Security II: Supervision 2
– exercises

Markus Kuhn with edits by David Llewellyn-Jones

Lent 2018 – Part II

Some of the exercises require the implementation of short programs. The model answers use Perl (see Part IB Unix Tools course), but you can use any language you prefer, as long as it supports an arbitrary-length integer type and offers a SHA-1 function. Include both your source code and the required output into your answers.

Before starting any programming exercise, first estimate how many minutes the solution will take you. Please include in your answers both this estimate, as well as the actual time you required.

**Exercise 1:** Use Euler’s theorem to calculate the inverse

(a) \( 5^{-1} \mod 7 \)

(b) \( 5^{-1} \mod 12 \)

(c) \( 5^{-1} \mod 15 \)

**Exercise 2:** On slide 49 of the lecture notes the following definitions is provided for CPA-security of a public-key encryption scheme \( \Pi \).

\[
\mathbb{P}(\text{PubK}_{A,H}^{\text{cpa}}(\ell) = 1) \leq \frac{1}{2} + \text{negl}(\ell). \tag{1}
\]

Assuming that the bit \( b \in_R \{0,1\} \) is picked by the challenger uniformly at random, demonstrate that the above definition is equivalent to the following.

\[
\text{Adv}_{\text{PubK}_{A,H}^{\text{cpa}}}(\ell) \leq \text{negl}(\ell). \tag{2}
\]

In the above we define

\[
\text{Adv}_{\text{PubK}_{A,H}^{\text{cpa}}}(\ell) = |\mathbb{P}(b = 1 \text{ and } b' = 1) - \mathbb{P}(b = 0 \text{ and } b' = 1)|.
\]

where \( b \) and \( b' \) are the challenger bit and adversary guess respectively from the CPA game defined on slide 47 from the lectures.

**Exercise 3:** Let \( G(1^\ell) \) be a polynomial-time group generator that outputs an \( \ell \)-bit prime \( p \) and a generator \( g \) of \( \mathbb{Z}_p^\ast \). Show that the DDH problem is not hard relative to \( G \).

[Hint: Recall that Euler’s criterion allows efficient detection of quadratic residues.]
**Exercise 4:** Let \( g \in \mathbb{Z}_p^* \) be a generator of the cyclic group \( G \) where \( q = |G| \) and let \( y \in G \).

(a) The following Python code implements a brute-force program that, given \( g, y \in G \), iterates through elements \( x \in \mathbb{Z}_q \) until it finds \( x \) such that \( g^x = y \).

```python
def naive(g, q, y, p):
    x = 0
    while x < q:
        exp = pow(g, x, p)
        if exp == y:
            return x
        x = x + 1
    return None
```

Alter this program to use the Baby-Step-Giant-Step (BSGS) algorithm from slide 83 of the lectures.

(b) Use your code to solve the discrete logarithm problem for \( g = 256 \) and \( y = 35630012572 \in \mathbb{Z}_p^* \) with \( p = 62831853803 \) and where \( g \) is a generator of the group \( G = \{256^i \in \mathbb{Z}_p^*: i \in \mathbb{Z}_q\} \).

Give timings for both the naive and BSGS approaches.

(c) Now set \( g = 64 \in \mathbb{Z}_p^* \) where \( p = q = 6421 \) and \( y = 66 \).

What numbers do the two approaches produce in this case and are they correct answers for the equation \( 64^x = 66 \mod 6421 \)? Why do the naive and BSGS approaches produce different results and what does this tell you about the efficiency of the BSGS algorithm in this case?

**Exercise 5:** With RSA encryption, it is common practice to choose \( e \) as a small number (e.g., 3, 17, \( 2^{16} + 1 \)).

(a) How does this affect the speed of encryption?

(b) If you wanted to make decryption faster, could you simply set \( d \) to one of these three values instead?

(c) How else can RSA decryption be calculated more efficiently using the Chinese Remainder Theorem and Fermat’s little theorem?

**Exercise 6:** In the textbook RSA encryption scheme, with \( n = pq \) being a product of two different primes and \( ed \mod \varphi(n) = 1 \), the identity \( m^{ed} \mod n = m \), which states that we obtain the same plaintext \( m \) after encryption and decryption, is only guaranteed by Euler’s theorem for any \( m \in \mathbb{Z}_n^* \), that is if \( \gcd(n, m) = 1 \).

(a) Show that it actually also holds for any \( m \in \mathbb{Z}_n \). [Hint: CRT]

(b) Conversely, if we instead had chosen \( n = p^2 \) being the square of a prime number (i.e., \( p = q \)), show a simple example for the fact that in this case \( ed \mod \varphi(n) = 1 \) does not imply \( m^{ed} \mod n = m \) for all \( m \in \mathbb{Z}_n \).
**Exercise 7:** A device vendor uses the DSA signature scheme to digitally sign configuration updates. The system parameters are

\[
p = 0x8df2a494492276aa3d25759bb06869cbeac0d83af8d0cf7cbb8324f0d7882e5 \\
d0762fc5b7210eafc2e9adac32ab7aac49693dfb83724c2ec0736ee31c80291 \\
q = 0xc773218c737ec8ee993b4f2ded30f48edace915f \\
g = 0x626d027839ea0a13413163a55b4cb500299d5522956cefb3bff10f399ce2c2e \\
71cb9de5fa24babf58e5b795219259cc42e9f6f464b088cc572af53e6d78802
\]

and the vendor’s public key is

\[
y = 0xeb772a91db3b69af90c5da844d7733f24270bdd11aac373b26f58ff528ef2678 \\
94b1e746e3f20b8b89ce9e5d641abbff3e3fa7dedd3264b1b313d7cd569656c
\]

The vendor has already signed two messages:

\[
H(m_1) = \text{SHA-1("Monday")} = 0x932eeb1076c85e522f02e15441fa371e3fd000ac \\
r_1 = 0x8f4378d1b2877d8aa7c0687200640d4bba72f2e5 \\
s_1 = 0x696de4ffbf102249ae907f348fb10ca704a4b186
\]

\[
H(m_2) = \text{SHA-1("Tuesday")} = 0x42e43b612a5dfae57ddf5929f0fb945ae83cbf61 \\
r_2 = 0x8f4378d1b2877d8aa7c0687200640d4bba72f2e5 \\
s_2 = 0x25f87cbb380eb4d7244963e65b76677bc968297e
\]

(a) Calculate \(g^q \mod p\).

(b) Verify that the two signatures are valid under the given public key \(y\). (Preferably perform the required calculations using the modinv and modexp routines that you implemented yourself in exercises 9 and 12. Alternatively, download a computer-algebra system, such as Sage or PARI/GP.)

(c) What mistake did the vendor make when generating these two signatures?

(d) Exploit this mistake to reconstruct the secrets \(k\) and \(x\) used to generate these signatures. [Hint: Start by subtracting the two defining equations for \(s_1\) and \(s_2\) from each other.]

(e) Use this information to falsify a signature for the new message

\[
H(m_3) = \text{SHA-1("Wednesday")} = 0x5656b9b79b0316fc611a9c30d2ffac25228b8371
\]

and then verify its correctness against public key \(y\).

**Exercise 8:** Computer Science Tripos Part II, 2013, Paper 8, Question 12

The RSA public-key cryptosystem performs calculations in the group \(\mathbb{Z}_n\), with \(n = pq\) being the product of two large prime numbers \(p\) and \(q\). The public key consists of the tuple \((n, e)\), with \(\gcd(\phi(n), e) = 1\), and the corresponding private key is \((n, d)\). A message \(m \in \mathbb{Z}_n\) is encrypted via \(c = m^e \mod n\) and decrypted as \(m = c^d \mod n\).

(a) Given \(p\), \(q\), and \(e\), how can you apply the extended Euclidian algorithm to find a suitable \(d\)? [6 marks]

(b) If we modified RSA to use as the public modulus a prime number instead of a composite of two large prime numbers, that is \(n = p\) instead of \(n = pq\), would this affect its security, and if so how? [4 marks]
In the UltraSecure virtual-private network, each router knows of each of its remote
communication peers the RSA public key \((n, e)\), which all have \(e = 3\) and \(2^{1023} \leq n < 2^{1024}\). If router \(alice\) needs to establish a shared 256-bit AES secret key \(k\) with
remote router \(bob\), it looks up \(bob\)’s \((n, e)\) and then uses this method:

- \(alice\) picks \(k \in \{0, 1\}^{256}\) by reading 32 bytes from \(/dev/random\)
- \(alice\) interprets \(k\) as binary integer \(m\) with \(0 \leq m \leq 2^{256}\)
- \(alice\) sends \(c = m^e \mod n\) to \(bob\)
- \(bob\) decrypts \(c\) into \(m\) and recovers \(k\) (by removing leading zeros)

Then \(alice\) and \(bob\) secure the rest of their communication with shared secret \(k\).

(i) How could an eavesdropper obtain \(m\) from \(c\)? [4 marks]

(ii) Suggest a better method of using RSA to establish an AES key than the one
given above. [6 marks]