dqfulswrjudildqfodvvlfd
ghvorfdphqwrqolqhdu
xqlyhuvlgdghqghqfrlpeud

cpetkrvqitchkcpencuukec
fgunqecogpvqpnkpgct
wpkxgtukfcfgpfgpeqkodtc

bodsjquphsbgjbodmbttjdb
eftmpdbnfoupomjofbs
vojwfstjebefoefodpjncsb
```

```
ancriptografianclassica
deslocamentonlinear
universidadendencoimbra
```

Portanto por interpretação dos textos acima, obviamente, que o último do textos dos 4 apresentados, que apresenta uma linguagem que permite ser compreendida, corresponde ao nosso texto original.

Logo conseguimos quebrar a cifra, obtendo a chave de encriptação.

Capítulo 3

Criptografia RSA

A implementação do algoritmo RSA é bastante utilizado para assegurar a segurança das comunicações digitais. Este algoritmo baseia-se em um par de chaves: uma chave pública utilizada para encriptar os dados e uma chave privada correspondente para desencriptar.

Na encriptação RSA, o remetente utiliza a chave pública do destinatário para transformar os dados originais em um formato encriptado. O resultado é um texto encriptado, que pode ser enviado com segurança.

Por outro lado, na desencriptação RSA, o destinatário utiliza sua chave privada correspondente para reverter o processo anterior. Com isso, o destinatário obtém novamente os dados originais, protegendo a confidencialidade da informação.

A segurança do algoritmo RSA baseia-se na dificuldade de fatorizar números primos de grande dimensão, pois as chaves pública e privada são geradas a partir desses números.[1]

Portanto, uma implementação bem-sucedida da cifra RSA requer desenvolver corretamente algoritmos matemáticos, incluindo geração de chaves, encriptação e desencriptação garantindo assim a segurança das comunicações digitais. Também são criadas funções secundárias que auxiliam as funções mencionadas anteriormente, nomeadamente de conversão e de operação de congruência.

3.1 Cifra RSA—Geração das Chaves

Para empregar o algoritmo RSA, em primeiro lugar, é preciso gerar chaves pública e privada para cifrar ou decifrar a mensagem, respetivamente.

Seja (e,n) a chave pública e (d,n) a chave privada, onde e tal que $1 < e < \phi(n)$ é primo relativo com $\phi(n)$ e d é o inverso multiplicativo de e módulo $\varphi(n)$, ou seja $de \equiv 1(mod\phi(n))$.

↪ p, q, dois números primos de grande dimensão;

↩ *(e,n)* e *(d,n)*, as chaves: pública e privada

```
int* gerarChavePrivada(int , int ,int );
```

Algoritmo 3.1 Cifra RSA —Especificação de `GerarChavePrivada`

[1] Size considerations for public and private keys, https://www.ibm.com/docs/en/zos/2.3.0?topic=certificates-size-considerations-public-private-keys

```
int* gerarChavePrivada(int p, int q,int e){
// Aloca dinamicamente um array de 2 inteiros
    int* chaveP = (int*)malloc(2 * sizeof(int));

if (chaveP == NULL) {
    // Se a alocação falhar
    return NULL;
}
chaveP[0]=inversoMultiplicativo((p-1)*(q-1),e);
chaveP[1]=p*q;
return chaveP;
}
```

Algoritmo 3.2 Gerar chave privada

```
int* gerarChavePublica(int ,int );
```

Algoritmo 3.3 Cifra RSA —Especificação de `GerarChavePública`

```
int* gerarChavePublica(int q,int p){
// Aloca dinamicamente um array de 2 inteiros
int* chaveP = (int*)malloc(2 * sizeof(int));
int omega_n=(p-1)*(q-1);
if (chaveP == NULL) {
    // Se a alocação falhar
    return NULL;}
//chaveP[1]=(p-1)*(q-1);
chaveP[1]=p*q;
//calcular e -->pequeno para melhorar a encriptação
for(int i=2;i<omega_n;i++){

    if(ePrimo(i,omega_n)){
        chaveP[0]=i;
        break;}}
    return chaveP;}
```

Algoritmo 3.4 Gerar chave privada

3.2 Cifra RSA—Encriptação

Para a encriptação RSA tem-se:

↪ mensagem a cifrar, a chave pública, (e,n);

encriptar número a número através da função de encriptação;

↩ texto cifrado.

```
int cifraEncriptar(int,int,int);
```

Algoritmo 3.5 Cifra RSA —Especificação de `CifraEncriptar`

```
void encriptar(int e,int n,fstream& fin,fstream& fout){
int i=0,aux;
string linha;
string nomeMsg;
while (getline(fin, linha)) {//Obter as linhas do ficheiros
      while(linha[i]!='\0'){
        //converter as letras da linha em numeros
        aux=conversaoDS_LN(linha[i]);
        aux=cifraEncriptar(e,n,aux);//Aplicação de encriptação
        i+=1;
        if(aux > n){
            aux=aux%(n);}
        fout <<aux<<"␣";}
    fout << endl;
    i=0;}
fout.close();//fechar ambos os ficheiros
fin.close();}
```

Algoritmo 3.6 Encriptação RSA

3.3 Cifra RSA—Desencriptação

Ao receber um texto cifrado, o destinatário precisa seguir um conjunto de passos para obter a mensagem original. O processo de desencriptação inicia-se com a mensagem a ser decifrada, que foi previamente cifrada usando a chave pública correspondente. A chave de desencriptação, mantida em segredo pelo destinatário, é indispensável para este procedimento.

Portanto a geração da chave de desencriptação é um passo essencial, pois é por meio dela que a informação codificada pode ser revertida ao seu estado original. Esta chave, em conjunto com a função de desencriptação associada ao algoritmo RSA, permite o processo de decifração do texto original.

Logo ao desencriptar número a número através da função de desencriptação utilizando a chave privada correspondente, o destinatário é capaz de recuperar o texto original, assegurando a privacidade e a segurança das informações transmitidas.

Portanto para desencriptação RSA tem-se:

↪ mensagem a decifrar;

desencriptar número a número através da função de desencriptação;

↩ texto original.

```
void desencriptar(int,int ,fstream& ,fstream& );
```

Algoritmo 3.7 Cifra RSA —Especificação de `CifraDesencriptar`

```
void desencriptar(int p,int q,fstream& fin,fstream& fout){
int i=0,aux,d;
int* chaveP = (int*)malloc(2 * sizeof(int)); // Aloca dinamicamente array
if (chaveP == NULL){
    // Se a alocação falhar
    cout<<"Alocação falhou...";}
chaveP=gerarChavePublica(p,q);
d=inversoMultiplicativo(chaveP[1],chaveP[0]);
string nomeMsg;
string linha;
    while (getline(fin, linha)) {//Obter as linhas do ficheiros
          istringstream linhaAux(linha);
          while(linhaAux >> aux){
          //Aplicação de encriptação
          aux=cifraEncriptar(d,p*q,aux);
            i+=1;
            if(aux > (p*q)){
                aux=aux%(p*q);}
            fout << conversaoDS_NL(aux);}
        fout << endl;
        i=0;}
    //fechar ambos os ficheiros
    fin.close();fout.close();}
```

Algoritmo 3.8 Gerar chave privada

Capítulo 4

Criptoanálise RSA

O sistema RSA é assimétrico, ou seja, possui uma chave pública (e,n) e uma chave privada (d,n), em que [1]:

$n = p \times q$ com p e q números primos;

$1 < e < \phi(n)$, em que $\phi(n)$, a funçao de Euler, é igual ao número co-primos de n;

$de \equiv 1(mod\phi(n))$

Sabemos que a chave pública é (e,n), portanto apenas tem-se que fatorizar n no produto de números primos p e q para obtermos d e, com isso, a chave secreta (d,n). No entanto, fatorizar um número em fatores primos é um problema computacionalmente difícil [1].

Para a implementação dos algoritmos a seguir optou-se pelo tipos de dados "*unsigned long long int*" uma vez que apenas iremos lhe dar com números positivos de grande dimensão.

4.1 Método da Divisão

Sendo um método de força bruta, vai-se tentar a divisão sucessiva por todos os números primos até $\lfloor\sqrt{n}\rfloor$, ou até que a solução seja encontrada.

```
unsigned long long int  metodoDivisao(unsigned long long int  );
```

Algoritmo 4.1 Criptoanálise RSA —Especificação de `MétodoDivisao`

```
unsigned long long int  metodoDivisao(unsigned long long int  n){
unsigned long long int  *resCrivo = new unsigned long long int [n + 1],i;
crivo(n,resCrivo);//Determinar os números primos
for(i=2;i<=n;i++){
    if(resCrivo[i]){//verificar quais dos numeros sao primos
        if((n%i)==0){
            return i;//retornar numero primo
        }
    }
 }
 return 0;
```

```
}
```

Algoritmo 4.2 Método de Divisão

Primeiramente, é necessário utilizar um método que elabore a lista de todos os números primos até um limite definido, e neste caso, foi utilizado o Crivo de Eratóstenes.

Crivo de Eratóstenes

começa-se por gerar um vetor com todos os números de 2 até n;

2 é primo;

Então utiliza-se o 2 para retirar todos os seus multiplos da lista, e assim sucessivamente;

os elementos que restar após as sucessivas aplicações constituem os números primos de 2 até n.

A necessidade de gerar a lista completa de todos os inteiros de 2 até n e iterar sobre ela várias vezes, pois é necessário percorrer essa lista para aplicar os sucessivos crivos, leva a que a utilização do *crivo de Erastóstenes* acrescente um peso muito significativo ao algoritmo, tanto temporal como especialmente [1].

4.2 Método de Euclides

Este algoritmo consiste em multiplicar todos os números entre 2 e $\lfloor\sqrt{n}\rfloor$, calcular de seguida o *mdc* entre esse produto e n, de forma a encontrar o fator primo pretendido.

Para evitar o problema de representação computacional dos números que se obtém do produto dos números primos de grande dimensão, vai dividir-se a multiplicação em vários produtos parcelares.

Para tal, os passos seguintes são adotados:

- Começa-se por definir os conjuntos auxiliares:

  $R = r_1, r_2, \ldots, r_n$, representando r_i um limite inferior $(r_1 = 1,\ r_i < r_{i+1})$;

  $S = s_1, s_2, \ldots, s_n$, representando s_i um limite superior$(s_i < si+1, s_{m-1} < \lfloor\sqrt{n}\rfloor < s_m)$;

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

- Para cada um dos P_i calcula-se o $mdc(P-i,n) = a_i$;

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

```
unsigned long long int metodoEuclides(unsigned long long int );
```

Algoritmo 4.3 Criptoanálise RSA —Especificação de `MétodoEuclides`

```
unsigned long long int  metodoEuclides(unsigned long long int  n){
unsigned long long int  *resCrivo = new unsigned long long int [n + 1],
i,aux,lim,res=1;
aux=floor(sqrt(n));
lim=(aux/10)+1;
unsigned long long int  *resM= new unsigned long long int [aux+1];
for(i=0;i<=lim;i++){
    resM[i]=1;
    }
crivo(n,resCrivo);//encontrar os numeros primos
for(i=2;i<=aux;i++){
    if(resCrivo[i]){
        resM[(i/10)]=i*resM[i/10];//calcular o produto
    }
}
i=0;
while(i<lim){
    if(mdc(resM[i],n)> 1){//verifcar mdc(resM[i],n]
        return mdc(resM[i],n);
    }
    i++;
}
delete[] resCrivo;
delete[] resM;
return res;
}
```

Algoritmo 4.4 Método de Euclides

Mais uma vez, tivemos de gerar números primos até n que é pesado temporalmente. Além disso, apenas atenuamos o problema anterior quando fizemos a multiplicação parcelar para contornar o obstáculo de representação computacional relativamente ao resultado da multiplicação dos números primos uma vez que à medida que n aumentar voltaremos a ter o mesmo problema.

4.3 Método de Fermat

Este método consiste em obter dois inteiros a e b que permitam representar o número natural n como a diferença de dois quadrados [1].

Para determinar os inteiros a e b de forma que $n = a^2 - b^2$, o processo pode ser conduzido da seguinte maneira:

Dado um inteiro n ímpar começamos por $a = \lfloor \sqrt{n} \rfloor + 1$

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;

```
unsigned long long int* metodoFermat(unsigned long long int );
```

Algoritmo 4.5 Criptoanálise RSA —Especificação de `MétodoFermat`

```
unsigned long long int* metodoFermat(unsigned long long int n){
long double a,b;
unsigned long long int *res= new unsigned long long int[2];
a=floor(sqrt(n))+1;
b=sqrt(a*a-n);
//retorna o maior número inteiro menor ou n
while(abs(b - floor(b)) > std::numeric_limits<long double>::epsilon()){
a=a+1;//incrementar a
b=sqrt(a*a-n);//incrementar b
}
//calcular p e q
res[0]=a+b;
res[1]=a-b;
return res;
}
```

Algoritmo 4.6 Método de Fermat

Como veremos mais a frente, este método mostra que quanto maior a diferença entre p e q maior vai ser o número de tentativas precisas para obter um valor inteiro para a raiz.

4.4 Estudo Comparativo dos Métodos

Os dados foram obtidos a partir de testes realizados em um ambiente computacional, utilizando o sistema Windows 10 Home, processador Intel(R) Core(TM) i3-7020U CPU 2.30GHz e memória RAM de 4 GB. Pode-se afirmar que, se a seleção dos números primos for apropriada, a segurança da cifra RSA é mantida.

Tabela 4.1 Estudo comparativo dos métodos

n	Fatores	Divisão	Euclides	Fermat
1457	$p = 31, q = 47$	0,000s	0,000s	0,000s
13199	$p = 67, q = 197$	0,002s	0,002s	0,000s
281161	$p = 79, q = 3559$	0,136s	0,145s	0,000s
701123	$p = 3559, q = 197$	0,521s	0,482s	0,003s
23420707	$p = 41017, q = 571$	71,124s	64,86s	0,000s
488754769	$p = 110503, q = 4423$	–	–	0,005s
2027651281	$p = 46061, q = 41017$	–	–	0,000s
103955963689	$p = 47188363, q = 2203$	–	–	1,75s
210528952589	$p = 95564663, q = 2203$	–	–	3,635s
2746662891777043	$p = 47188363, q = 58206361$	–	–	0,024s
4509540007616669	$p = 47188363, q = 95564663$	–	–	0,299s

Com efeito, por observação da tabela *Estudo comparativo dos métodos* constata-se que:

Os métodos de Divisão e Euclides apresentam um aumento considerável no tempo de execução à medida que os fatores primos crescem significativamente, além de perderem eficácia na resolução do problema mesmo para valores relativamente pequenos de $n = p \times q$. Essas

limitações surgem da exigência de gerar números primos até $\lfloor\sqrt{n}\rfloor$ e, no caso do método de Euclides, da multiplicação de números de grande escala.

O método de Fermat observa um aumento muito acentuado em seus tempos de execução com o crescimento da dimensão dos fatores primos; no entanto, uma análise mais detalhada revela que quando os fatores primos estão em proximidade, o método de Fermat demonstra ser altamente eficaz.

Podemos inferir que, para garantir a segurança da criptografia RSA contra os métodos apresentados acima, os fatores primos devem estar consideravelmente distantes um do outro, e o valor de n deve ser maior que 20 dígitos decimais [1].

Presentemente, sobretudo devido a algoritmo mais eficazes, a cifra RSA utiliza valores de n com 1024 bits (a que corresponde 309 dígitos decimais) ou mais [1].[1]

Capítulo 5

Conclusões

Ao longo do trabalho constatou-se o procedimento eficaz e seguro para criptografar a informação desempenhada pela sistema RSA.

De facto, verifica-se na secção 4 que os métodos analisados não têm capacidade de “quebrar” o método RSA em chaves maiores que 20 dígitos. As chaves seguras utilizadas atualmente para o algoritmo RSA têm cerca de 2048 bits.[1]

No próximo semestre, iremos aprofundar o estudo da criptoanálise do método RSA, com foco especial no crivo quadrático."

[1] Ver página: *Size considerations for public and private keys*, https://www.ibm.com/docs/en/zos/2.3.0?topic=certificates-size-considerations-public-private-keys

Bibliografia

[1] Quaresma, P. and Pinho, A. (2009). Criptoanálise. *Gazeta de Matemática*, 157:22–31.

[2] Songo, J. (2024). Criptografia RSA. Technical report, Departamento de Matemática, Universidade de Coimbra.