Universidade do Minho

Lic. em Matemática e Ciências de Computação
Fundamentos de Criptografia
2006/2007

Trabalho Prático 1: Cifra RSA
Relatório de Desenvolvimento

Rui Carlos A. Gonçalves (41031)

Braga, 22 de Agosto de 2007

Resumo

Neste relatório descreve-se de uma forma sucinta o trabalho realizado no seguimento do primeiro trabalho prático da disciplina de Fundamentos de Criptografia. O objectivo do trabalho era estudar a cifra RSA, implementando as primitivas de geração de chaves, cifragem e decifragem e os esquemas com *padding* (PKCS1.5) e sem *padding*. Foram também estudadas algumas falhas da cifra RSA. O trabalho foi realizado usando a ferramenta computacional *PARI/GP*.

Conteúdo

1 Introdução

A criptografia assimétrica é uma técnica criptográfica em que existem duas chaves: a chave pública, que é divulgada por todos, e a chave privada, que apenas é do conhecimento de quem a gerou. Nestes métodos as mensagens cifradas com a chave pública apenas podem ser decifradas pela chave privada correspondente, assim como as mensagens cifradas com a chave privada só podem ser decifradas com a chave pública. Estas propriedades permitem que esta técnica possa ser usada para garantir autenticidade e confidencialidade.

O RSA é um algoritmo que se enquadra neste tipo de técnica, e foi desenvolvido por Ron Rivest, Adi Shamir e Len Adleman. Descrito em 1977, é ainda hoje uma das técnicas mais usadas. O algoritmo tem por base o problema da factorização de inteiros, que, para valores elevados, se pensa ser computacionalmente intratável.

2 Descrição do Trabalho

Implemente em PARI/GP a geração de chaves RSA assim como as primitivas de cifragem e decifragem RSA.

Implemente também dois esquemas RSA: um sem padding (ou padding a zero) e outro que use a codificação PKCS1.5 (note que, apesar de estar ainda presente no Standard, o seu uso é preterido em favor de esquemas do tipo OAEP).

A implementação deverá cobrir as duas possíveis representações de chave privada. Comente adequadamente, optimize o código e inclua exemplos de execução.

Os seguintes desafios foram todos cifrados sem padding e demonstram algumas más práticas a evitar ou situações susceptíveis de ataque. A dificuldade dos desafios vai aumentando.

Tente recuperar o texto limpo original correspondente a cada um dos criptogramas RSA apresentados. O texto limpo foi dividido em blocos de $k/8$ caracteres, onde k é o tamanho do módulo RSA usado e codificando cada carácter com base no seu código ASCII. Na implementação foram usadas as funções de conversão de dados OS2IP e I2OSP para converter as *strings* de octetos em inteiros para uso das primitivas RSA e vice-versa (ver documentação acima). Em particular, os criptogramas estão apresentados como blocos de *strings* de octetos (em hexadecimal).

- Problema 1

  $e = 5$

  $N =$ 35875741 92954427 23106714 14455212 43178607 58771428 65793354 19222874 94392030 35201

  $C =$ 14 e7 47 40 30 e9 d6 81 63 1e 9c 27 84 3a 61 3a 3a 32 4a 48 e7 e2 db e7 54 4c b4 df 97 51 c6 8d

- Problema 2

  $e = 3$

  $N =$ 10890326 83577364 55066815 31468096 24530428 14775006 20649272 16784759 69098339 19686793 38591464 94098327 68160832 68595581 07942644 64577272 15780237 86494435 78621853 51011217 49241579 47418994 68334776 45334025 67120030 90427026 72365522 90956722 14149022 58969417 44183688 41198294 49701459 19402707 57209267 66820866 41769478 99836600 54201

  $C =$ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 05 1b 2b 27 94 53 9b 1a 7d 1c 51 d9 31 1c 58 d1 70 cc cc 0c 4b ca 28 8f 27 d1 a4 bd 91 d2 b2 4a 0c 54 10 bc 3f ee 2d 47 a9 a4 d5 5b fd 63 47 8d 0f 06 0d 97 68 c0

- Problema 3 - Uma mesma mensagem M foi cifrada para três destinatários distintos com chaves públicas, respectivamente, (N_A, e), (N_B, e) e (N_C, e).

  $e = 3$

  N_A = 11221088 19125288 32873488 73089762 61927054 73369219 61653875 30817045 51444228 28779927 05495181 83502360 74434502 33408237 31796310 20451837 15448390 19675852 32833164 49803848 51027727 54243488 99295466 15023950 78707201 21300501 58370679 94235368 65480798 06643019 02331811 61961115 24880623 90885355 71687505 13262508 51418329 55675633 96901

  C_A = 5d f0 ba 37 0c ed 20 7b ef 9c 80 f2 fa e3 b7 98 91 44 ed 3f 34 5f a8 df 60 92 bb a1 c0 5b 2c ee 4b db b8 8a 91 e3 f2 4e 77 1f 5a 9a e0 12 68 e1 3d d5 8a 8a 29 be 59 fc c6 e1 b5 ed ed 50 c6 1a 28 75 76 59 c5 d9 0c 13 05 8e f2 77 09 5b 04 56 90 2e bc d5 bc 40 84 d0 11 6e 32 d1 7b 80 78 df ca 9d 5c 53 26 ec 89 77 8f d0 25 c6 0b a4 4e fe b6 19 10 15 17 67 58 fa 58 5d 3a 29 bb ad 80 7d

  N_B = 12039209 84865608 81218868 53033797 59281708 28473709 78857568 70024244 22185252 09633774 37040943 51935139 81140455 56006396 09802793 13799625 78333140 68179655 03969227 04908239 80274420 62646596 96670054 17243382 18600061 89185562 58697621 08251423 93988849 70118558 43779219 96904077 36712811 68660000 16593840 83176017 01481588 11123231 15387

  C_B = 8b ba 35 a0 e2 39 4b 76 b4 36 56 bc 59 c7 6b 42 45 a3 e2 86 55 a7 2b bb 6c 2f 03 62 4d 22 da ae dc fc 07 07 82 c8 00 07 bc a4 ec a6 76 9d cf c6 2c 78 1d 1c 51 11 8a ea e9 11 6b 12 1c 55 33 72 28 ca 49 ee 1c 8c d7 f2 33 57 e9 72 38 91 11 b0 b9 35 6b 40 ce b3 5f fa 10 a8 29 1a 5e 3a a9 ee f4 57 af 37 a5 d7 a3 61 ec 4a d4 b9 08 22 f4 7a 41 bb 0e 7b d4 23 87 1c 79 0d da 49 ad 00 d5 7b

  N_C = 10225549 75542090 06348271 07216004 43949132 21464227 56777417 79805505 60210405 30801119 34766724 88241332 96162673 40510879 99662634 34683008 13972622 46730728 06048480 65124499 99058632 11426335 06094766 73917049 52437775 20161990 77765243 75057209 80003147 74323255 22392699 09862465 38360263 91545000 58856398 42950464 91454041 49732281 30477

- Problema 4 - Uma entidade gerou dois pares de chaves (e_A, d_A), (e_B, d_B) com o mesmo módulo N. O criptograma C resultou de cifrar uma mensagem M com a chave pública (N, e_A). O par (e_B, d_B) é conhecido.

  $e_A = 7$

  $e_B = 65537$

  d_B = 39416603 08803989 58879640 60371334 74870767 01818692 71319109 96047734 77021058 80619234 45842093 22989265 40265854 46503283 59018770 79191136 36255960 47742324 91527176 49943178 54466603 97131847 39697924 23131558 09991190 99507819 57298308 49959370 43975358 56487483 94681408 32738389 23526495 28455574 85732600 34058188 18014111 47624918 2273

  N = 93190689 63134454 02889430 23973887 64163256 06316438 10766252 14573607 27400040 80019580 40048819 04788870 37135761 33026179 53283303 80192983 54231849 99310560 89365677 01226900 39118217 68807536 89742682 75502654 02624038 51351000 47956910

89654752 46903102 66577893 56395540 17423017 47706490 63948079 50906983 04800723 23583374 60149553 0559

C = 80 f8 31 9a 3c 3c 59 e3 a3 50 58 15 18 72 d2 d6 9a ce 3e d9 8e a0 69 55 55 dc 7e 65 63 ce 3d f1 a2 c2 d1 79 dd 71 e0 a8 dc b9 8d 85 61 63 fb c6 fb 86 e0 a2 4d 53 6b c5 29 8d c8 a6 82 12 4c a2 a2 7c 0f db ee 1a cb 0b 82 5e 66 ec dc 8d e1 ac f9 d0 5b 9f 6a 35 85 f7 99 1a 63 39 9c 10 2d c7 72 1b 22 ca 55 e4 a9 51 77 04 4f dd bd b0 a9 9c 59 8b a6 77 21 d8 e4 01 ef 87 60 45 48 1a 4b f1

- Problema 5

e = 27812569 04645424 48243208 16342425 55131496 59501989 63961226 61652483 18528187 07612280 20036949 45711829 20614986 90774019 71597106 39233959 93626211 30977624 64847197 15841091 05417809 72991512 03805829 76281221 65933139 34362689 14954738 09004783 72287256 44919197 27681115 48274700 14626211 44735141 53992565 67620245 82802558 93912036 9493

N = 13906284 52322712 24121604 08171212 77565748 29750994 81980613 30826241 59264093 53806140 10018474 72855914 60307493 45387009 85798553 19616979 96813105 65488812 32423598 58157311 86882762 04589907 38774012 44364116 65411744 61115382 05094395 81940703 24865198 44416577 73938538 85199987 26871321 61290315 04832006 86546285 15575160 75965463 40899

C = 77 81 2f 94 ed 89 d2 3a 00 fd 62 99 c8 2a e0 61 b1 bb f7 9a b8 80 24 9e 9f c2 22 a4 dc 7d 34 32 69 f0 cd 2f 71 87 d1 13 64 7b c3 3d b5 30 bb d0 64 99 fc c9 67 40 c6 eb 1b 5f df aa 1b c5 0f 24 50 b4 de 42 54 0f 52 fb ab 05 c3 89 4a 84 c9 8c ae c1 c2 42 21 03 93 63 e6 3b da 64 2b 99 38 d2 b4 06 1b 67 59 cd 66 b9 b9 14 8b ea 93 15 57 47 a2 b9 db 45 08 2a 07 9f 4b b2 34 31 b9 03 99 8f

3 Resolução do Trabalho

A resolução deste trabalho está dividida em duas partes: numa primeira parte são implementadas as funções necessárias à utilização desta técnica criptográfica, isto é, que permitem cifrar e decifrar mensagens; na segunda parte são abordadas algumas falhas que se podem cometer e que permitem recuperar com alguma facilidade a mensagem cifrada.

3.1 Implementação das Primitivas e Esquemas do RSA

Foi assumido que as mensagens originais eram representadas por *strings* de caracteres. Devido às limitações do *PARI/GP* (na representação do carácter 0) o resultado da cifragem das mensagem será uma *string* de octetos (inteiros entre 0 e 255).

Para a concretização do trabalho, foram implementadas as seguintes funções:

- `prandom(n)` - gera um número primo aleatório de n bits;
- `rsakeys(t)` - gera uma par *chave pública/chave privada* (permite gerar os dois tipos de chave privada, em função de t, que por defeito tem o valor 1, sendo neste caso a chave privada constituída pelo N e pelo d);
- `str2os(str)` - converte a *string* de caracteres *str* para uma *string* de octetos;
- `os2str(s)` - converte a *string* de octetos s para uma *string* de caracteres;
- `i2osp(n,len)` - converte o inteiro n para uma *string* de octetos de comprimento *len*;

- `os2ip(s)` - converte a *string* de octetos s para um inteiro;
- `rsaep(pk,m)` - cifra a mensagem m, usando a chave pk;
- `rsadp(sk,c)` - recupera a mensagem original a partir do criptograma c e da chave sk;
- `rsaes_encryption(pk,m)` - cifra o texto m, usando a chave pk;
- `rsaes_decryption(sk,c)` - recupera o texto original a partir do criptograma c e da chave sk;
- `rsaes_pkcs_encryption(pk,m)` - cifra o texto m, usando a chave pk (é utilizada a codificação *PKCS1.5*);
- `rsaes_pkcs_decryption(sk,c)` - recupera o texto original a partir do criptograma c e da chave sk (é utilizada a codificação *PKCS1.5*).

No anexo A.1 é disponibilizado o código fontes das várias funções implementadas.

3.2 Ataques à Cifra RSA

Neste secção será dada uma breve explicação das falhas que cada problema possui e que permitem recuperar as mensagem. No anexo A.2 é disponibilizado o código para *PARI/GP* que foi usado para auxiliar esta tarefa.

3.2.1 Problema 1

No primeiro problema verifica-se que o valor de N não é muito elevado, como tal, será possível calcular o valor de $\varphi(N)$. A partir daqui podemos determinar o d, fazendo $d = (e \ mod \ N)^{-1}$. Tal permite recuperar o texto original usando a função `rsaes_decryption` (*Tres tristes tigres*).

Apesar de ter sido possível determinar o valor de $\varphi(N)$, sublinha-se que a operação demorou mais de 1 hora!

3.2.2 Problema 2

O problema 2 foi resolvido determinando a raiz cúbica do inteiro correspondente à *string* de octetos dada. Depois de convertida, e recorrendo ao *Mathematica*, verificou-se que a raiz cúbica era um inteiro, logo foi só converter esse inteiro para uma *string* de octetos e posteriormente para uma *string* de caracteres e obteve-se a mensagem original (*Em Abril aguas de mil*).

3.2.3 Problema 3

Neste problema é usado $e = 3$ para 3 valores de N diferentes. Ou seja, se m é a mensagem original, os criptogramas correspondentes serão $c_i = m^3 \ mod \ N_i$, para $i \in \{1, 2, 3\}$. Temos então o seguinte sistema de equações:

$$\begin{cases} x & \equiv & c_1 \ mod \ N_1 \\ x & \equiv & c_2 \ mod \ N_2 \\ x & \equiv & c_3 \ mod \ N_3 \end{cases}$$

Recorrendo ao teorema chinês dos restos, pode-se determinar o valor de x. Sendo que $x = m^3$, temos então $m = \sqrt[3]{x}$, o que é fácil de determinar visto que $x < N_1 N_2 N_3$ (recorreu-se ao *Mathematica* para efectuar este cálculo). Depois é só converter o inteiro para uma *string* de octetos e por fim para uma uma *string* de caracteres (*Agua mole em pedra dura tanto da ate que fura*). Este ataque é denominado em [3] como *Small encryption exponent e*.

3.2.4 Problema 4

No problema 4 é conhecido um par de chaves para o valor de N em causa. Isto permite a aplicação do ataque *Common Modulus* descrito em [1].

Seja $k = d\,e - 1$, que facilmente se conclui ser múltiplo de $\varphi(N)$. Verifica-se que $g^k = 1\ mod\ N$, logo $g^k - 1 = 0\ mod\ N$, que é equivalente a $(g^{k/2} - 1)(g^{k/2} + 1) = 0\ mod\ N$, sendo que $g^{k/2}$ é uma raiz quadrada de 1 em $\mathbb{Z}_n$. Pelo teorema chinês dos restos sabe-se que existem quatro raízes, ± 1 e $\pm x$. Pode-se concluir que $x - 1$ divide um dos factores de N (e que $x + 1$ divide o outro), assim basta calcular o máximo divisor comum entre $x - 1$ e N, obtendo-se então um dos factores de N. Agora só falta calcular x. Seja $k = 2^t\ r$, com r ímpar, sabe-se que, para um valor aleatório de $g \in \mathbb{Z}_N$, existe uma probabilidade de $1/2$ de que um dos valores do conjunto $\{g^{k/2^i} \mid 1 \leq i \leq t\}$, seja o valor x, logo basta usar vários valores de g para o determinar. Desta forma chega-se à factorização de N, sendo então possível calcular a chave privada.

Visto que foi possível factorizar o N com sucesso, devia agora ser possível obter a mensagem original. Para tal foi usada a função `rsaes_decryption`, no entanto a mensagem obtida não parece estar correcta.

Pode-se, contudo, facilmente verificar que os valores obtidos são os correctos, pois decifrando e voltando a cifrar a *string* de octetos obtém-se o valor original, ou seja, é possível recuperar a *string* de octetos original.

3.2.5 Problema 5

No problema 5 verifica-se que valor de e e N têm ordens de grandeza semelhantes, logo será de esperar que o valor de d seja pequeno. Tal permite usar um ataque identificado em [1] como *Low Private Exponent*.

Sabe-se que $e\,d \equiv 1\ mod\ \varphi(N)$, logo existe um inteiro k tal que $e\,d\ -\ k\,\varphi(N) = 1$. Como $\varphi(N) \approx N$, então $k/d \approx e/N$. Fazendo a expansão de e/N como fracções contínuas, um dos convergentes será k/d. Calculam-se então estes valores até que $2^{e\,d} \equiv 2\ mod\ N$. Quando isto acontecer ter-se-á encontrado o valor de d e será então possível determinar a mensagem usando a função `rsaes_decryption` (*Parabens! Conseguiu!*).

4 Conclusão

Com a realização deste trabalho testou-se a implementação da cifra RSA, dois dos tipos de chaves possíveis e ainda a utilização de *padding*. Ainda no que diz respeito à implementação da cifra, através de alguns testes efectuados (anexo B) verificou-se que não existia uma diferença significativa entre a utilização dos dois tipos de chave (ao contrário do que se estava à espera).

Foi também possível ver algumas situações em que as mensagens cifradas eram facilmente recuperadas, alertando para alguns cuidados a ter na utilização desta técnica criptográfica.

Por falta de tempo, não foram analisadas possíveis optimizações aos algoritmos, sendo esse um aspecto a analisar no futuro.

Referências

[1] Boneh, D. *Twenty Years of Attacks on the RSA Cryptosystem*

[2] Dujella, A. *Continued Fractions And RSA With Small Secret Exponent.*

[3] Menezes, A., Oorschot, P. e Vanstone, S. *Handbook of Applied Cryptography.* CRC Press, 2001.

[4] *PKCS #1 v2.1: RSA Cryptography Standard*. RSA Laboratories, 2002.

[5] *IEEE Std 1363-2000, IEEE Standard Specifications for Public-Key Cryptography*. IEEE, 2000.

[6] *IEEE Std 1363a-2004, IEEE Standard Specifications for Public-Key Cryptography - Amendment 1: Additional Techniques*. IEEE, 2004.

[7] `http://en.wikipedia.org`

[8] `http://mathworld.wolfram.com`

[9] `http://pt.wikipedia.org`

Anexos

A Código

A.1 Implementação da Cifra RSA

```
/**
 * Rui Carlos A. Goncalves
 * LMCC - FC
 * 2006/07
 * TP 1
 */

\\=============================================================================

/**
 * Generate a random prime.
 *
 * @param n number of bits.
 *
 * @return a prime number with n bits.
 */
prandom(n)=
{
  local(p);

  while(1,
    p=2^(n-1)+random(2^(n-1));
    if(ispseudoprime(p),if(isprime(p),return(p)))
  )
}

/**
 * Generate a pair private/public key.
 *
 * @param t type of private key (default=1):
 *          1 (n,d)
 *          2 (p,q,d1,d2,q)
 *
 * @return a pair public/private key.
 */
rsakeys(t=1)=
{
  local(p,q,fi,e,d);

  p=prandom(512);
  q=prandom(512);
  phi=(p-1)*(q-1);

  e=1+random(phi-1);
  while(gcd(e,phi)!=1,e=1+random(phi-1));
```

```
  d=lift((Mod(e,phi))^-1);

  if(t=1,return([[p*q,e],[p*q,d]]));
  if(t=2,return([[p*q,e],[p,q,d%(p-1),d%(q-1),(q^-1)%p]]));
  error("invalid private key type")
}

\\=============================================================================

/**
 * Convert a char string to an octet string.
 *
 * @param str string to be converted.
 *
 * @return corresponding octet string.
 */
str2os(str)=
{
  return(Vecsmall(str))
}

/**
 * Convert an octet string to a char string.
 *
 * @param s octet string to be converted.
 *
 * @return corresponding char string.
 */
os2str(s)=
{
  local(i,j,new,len);

  len=length(s);
  i=1;
  while(i<=len&&s[i]==0,i++);

  new=vector(len-i+1);

  for(j=1,len-i+1,
    new[j]=s[j+i-1]
  );

  return(Strchr(new))
}

/**
 * Integer-to-Octet-String primitive.
 * Converts a nonnegative integer to an octet string of a specified length.
 *
 * @param n   nonnegative integer to be converted.
 * @param len intended length of the resulting octet string.
 *
 * @return corresponding octet string of length len.
```

```
 */
i2osp(n,len)=
{
  local(vec,i);

  if(n>=256^len,error("integer to large"));

  vec=vector(len,x,0);
  i=len;
  while(n>0,
    vec[i]=n%256;
    n=n\256;
    i--
  );

  return(vec)
}

/**
 * Octet-String-to-Integer primitive.
 * Converts an octet string to a nonnegative integer.
 *
 * @param s octet string to be converted.
 *
 * @return corresponding nonnegative integer.
 */
os2ip(s)=
{
  local(res,i,len);

  len=length(s);
  res=0;

  for(i=1,len,
    res+=s[i]*256^(len-i)
  );

  return(res)
}

\\============================================================================

/**
 * RSA-Encryption-Primitive.
 * Produce a ciphertext representative from a message representative.
 *
 * @param pk RSA public key.
 * @param m  message representative.
 *
 * @return ciphertext representative.
 */
rsaep(pk,m)=
{
```

```
  if(m<0||m>=(pk[1]),error("message representative out of range"));
  return(lift(Mod(m,pk[1])^pk[2]))
}

/**
 * RSA-Decryption-Primitive.
 * Recovers the message representative from the ciphertext representative.
 *
 * @param sk RSA private key.
 * @param c  ciphertext representative.
 *
 * @return message representative.
 */
rsadp(sk,c)=
{
  local(m1,m2,h);

  if(length(sk)==2,
    if(c<0||c>=sk[1],error("ciphertext representative out of range"));
    return(lift(Mod(c,sk[1])^sk[2]))
  );

  if(length(sk)==5,
    if(c<0||c>=(sk[1]*sk[2]),error("ciphertext representative out of range"));
    m1=lift(Mod(c,p)^sk[3]);
    m2=lift(Mod(c,q)^sk[4]);
    h=lift(Mod(m1-m2,p)*Mod(sk[5],p));
    return(m2+q*h)
  );

  error("invalid private key")
}

\\=============================================================================

/**
 * RSA Encryption Scheme (Encryption Operation).
 * Produce a ciphertext from a message.
 *
 * @param pk RSA public key.
 * @param m  message to be encrypted (a char string).
 *
 * @return ciphertext (an octet string).
 */
rsaes_encryption(pk,m)=
{
  local(l,len,os,g,c,r);

  os=str2os(m);

  l=ceil(log(pk[1]+1)/log(2))-1;
  len=length(os);
  if(len*8>l,error("error"));
```

```
  g=os2ip(os);
  c=rsaep(pk,g);
  r=i2osp(c,ceil(l/8));

  return(r)
}

/**
 * RSA Encryption Scheme (Decryption Operation).
 * Recovers the original message from the ciphertext.
 *
 * @param sk RSA private key.
 * @param c  ciphertext (an octet string).
 *
 * @return original message (a char string).
 */
rsaes_decryption(sk,c)=
{
  local(l,len);

  g=os2ip(c);
  m=rsadp(sk,g);
  s=i2osp(m,ceil((log(sk[1]+1)/log(2)-1)/8));

  return(os2str(s))
}

\\=============================================================================

/**
 * RSA Encryption Scheme PKCS (Encryption Operation).
 * Produce a ciphertext from a message.
 *
 * @param pk RSA public key.
 * @param m  message to be encrypted (a char string).
 *
 * @return ciphertext (an octet string).
 */
rsaes_pkcs_encryption(pk,m)=
{
  local(len,l,k,os,em,g,c,r,i,j);

  os=str2os(m);

  l=ceil(log(pk[1]+1)/log(2)-1);
  len=length(os);
  if((len+11)*8>l,error("message too long"));

  k=ceil(l/8);

  em=vector(k);
  em[2]=2;
```

```
  for(i=3,k-len-1,em[i]=1+random(254));
  i=k-len;
  for(j=1,len,em[j+i]=os[j]);

  g=os2ip(em);
  c=rsaep(pk,g);
  r=i2osp(c,k);

  return(r)
}

/**
 * RSA Encryption Scheme PKCS (Decryption Operation).
 * Recover the originl message from the ciphertext.
 *
 * @param sk RSA private key.
 * @param c  ciphertext (an octet string).
 *
 * @return original message (a char string).
 */
rsaes_pkcs_decryption(sk,c)=
{
  local(g,l,k,m,em,i);

  l=ceil(log(sk[1]+1)/log(2)-1);
  k=ceil(l/8);

  if(length(c)!=k||k<11,error("decryption error"));

  g=os2ip(c);
  m=rsadp(sk,g);
  em=i2osp(m,k);

  if(em[1]!=0,error("decryption error"));
  if(em[2]!=2,error("decryption error"));

  i=3;
  em[2]=0;
  while(i<=k&&em[i]!=0,em[i]=0;i++);
  if(i>k||i<11,error("decryption error"));

  return(os2str(em))
}
```

A.2 Ataques à Cifra RSA

```
\r rsa.gp

prob1()=
{
  local(e,N,C,fi,d);
```

```
  e=5;
  N=35875741 92954427 23106714 14455212 43178607 58771428 65793354 19222874
      94392030 35201;
  C=[20, 231, 71, 64, 48, 233, 214, 129, 99, 30, 156, 39, 132, 58, 97, 58, 58,
      50, 74, 72, 231, 226, 219, 231, 84, 76, 180, 223, 151, 81, 198, 141];

  fi=eulerphi(N);
  d=lift(Mod(e,fi)^-1);

  return(rsaes_decryption([N,d],C))
}

prob2()=
{
  local(e,N,C,c,m,os,str);

  e=3;
  N=10890326 83577364 55066815 31468096 24530428 14775006 20649272 16784759
      69098339 19686793 38591464 94098327 68160832 68595581 07942644 64577272
      15780237 86494435 78621853 51011217 49241579 47418994 68334776 45334025
      67120030 90427026 72365522 90956722 14149022 58969417 44183688 41198294
      49701459 19402707 57209267 66820866 41769478 99836600 54201;
  C=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 5, 27, 43, 39, 148, 83, 155, 26, 125, 28, 81, 217, 49, 28, 88, 209,
      112, 204, 204, 12, 75, 202, 40, 143, 39, 209, 164, 189, 145, 210, 178,
      74, 12, 84, 16, 188, 63, 238, 45, 71, 169, 164, 213, 91, 253, 99, 71,
      141, 15, 6, 13, 151, 104, 192];

  c=os2ip(C);
  \\m=c^(1/3) << Mathematica
  m=60478813 82077741 01153370 77864365 52147429 740;
  os=i2osp(m,128);
  str=os2str(os);

  return(str)
}

prob3()=
{
  local(e,N1,N2,N3,C1,C2,C3,c1,c2,c3,m3,n,os,str);

  e=3;
  N1=11221088 19125288 32873488 73089762 61927054 73369219 61653875 30817045
      51444228 28779927 05495181 83502360 74434502 33408237 31796310 20451837
      15448390 19675852 32833164 49803848 51027727 54243488 99295466 15023950
      78707201 21300501 58370679 94235368 65480798 06643019 02331811 61961115
      24880623 90885355 71687505 13262508 51418329 55675633 96901;
  N2=12039209 84865608 81218868 53033797 59281708 28473709 78857568 70024244
      22185252 09633774 37040943 51935139 81140455 56006396 09802793 13799625
      78333140 68179655 03969227 04908239 80274420 62646596 96670054 17243382
```

```
      18600061 89185562 58697621 08251423 93988849 70118558 43779219 96904077
      36712811 68660000 16593840 83176017 01481588 11123231 15387;
  N3=10225549 75542090 06348271 07216004 43949132 21464227 56777417 79805505
      60210405 30801119 34766724 88241332 96162673 40510879 99662634 34683008
      13972622 46730728 06048480 65124499 99058632 11426335 06094766 73917049
      52437775 20161990 77765243 75057209 80003147 74323255 22392699 09862465
      38360263 91545000 58856398 42950464 91454041 49732281 30477;

  C1=[93, 240, 186, 55, 12, 237, 32, 123, 239, 156, 128, 242, 250, 227, 183,
      152, 145, 68, 237, 63, 52, 95, 168, 223, 96, 146, 187, 161, 192, 91, 44,
      238, 75, 219, 184, 138, 145, 227, 242, 78, 119, 31, 90, 154, 224, 18,
      104, 225, 61, 213, 138, 138, 41, 190, 89, 252, 198, 225, 181, 237, 237,
      80, 198, 26, 40, 117, 118, 89, 197, 217, 12, 19, 5, 142, 242, 119, 9, 91,
       4, 86, 144, 46, 188, 213, 188, 64, 132, 208, 17, 110, 50, 209, 123, 128,
       120, 223, 202, 157, 92, 83, 38, 236, 137, 119, 143, 208, 37, 198, 11,
      164, 78, 254, 182, 25, 16, 21, 23, 103, 88, 250, 88, 93, 58, 41, 187,
      173, 128, 125];
  C2=[139, 186, 53, 160, 226, 57, 75, 118, 180, 54, 86, 188, 89, 199, 107, 66,
      69, 163, 226, 134, 85, 167, 43, 187, 108, 47, 3, 98, 77, 34, 218, 174,
      220, 252, 7, 7, 130, 200, 0, 7, 188, 164, 236, 166, 118, 157, 207, 198,
      44, 120, 29, 28, 81, 17, 138, 234, 233, 17, 107, 18, 28, 85, 51, 114, 40,
       202, 73, 238, 28, 140, 215, 242, 51, 87, 233, 114, 56, 145, 17, 176,
      185, 53, 107, 64, 206, 179, 95, 250, 16, 168, 41, 26, 94, 58, 169, 238,
      244, 87, 175, 55, 165, 215, 163, 97, 236, 74, 212, 185, 8, 34, 244, 122,
      65, 187, 14, 123, 212, 35, 135, 28, 121, 13, 218, 73, 173, 0, 213, 123];
  C3=[33, 163, 61, 191, 229, 69, 40, 162, 166, 221, 44, 205, 190, 61, 253, 60,
      102, 41, 28, 245, 30, 9, 109, 230, 13, 253, 159, 36, 71, 57, 247, 72, 20,
       224, 77, 182, 11, 246, 9, 81, 201, 215, 243, 216, 229, 4, 172, 88, 114,
      211, 99, 135, 60, 120, 241, 118, 185, 149, 105, 99, 62, 105, 165, 68,
      192, 167, 94, 247, 26, 92, 254, 200, 129, 16, 25, 4, 116, 213, 59, 244,
      151, 15, 158, 160, 35, 232, 27, 212, 109, 77, 58, 135, 76, 214, 94, 11,
      65, 223, 254, 205, 3, 109, 251, 243, 223, 79, 8, 254, 126, 233, 239, 110,
       167, 87, 59, 115, 171, 203, 210, 71, 33, 239, 194, 23, 228, 80, 44,
      119];

  c1=os2ip(C1);
  c2=os2ip(C2);
  c3=os2ip(C3);

  m3=matsolvemod([1;1;1],[N1,N2,N3],[c1,c2,c3]~);
  m3=m3[1];
  \\n=m3^(1/3) << Mathematica
  n=60001717 07360495 45494927 67988485 05133710 13869467 61957568 98401173
      76077778 73877284 43936393 61049340 50108260 0033;
  os=i2osp(n,128);
  str=os2str(os);

  return(str);
}

/**
 * n-1=2^t*s
 */
```

```
fact2(n)=
{
  local(i);

  if(n==1,error("valor invalido"));

  n--;
  i=0;
  while(n%2==0,
    n=n/2;
    i++
  );

  return([n,i])
}

prob4()=
{
  local(e1,d1,e2,d2,N,p,q,C,k,g,t,r,aux,x);

  e1=7;
  e2=65537;
  d2=39416603 08803989 58879640 60371334 74870767 01818692 71319109 96047734
       77021058 80619234 45842093 22989265 40265854 46503283 59018770 79191136
       36255960 47742324 91527176 49943178 54466603 97131847 39697924 23131558
       09991190 99507819 57298308 49959370 43975358 56487483 94681408 32738389
       23526495 28455574 85732600 34058188 18014111 47624918 2273;
  N=93190689 63134454 02889430 23973887 64163256 06316438 10766252 14573607
       27400040 80019580 40048819 04788870 37135761 33026179 53283303 80192983
       54231849 99310560 89365677 01226900 39118217 68807536 89742682 75502654
       02624038 51351000 47956910 89654752 46903102 66577893 56395540 17423017
       47706490 63948079 50906983 04800723 23583374 60149553 0559;
  C=[128, 248, 49, 154, 60, 60, 89, 227, 163, 80, 88, 21, 24, 114, 210, 214,
       154, 206, 62, 217, 142, 160, 105, 85, 85, 220, 126, 101, 99, 206, 61,
       241, 162, 194, 209, 121, 221, 113, 224, 168, 220, 185, 141, 133, 97, 99,
       251, 198, 251, 134, 224, 162, 77, 83, 107, 197, 41, 141, 200, 166, 130,
       18, 76, 162, 162, 124, 15, 219, 238, 26, 203, 11, 130, 94, 102, 236, 220,
        141, 225, 172, 249, 208, 91, 159, 106, 53, 133, 247, 153, 26, 99, 57,
       156, 16, 45, 199, 114, 27, 34, 202, 85, 228, 169, 81, 119, 4, 79, 221,
       189, 176, 169, 156, 89, 139, 166, 119, 33, 216, 228, 1, 239, 135, 96, 69,
        72, 26, 75, 241];

  k=d2*e2-1;
  aux=fact2(k+1);
  t=aux[2];
  r=aux[1];

  g=Mod(1+random(2^512),N);
  x=g^k/2;
  i=2;
  while(x==Mod(1,N)||(i<=t&&(x^2)!=Mod(1,N)),
    x=g^(k/2^i);
    i++
```

```
  );

  p=gcd(lift(x-1),N);
  q=N/p;

  d1=lift(Mod(e1,(p-1)*(q-1))^-1);

  return(rsaes_decryption([N,d1],C))
}


prob5()=
{
  local(e,N,C,a0,b0,b1,a1,d1,d2,d3,i);

  e=27812569 04645424 48243208 16342425 55131496 59501989 63961226 61652483
      18528187 07612280 20036949 45711829 20614986 90774019 71597106 39233959
      93626211 30977624 64847197 15841091 05417809 72991512 03805829 76281221
      65933139 34362689 14954738 09004783 72287256 44919197 27681115 48274700
      14626211 44735141 53992565 67620245 82802558 93912036 9493;
  N=13906284 52322712 24121604 08171212 77565748 29750994 81980613 30826241
      59264093 53806140 10018474 72855914 60307493 45387009 85798553 19616979
      96813105 65488812 32423598 58157311 86882762 04589907 38774012 44364116
      65411744 61115382 05094395 81940703 24865198 44416577 73938538 85199987
      26871321 61290315 04832006 86546285 15575160 75965463 40899;
  C=[119, 129, 47, 148, 237, 137, 210, 58, 0, 253, 98, 153, 200, 42, 224, 97,
      177, 187, 247, 154, 184, 128, 36, 158, 159, 194, 34, 164, 220, 125, 52,
      50, 105, 240, 205, 47, 113, 135, 209, 19, 100, 123, 195, 61, 181, 48,
      187, 208, 100, 153, 252, 201, 103, 64, 198, 235, 27, 95, 223, 170, 27,
      197, 15, 36, 80, 180, 222, 66, 84, 15, 82, 251, 171, 5, 195, 137, 74,
      132, 201, 140, 174, 193, 194, 66, 33, 3, 147, 99, 230, 59, 218, 100, 43,
      153, 56, 210, 180, 6, 27, 103, 89, 205, 102, 185, 185, 20, 139, 234, 147,
       21, 87, 71, 162, 185, 219, 69, 8, 42, 7, 159, 75, 178, 52, 49, 185, 3,
      153, 143];

  b1=e/N;
  a1=floor(b1);

  d1=1;
  d2=0;
  d3=a1*d2+d1;

  i=0;
  while(Mod(2,N)^(e*d3)!=Mod(2,N),
    i++;
    a0=a1;
    b0=b1;
    b1=1/(b0-a0);
    a1=floor(b1);

    d1=d2;
    d2=d3;
    d3=a1*d2+d1;
```

```
  );

  return(rsaes_decryption([N,d3],C))
}
```

B Testes Efectuados

```
? k=rsakeys(2);
? rsaes_encryption(k[1],"1234567890 1234567890 1234567890 1234567890 1234567890
    1234567890 1234567890 1234567890 1234567890 1234567890")
%16 = [1, 251, 217, 149, 184, 179, 215, 154, 175, 229, 144, 54, 27, 115, 247,
    180, 57, 106, 203, 74, 141, 64, 92, 13, 112, 16, 23, 18, 51, 96, 157, 214,
    68, 109, 68, 163, 176, 68, 14, 82, 0, 69, 112, 147, 219, 43, 145, 120, 5,
    89, 181, 187, 35, 47, 123, 62, 74, 17, 212, 137, 46, 62, 107, 242, 48, 66,
    242, 60, 113, 195, 232, 131, 45, 23, 87, 58, 198, 217, 115, 58, 191, 141,
    165, 53, 207, 135, 61, 198, 155, 86, 3, 20, 165, 185, 126, 127, 19, 30,
    243, 9, 138, 145, 107, 5, 236, 99, 143, 110, 16, 54, 57, 119, 34, 27, 223,
    118, 55, 79, 43, 121, 198, 24, 27, 237, 242, 255, 222, 193]
? ##  ***   last result computed in 29 ms.
? rsaes_decryption(k[2],%16)
%17 = "1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
    1234567890 1234567890 1234567890 1234567890"
? ##
  ***   last result computed in 57 ms.
? for(i=1,100,rsaes_decryption(k[2],%16))
? ##
  ***   last result computed in 2,949 ms.
? k=rsakeys(1);
? rsaes_encryption(k[1],"1234567890 1234567890 1234567890 1234567890 1234567890
    1234567890 1234567890 1234567890 1234567890 1234567890")
%21 = [49, 121, 27, 62, 79, 228, 166, 143, 227, 232, 66, 203, 27, 221, 173, 8,
    191, 251, 20, 145, 142, 249, 204, 250, 55, 51, 24, 191, 127, 186, 17, 148,
    232, 23, 48, 155, 48, 34, 246, 203, 249, 15, 145, 120, 63, 113, 159, 121,
    169, 65, 52, 16, 190, 209, 194, 67, 180, 58, 69, 187, 18, 164, 1, 247, 218,
     124, 27, 17, 72, 217, 33, 59, 204, 147, 146, 17, 46, 244, 78, 84, 176,
    179, 157, 15, 2, 165, 13, 40, 8, 172, 239, 128, 126, 71, 31, 52, 149, 75,
    198, 130, 193, 220, 79, 164, 234, 168, 201, 3, 78, 228, 110, 16, 38, 49,
    85, 119, 74, 174, 167, 29, 96, 226, 21, 57, 152, 59, 75, 191]
? ##
***   last result computed in 30 ms.
? for(i=1,100,rsaes_decryption(k[2],%21))
? ##
  ***   last result computed in 3,040 ms.
? k=rsakeys();
? rsaes_encryption(k[1],"uma mensagem pequena...")
%2 = [82, 132, 190, 47, 149, 72, 141, 235, 52, 29, 201, 132, 206, 105, 104, 11,
    14, 164, 101, 54, 225, 37, 146, 100, 2, 26, 70, 27, 25, 156, 5, 110, 44,
    147, 207, 243, 140, 75, 14, 35, 83, 127, 105, 32, 31, 165, 186, 241, 121,
    201, 56, 240, 7, 16, 32, 123, 166, 146, 109, 10, 127, 235, 217, 25, 128,
    255, 150, 226, 27, 211, 87, 177, 233, 246, 216, 55, 88, 124, 227, 9, 218,
    80, 61, 238, 80, 233, 202, 183, 121, 51, 177, 203, 137, 103, 194, 219, 171,
```

```
     44, 56, 230, 71, 238, 194, 39, 221, 138, 111, 91, 249, 115, 106, 141, 35,
    14, 10, 26, 67, 56, 250, 155, 179, 53, 100, 155, 135, 222, 78, 79]
? rsaes_decryption(k[2],%2)
%3 = "uma mensagem pequena..."
? rsaes_encryption(k[1],"um mensagem um pouco maior do que a anterior...")
%4 = [74, 21, 126, 176, 53, 129, 108, 219, 3, 152, 141, 57, 121, 72, 238, 174,
    46, 173, 1, 66, 226, 225, 44, 23, 115, 222, 238, 177, 220, 254, 239, 25,
    48, 147, 149, 90, 108, 223, 141, 192, 175, 130, 224, 81, 10, 235, 169, 185,
     54, 46, 170, 40, 27, 41, 6, 127, 109, 188, 248, 73, 6, 232, 129, 43, 232,
    249, 206, 155, 189, 55, 111, 85, 31, 73, 95, 153, 82, 102, 142, 19, 92, 79,
     14, 60, 253, 251, 0, 18, 2, 65, 127, 243, 186, 142, 84, 154, 140, 140,
    130, 232, 237, 34, 57, 78, 1, 180, 26, 39, 231, 48, 184, 183, 235, 58, 128,
     101, 47, 197, 46, 101, 177, 2, 120, 65, 47, 157, 248, 17]
? rsaes_decryption(k[2],%4)
%5 = "um mensagem um pouco maior do que a anterior..."
? rsaes_pkcs_encryption(k[1],"uma mensagem pequena com pkcs...")
%6 = [58, 177, 204, 188, 191, 117, 234, 12, 98, 232, 240, 50, 229, 39, 34, 63,
    10, 219, 228, 66, 228, 76, 100, 110, 152, 70, 32, 45, 57, 154, 52, 175,
    213, 15, 57, 13, 173, 236, 213, 32, 222, 68, 180, 204, 94, 49, 212, 144,
    82, 29, 194, 207, 0, 171, 86, 106, 164, 60, 31, 195, 225, 241, 134, 211,
    139, 209, 80, 53, 111, 183, 100, 133, 116, 238, 156, 104, 15, 237, 17, 152,
     27, 87, 63, 148, 245, 155, 252, 68, 239, 67, 26, 23, 207, 81, 31, 0, 192,
    146, 3, 173, 2, 198, 193, 192, 217, 153, 126, 145, 62, 222, 231, 33, 9,
    211, 56, 242, 109, 4, 103, 86, 209, 234, 32, 226, 165, 19, 149, 12]
? rsaes_pkcs_decryption(k[2],%6)
%7 = "uma mensagem pequena com pkcs..."
? rsaes_pkcs_encryption(k[1],"um mensagem um pouco maior do que a anterior com
    pkcs...")
%8 = [38, 213, 240, 54, 207, 67, 119, 189, 70, 115, 128, 30, 27, 84, 154, 246,
    96, 223, 65, 129, 247, 53, 53, 34, 104, 114, 88, 112, 126, 68, 156, 116,
    74, 79, 193, 182, 2, 14, 133, 209, 162, 14, 59, 105, 161, 82, 72, 217, 165,
     126, 206, 70, 11, 5, 114, 139, 64, 164, 88, 157, 149, 153, 203, 111, 230,
    12, 205, 171, 199, 63, 166, 13, 144, 240, 220, 79, 61, 92, 88, 174, 12, 62,
     60, 100, 224, 212, 112, 238, 1, 121, 72, 14, 254, 0, 219, 205, 244, 108,
    108, 253, 244, 115, 153, 127, 255, 9, 83, 41, 208, 236, 69, 68, 40, 31, 94,
     54, 98, 58, 204, 138, 106, 28, 119, 10, 38, 100, 251, 0]
? rsaes_pkcs_decryption(k[2],%8)
%9 = "um mensagem um pouco maior do que a anterior com pkcs..."
? rsaes_pkcs_encryption(k[1],"um mensagem maior do que o que devia ser......
    .... .. .  . . . . . .. .  .. .  . .    ...... .... ....... ........ .. . .
     .. ..")
  ###   user error: message too long
```