Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Filière Télécommunication, Internet et Communication

2019-2020

Labo 01 - RSA

Sécurité IT 2

Auteurs

Loïc Guibert
Raphaël Pittet
Célestin Rumo

Professeur

Jean-Roland Schuler

Fribourg, Le 19 février 2020

Liste des fichiers sources

Listing 1 – Python code

```
#! /usr/bin/python2



## /usr/bin/python2 ./1_rsa_compute_key_3_PEM.py



#https://bitbucket.org/sybren/python-rsa/src/01487233708ecd4ddb3063fc36dab658280e469a/rsa/key.py?at=default

# extended Euclidean Algorithm: egcd
# e*d%(p-1)(q-1) = 1
# e is fixed and must be relative prime with (p-1)(q-1) -> gcd ((p-1)(q-1), e) = 1
# this equation e*d%(p-1)(q-1) = 1 can be solved with the egcd

# g = gcd(a, b) = ax + by: g must be 1: a = (p-1)(q-1): b = e
# (g, x, y) = egcd(a, b)
# e is fixed, d = g%(p-1)(q-1)

from pyasn1.type import univ, namedtype
from pyasn1.codec.der import encoder
import base64



def save_pkcs1_pem(n, d, e, p, q, exp1, exp2, coef):
    '''Saves the private key in PKCS#1 DER format.
    '''

    class AsnPrivKey(univ.Sequence):
        componentType = namedtype.NamedTypes(
            namedtype.NamedType('version', univ.Integer()),
            namedtype.NamedType('modulus', univ.Integer()),
            namedtype.NamedType('publicExponent', univ.Integer()),
            namedtype.NamedType('privateExponent', univ.Integer()),
            namedtype.NamedType('prime1', univ.Integer()),
            namedtype.NamedType('prime2', univ.Integer()),
            namedtype.NamedType('exponent1', univ.Integer()),
            namedtype.NamedType('exponent2', univ.Integer()),
            namedtype.NamedType('coefficient', univ.Integer()),
        )

    # Create the ASN object
    asn_key = AsnPrivKey()
    asn_key.setComponentByName('version', 0)
    asn_key.setComponentByName('modulus', n)
    asn_key.setComponentByName('publicExponent', e)
    asn_key.setComponentByName('privateExponent', d)
    asn_key.setComponentByName('prime1', p)
    asn_key.setComponentByName('prime2', q)
    asn_key.setComponentByName('exponent1', exp1)
```

```
52      asn_key.setComponentByName('exponent2', exp2)
53      asn_key.setComponentByName('coefficient', coef)
54
55      b64 = base64.encodestring(encoder.encode(asn_key))
56
57      asn_key_pem = "-----BEGIN RSA PRIVATE KEY-----\n%s-----END RSA PRIVATE KEY
            ↪ -----" % b64
58
59      return asn_key_pem
60
61
62  def egcd(a, b):
63      if b == 0:
64          return (a, 1, 0)
65      else:
66          g, x, y = egcd(b, a % b)
67          #print a, b, (a//b), g, x, y
68          return (g, y, x - (a // b) * y)
69
70
71  def crypt(message, e , n, hasPadding):
72      f = None
73      if (hasPadding):
74          f = open("crypted_padding.txt", "wb")
75          message = 0
                ↪ x0002111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
                ↪
76      else :
77          f = open("crypted_nopadding.txt", "wb")
78      c = (message ** e)%n
79      b = bytearray.fromhex(hex(c)[2:-1])
80      f.write(b)
81      f.close()
82
83  def getPrivateKey(p, q, toFile):
84      n=p*q
85      phi = (p-1)*(q-1)
86      e=0x10001
87
88      g,x,y=egcd(phi, e)
89      if (g==1):
90          d=y%phi
91          pem=save_pkcs1_pem(n, d, e, p, q, d%(p-1), d%(q-1), 0)
92
93          #coefficient INTEGER, -- (inverse of q) mod p
94
95          #print hex(n), hex(d), hex(e), hex(p), hex(q), hex(d%(p-1)), hex(d%(q-1))
96
97          print ("p: " + hex(p))
98          print ("q: " + hex(q))
99          print ("e: " + hex(e))
100         print ("d: " + hex(d))
101         print ("n: " + hex(n))
102
103         # print (pem)
104
```

```
105         if (toFile) :
106             f = open("rsaprivatechallenge.pem", "w")
107             f.write(pem)
108             f.close()
109 
110         return e, n
111 
112     else:
113         print ("e and phi are not relatively prime")
114         print ("e=", e, "phi=", phi)
115 
116 
117 #http://b3ck.blogspot.ch/2011/06/how-to-break-rsa-explicitly-with.html
118 #p=25890325045218759213263085202117598708 9
119 #q=27031722695324063496099599033189179262 3
120 #p=0xf06b
121 #q=0xef9f
122 
123 print("-------------------------------------------------")
124 
125 
126 print("Processing RSA keys")
127 
128 p1_keys = '''
129     00:ed:4d:eb:ee:2e:c0:1f:3a:06:c1:99:94:18:23:
130     95:c8:03:2a:cf:fa:ae:ae:43:08:5b:a1:c0:cb:96:
131     79:34:c3
132 '''
133 
134 q1_keys = '''
135     00:dc:88:e3:d8:d9:f8:12:e9:b3:76:d2:4f:3d:94:
136     98:88:75:0f:07:8c:31:12:ce:c5:02:e9:92:92:ae:
137     6b:d6:37
138 '''
139 
140 p_keys=int(p1_keys.translate(None, '\n :'),16)
141 q_keys=int(q1_keys.translate(None, '\n :'),16)
142 
143 e_keys, n_keys = getPrivateKey(p_keys, q_keys, False)
144 
145 print("-------------------------------------------------")
146 
147 print("Processing message without padding")
148 
149 message=0x101112131415161718191a1b1c1d1e1f;
150 crypt(message, e_keys, n_keys, False)
151 
152 print("-------------------------------------------------")
153 
154 
155 print("Processing message with padding")
156 
157 
158 crypt(message, e_keys, n_keys, True)
159 
160 print("-------------------------------------------------")
```

```
161
162 print("Retrieving the private key")
163
164
165 p_challenge=59897668573486112153258715413
166 q_challenge=63828110732706550243826648267
167
168 getPrivateKey(p_challenge, q_challenge, True)
169 print("-----------------------------------------------------")
170
171 print("All done")
```

Listing 2 – rsa.c

```
1  // gcc -Wall -Wextra -g -o rsa rsa.c -lcrypto
2
3  // see: openssl.../demos/maurices/example2.c
4
5  //openssl genrsa -out rsaprivatekey.pem 1024 // Generate private and public keys
6  //openssl rsa -in rsaprivatekey.pem -pubout -out rsapublickey.pem // Get only the
       ↪  public key
7  //openssl rsa -in rsaprivatekey.pem -text // Show the private and public key
8  //openssl rsa -in rsapublickey.pem -pubin -text // Show the public key
9
10
11 #include <stdlib.h>
12 #include <stdio.h>
13 #include <strings.h>
14 #include <stdbool.h>
15
16
17 #include <openssl/evp.h>
18 #include <openssl/rsa.h>
19 #include <openssl/objects.h>
20 #include <openssl/x509.h>
21 #include <openssl/err.h>
22 #include <openssl/pem.h>
23 #include <openssl/ssl.h>
24
25 #include <time.h>
26
27
28 //#define LEN_PADDING 50
29 #define LEN_PADDING_OAEP 41 //50
30 #define LEN_PADDING_PKCS1 11
31 #define LEN_PADDING_NO 0
32
33 #define LEN_PADDING LEN_PADDING_NO
34
35
36
37 static int decryptWithPrivKey (char *pCryptFile, char *pClearFile, char *
       ↪ pPrivKeyFile, bool hasPadding);
38
39
40 int main()
41 {
42 //time_t t1, t2;
43     printf("-----------------------------------\n");
44     printf("Decrypt first file, without padding\n");
45     decryptWithPrivKey ("crypted_nopadding.txt", "clearFile_nopadding.txt", "rsa.
           ↪ pem", false);
46     printf("First file ok\n");
47     printf("-----------------------------------\n");
48
49     printf("Decrypt second file, with padding\n");
50     decryptWithPrivKey ("crypted_padding.txt", "clearFile_padding.txt", "rsa.pem"
```

```
        ↪ , true);
51      printf("Second file ok\n");
52      printf("---------------------------------\n");
53
54      printf("Decrypt third file, with padding\n");
55      decryptWithPrivKey ("challenge.rsa", "clearChallenge.txt", "
            ↪ rsaprivatechallenge.pem", true);
56      printf("Third file ok\n");
57      printf("---------------------------------\n");
58      printf("All done");
59
60      return (0);
61
62  }
63
64
65
66  static int decryptWithPrivKey (char *pCryptFile, char *pClearFile, char *
        ↪ pPrivKeyFile, bool hasPadding)
67  {
68  FILE *pClearFile1;
69  FILE *pCryptFile1;
70  FILE *pPrivKeyFile1;
71  unsigned char *pBufClear;
72  unsigned char *pBufCrypt;
73  EVP_PKEY *pPrivKey;
74  int lenPrivKey;
75  int len;
76  int len1;
77
78
79
80
81      ERR_load_crypto_strings();
82
83      pPrivKey = EVP_PKEY_new();
84
85      pPrivKeyFile1 = fopen (pPrivKeyFile, "r");
86      PEM_read_PrivateKey(pPrivKeyFile1, &pPrivKey, 0, NULL);
87      fclose (pPrivKeyFile1);
88
89      lenPrivKey = EVP_PKEY_size(pPrivKey);
90      pBufCrypt = malloc(lenPrivKey);
91      pBufClear = malloc(lenPrivKey);
92
93
94      pCryptFile1 = fopen (pCryptFile, "r");
95      pClearFile1 = fopen (pClearFile, "w");
96
97      for (len=lenPrivKey; len == lenPrivKey; )
98      {
99          len = fread (pBufCrypt, 1, lenPrivKey, pCryptFile1);
100         if (len == lenPrivKey)
101         {
102             //-----------------------if openssl version > OpenSSL
                ↪ 1.1----------------------------------
```

```
103       if (hasPadding) {
104         len1 = RSA_private_decrypt(len, pBufCrypt, pBufClear, EVP_PKEY_get1_RSA(
                ↪ pPrivKey),RSA_PKCS1_PADDING);
105       } else {
106         len1 = RSA_private_decrypt(len, pBufCrypt, pBufClear, EVP_PKEY_get1_RSA(
                ↪ pPrivKey),RSA_NO_PADDING);
107       }
108 
109             if (len1 >= 0)
110                fwrite (pBufClear, 1, len1, pClearFile1);
111         }
112     }
113 
114     fclose (pClearFile1);
115     fclose (pCryptFile1);
116 
117 
118     EVP_PKEY_free(pPrivKey);
119     free (pBufCrypt);
120     free (pBufClear);
121     ERR_free_strings();
122     return (0);
123 }
```