Home Assignment 5

Advanced Web Security

2017

B-assignments

For grade 3, complete the three B-assignments below and solve them in groups of two students.

B-1 The following is based on a true event. Joe User has created a private RSA key. Accidentally, it ended up on some public Internet location, i.e., some sort of cloud service. Joe tried to save the situation by "censoring" part of the published private key.

```
$ cat key.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDGlcensoredcensoredcensoredcensored1TUxhnjkCbowxZc
7PIpI1E2Po6aIgCBd9+6i0NUIfYm8vR6kqiqLz8k8o4LYoBkq/9Jx7pgV2Jqhr4u
wvlaQQUzi9c4qPKXp+QGoUu9f1zp8ORIMpeJmF7uA2ODC93uba07qdC6twIDAQAB
AoGBAIovDuYnGiiQS6K27L4EY8e/5sbqAwdlTOVlWsfz+ai3DLNiFPSbbT1Wx9G4
4b06X6O258SD1suZ/g/ICnmnxxe5ua3a5+iiDIwGYmBDcNfq5gMq/d+1/UJF/Bb4
A1nuH2iUg6gRTPEpbg2+RYwquyWenFbqfHMgXqbHVGmOXj7hAkEA8rChKjs5zVmd
j9Gk53psry4CtuxRc39NrHuLqat9Iu0MA51Sgv4c+8dgo75DVAnT5PoLBhHJJAVa
e+rUMC4kfwJBANF7jcKzJ2UuPmL6JpbWcyirybjMIm2eCxR5U1bYlNYT+A49oOFS
Eg5woswgCyH9gDPk2Zwpq3qud9HD7Rn0bckCQQDHgwdrRXc2ZybN1eZAWffBaAzZ
PpuTXKOJWaOuX4mnTcLjsdDkWW2QWw8Kbd7B1rZ49kpbugFmeHQzjRDVbwmXAkBm
T3nFBcrP1+4QWSxPrx0/V+eFoe2OrAmtTjQtzkmi5M3Z5q+UXIkFFG3uVBgb2bur
nLHLW26s1Fkg0hgS/RZBAkAFnE+7QvRCW4+v3OsIkN63f+GIjHfCuv8L15RpBLlf
XXQyOmmu8YekTu5vbFHtSAiLyuW1yCeSsNmKYkX6Ew99
-----END RSA PRIVATE KEY-----
```

Flo Friend has sent Joe User an encrypted message, which you have intercepted:

```
Qe7+h9OPQ7PN9CmF0ZOmD32fwpJotrUL67zxdRvhBn2U3fDtoz4iUGRXNOxwUXdJ2Cmz7zjS0DE8
ST5dozBysByz/u1H//iAN+QeGlFVaS1Ee5a/TZilrTCbGPWxfNY4vRXHP6CB82QxhMjQ7/x90/+J
LrhdAO99lvmdNetGZjY=
```

a) Recover the private key in PEM format.

b) Decrypt the message.

There are several ways to solve this, so make sure that you clearly explain each step in your solution. Hint: OpenSSL can be very helpful, in particular in the decryption phase.

Assessment:

- Upload your report to Urkund, paul.stankovski.lu@analys.urkund.se.
- Upload your report to Moodle (it will be manually graded).

B-2 CMS can be used to nest encryptions, signatures MACs and message digests in any order. CMS is e.g., used by the S/MIME message format for sending signed and encrypted emails. Alice has sent an important message to Bob using the S/MIME format. Instead of signing it, she has first hashed the message, and then encrypted it using the Enveloped Data content type.

You will receive 3 emails, named mail1.msg, mail2.msg and mail3.msg. All are sent from Alice to Bob by first hashing them using
`openssl cms -digest_create ...`
and then enveloping the result using
`openssl cms -encrypt ...`
However, only one of the messages has a valid hash. Your task is to find the subject line of the message with a valid hash. The subject will be an integer. The problem is easiest solved by using the `openssl cms` tool.

In addition to the emails, you will be given Bob's private key (`keyreceiver.pem`), Bob's certificate (`certreceiver.pem`) and a CA certificate (`CAcert.pem`) that has been used to sign Bob's certificate.
Assessment:

- There will be one Moodle question following the problem statement above. **Both students must finish the Moodle quiz**. There will be a test quiz on Moodle which you can use to test your OpenSSL commands. You can try the test quiz as many times as you like. The test quiz will not be graded.

B-3 The following excerpt from PKCS #1 v2.2 describes the ASN.1 format of an RSA private key with additional CRT information for decryption efficiency (the most common in practice).

```
RSAPrivateKey ::= SEQUENCE {
version           Version,
modulus           INTEGER,  -- n
publicExponent    INTEGER,  -- e
privateExponent   INTEGER,  -- d
prime1            INTEGER,  -- p
prime2            INTEGER,  -- q
exponent1         INTEGER,  -- d mod (p-1)
exponent2         INTEGER,  -- d mod (q-1)
coefficient       INTEGER,  -- (inverse of q) mod p
otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
```

First learn how ASN.1 works. Then implement your own function for DER-encoding large (up to 2048-bit) integers.

Example 1: The (decimal) integer 2530368937 has TLV DER-encoding `02050096d25da9`. Note that a zero byte `00` is appended to the hex representation of the integer value. This is because the value is represented in two's complement, which is a *signed* representation of integers where the highest bit is the sign bit. Without the zero padding, the number `96d25da9` would therefore represent a negative value.

While the encoding above uses the short definite form to represent the length part L in the TLV encoding, your implementation needs to support long definite form (using `81` and `82`) as well.

Example 2: If, say, 129 bytes are needed to represent the integer (the value part V of the TLV), then the L part is represented as `8181`. The first `81` denotes long definite length encoding with one length

byte, and the length itself is encoded in the second byte (129 is `81` in hex). Note that here we do not pad with zero bytes as in Example 1 (lengths are never negative).

Your core function should output a byte vector, but prepare an alternative version that outputs a string representation of the byte vector, as the one shown in Example 1. The alternative implementation will be useful later when you take the Moodle quiz.

Using this core function as a subroutine, implement another function that takes the private RSA parameters p and q as input and outputs the entire Base64-encoded DER-encoding of that private key according to the above format. This is usually what you will find if you look inside PEM-files containing private RSA keys.

Example 3: Given $p = 2530368937$, $q = 2612592767$ and $e = 65537$ (all decimal), you can compute the following values.

parameter	decimal value	DER-encoding
n	6610823582647678679	`02085bbe5d05d47d76d7`
e	**65537**	`0203010001`
d	3920879998437651233	`02083669c395b9cf7321`
p	**2530368937**	`02050096d25da9`
q	**2612592767**	`0205009bb9007f`
exponent1	2013885953	`020478097601`
exponent2	1498103913	`0204594b4069`
coefficient	1490876340	`020458dcf7b4`

Note that `Version` is an `INTEGER` with value 0 (zero). Furthermore, we will always use the value $e = 65537$. Also, for our purposes in this assignment, the optional parameter `otherPrimeInfos` will simply be omitted.

The DER-encoded RSA private key is then assembled as

`303c02010002085bbe5d05d47d76d70203010001020836...<omitted bytes>...4069020458dcf7b4`,

with the Base64-encoded RSA private key (your answer) being

`MDwCAQACCFu+XQXUfXbXAgMBAAECCDZpw5W5z3MhAgUAltJdqQIFAJu5AH8CBHgJdgECBFlLQGkCBFjc97Q=`

Hint: There are several online parsing tools that you may find useful during development, e.g., `https://lapo.it/asn1js/`.
Note: You may utilize third-party code for Base64-encoding.

Assessment:

- There will be two Moodle questions for this part. The first (0.5p) will ask you to DER-encode integers. That is, given an integer, you will answer with the DER-encoding of that integer in textual format, as detailed above.

  The second question (1.5p) will ask you to convert given integers p and q according to the full CRT-enhanced format given above. Your answer should be the Base64-encoding of the DER-encoding as a single line of text.

  Both students must finish the Moodle quiz. There will be a test quiz on Moodle that you can use to test your implementation. You can try the test quiz as many times as you like. The test quiz will not be graded.

- Upload your code to Urkund, paul.stankovski.lu@analys.urkund.se.

C-Assignments

For grade 4, complete the C-assignment below and solve it in groups of two students.

C-1 A problem with public key cryptography using a *Public Key Infrastructure*, PKI, is that the keys need to be distributed prior to encryption which may not always be feasible. Instead, one may use *Identity Based Encryption*, IBE. In an IBE system, the public key is generated from a known identity, e.g. an email address. A trusted third party, known as the *Private Key Generator*, PKG, generates the corresponding private key.

One implementation of an IBE system is known as Cock's encryption scheme, which is based on the difficulty of finding quadratic residues[1]. Given a and p, a is said to be a *quadratic residue modulo* p if there exists an integer b such that

$$a \equiv b^2 \pmod{p}$$

otherwise, a is a *non-residue*. It is customary to use the *Legendre symbol*:

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{if } a \equiv 0 \pmod{p} \\ 1, & \text{if } a \text{ is a quadratic residue modulo } p \\ -1, & \text{if } a \text{ is a quadratic non-residue modulo } p \end{cases}$$

where p is an odd prime number. For composite numbers, e.g. $M = p \cdot q$, the *Jacobi symbol* is used instead, which is just a product of the Legendre symbol of the prime factors of M. There are more properties for the Jacobi symbol[2]. You are given the code for calculating the Jacobi symbol, see listing 1 and 2.

Read the paper,

> Cocks. "*An Identity Based Encryption Scheme Based on Quadratic Residues*" Proceedings of the 8th IMA International Conference on Cryptography and Coding, 2001

and implement both the PKG and the decryption function. Note that in the paper for encryption, (a/t) means $(a \cdot t^{-1})$, where t^{-1} is the multiplicative inverse modulo M. Encode "+1" as 1 and "-1" as 0.

Your program should take as input

- your public identity, e.g. `alice@crypto.sec`,
- the primes, p and q,
- a list of the encrypted bits.

To derive the public *identity value*, a, use SHA-1 repeatedly on the given *identity*, e.g.

$$a_{i+1} = H(a_i)$$
$$a_0 = \texttt{alice@crypto.sec}$$

Encode the identity, e.g. `alice@crypto.sec`, as a byte array.

Example:
You have received an encrypted message to your mail, which is your public identity. You are given the following values:

[1] `https://en.wikipedia.org/wiki/Quadratic_residuosity_problem`
[2] `https://en.wikipedia.org/wiki/Jacobi_symbol#Properties`

```
id: walterwhite@crypto.sec
p: 9240633d434a8b71a013b5b00513323f
q: f870cfcd47e6d5a0598fc1eb7e999d1b

Encrypted bits:
83c297bfb0028bd3901ac5aaa88e9f449af50f12c2f43a5f61d9765e7beb2469
519fac1f8ac05fd12f0cbd7aa46793210988a470d27385f6ae10518a0c6f2dd6
2bda0d9c8c78cb5ec2f8c038671ddffc1a96b5d42004104c551e8390fbf4c42e
```

The identity value, a, is computed to be
`25a4d152bf555e0f61fb94ac4ee60962decbbe99`
The PKG computed the private key, r, to be
`814a8c2282ca8f4d0f2b2b72dfeeee6e5e3d8f438c039bdb5d059550739fdcec`
Decrypting the above bits, we get `[1, 1, -1]` which we decode to $110_2 = 6$.

```python
def jacobi(a, m):
j = 1

a %= m
while a:
t = 0
while not a & 1:
a = a >> 1
t += 1

if t&1 and m%8 in (3, 5):
j = -j

if (a % 4 == m % 4 == 3):
j = -j

a, m = m % a, a

return j if m == 1 else 0
```

Listing 1: The Jacobi symbol in Python.

Do not fear: The assignment is not as hard as it looks.

Assessment:

- Upload your code to Urkund, paul.stankovski.lu@analys.urkund.se. One upload per group is sufficient.
- There will be one Moodle question following the problem statement above. **Both students must finish the Moodle quiz**. There will be a test quiz on Moodle, where you can try your implementation as many times as you like. The test quiz will not be graded.

```
import java.math.*;

class Jacobi {

static BigInteger zero = BigInteger.ZERO;
static BigInteger one = BigInteger.ONE;
static BigInteger two = new BigInteger("2");
static BigInteger three = new BigInteger("3");
static BigInteger four = new BigInteger("4");
static BigInteger five = new BigInteger("5");
static BigInteger eight = new BigInteger("8");

public static void main(String[] args) {
BigInteger a = new BigInteger("12");
BigInteger b = new BigInteger("17");
System.out.println("jacobi (12/17) (= -1): " + jacobi(a, b));
}

static int jacobi(BigInteger n, BigInteger m) {
int j = 1;
int t;
BigInteger tmp;

n = n.mod(m);
while (!n.equals(zero)) {
t = 0;

while (!n.and(one).equals(one)) {
n = n.divide(two);
t++;
}

BigInteger mmod8 = m.mod(eight);
if (((t & 0x01) == 1) && (mmod8.equals(three) || mmod8.equals(five))) {
j = -j;
}

if (n.mod(four).equals(three) && m.mod(four).equals(three)) {
j = -j;
}

tmp = n;
n = m.mod(n);
m = tmp;
}
if (m.equals(one)) {
return j;
}
return 0;
}

}
```

Listing 2: The Jacobi symbol in Java.