

Network Security Winter Term 2013/2014
Assignment 1

Submission until **Thu, 14 November 2013, 12:00 CET (UTC+1)**.
Tutorial hours probably on **Wed, 20 November 2013**.

General remarks:

- The purpose of the assignments is to help you deepen your understanding of the subject and to get your questions answered. In our experience, students who have done well in the assignments also do well in the exams. We thus recommend doing the assignments and also to attend the tutorial hours.
- We will award credits for every assignment and task, split in T-credits (for questions) and P-Credits (for programming tasks). If you achieve at least 50% of all credits in *each* of the two categories, distributed over *all* assignments, you earn a bonus of 0.3 in your exam grade.
- There will be 4 or 5 assignments, roughly every 2–3 weeks, depending on how fast the lecture progresses.
- Students are allowed to form a team of **exactly 2 persons** and submit a solution together.
- Submit your solutions by SVN – see instructions below.

Important directions:

- Be sure you do not violate our plagiarism guidelines (see slide deck).
- Submit text in a **PDF**. Submit **well-documented code**. Your documentation is expected to explain your intentions and every important step you do. Only executable code will be graded. If you choose Java, make sure you comply with the instructions for the format of the program's output!
- A UNIX system is best for the programming. If you don't have a Linux system, you can also use a Virtual Machine – we provide a link to one on the lecture's home page.
- If you downloaded the VM before 2013-11-06 14:00 CET, please run `sudo apt-get install subversion` as your first command.

Directions for SVN:

- Check out the SVN directory with
`svn co --username LRZID https://projects.net.in.tum.de/svn-tum/netsecw13`
- The user name is your ID at LRZ. E.g., `gu27jaf`

- You will only be able to read/write in your own directory and the public ones.
- Check in (partial) solutions with:
`svn ci -username LRZID`
- SVN documentation can be found at <http://svnbook.red-bean.com>.
- Teams: check in to the directory of *one* of the team members. Add a text file in *both* directories of the team members stating who the team is (state the LRZ ID!).

Task 1 Understanding crypto primitives (4 T-credits)

The goal of this task is to understand which cryptographic functionality can be used to achieve a certain security goal. Assume a client-server architecture with an attacker on the data path. We define an attacker model in which the attacker is able to do the following:

- Eavesdrop on messages, i.e. listen to the transmission as a sequence of bits
- Modify messages
- Delay messages
- Delete messages

We assume a protocol with mechanisms for the following security services:

- Confidentiality by using symmetric cryptography
- Message integrity, applied per message, using MACs
- Authenticity of the sender

These are the *only* mechanisms the protocol supports.

a) Some questions with respect to keys:

1. The designers of the protocol are very lazy and want to use the same symmetric key for encryption and the MACs. They are going to use AES-CBC for both. What weaknesses are they going to introduce? (1 T-credit)
2. Regarding the number of necessary keys (1 T-credit):
 - a) Assume there are n users in the system. How many keys need to exist to support confidentiality between all participants? Use O -notation if you like.
 - b) If public key cryptography were used for encryption instead of symmetric cryptography, how many keys would be necessary in the system? Use O -notation if you like.

b) Which of the attacks in our attacker model are reliably detectable by the receiver? Give reasons (1 T-credit).

c) The designers also want to detect message replay, i.e. the attacker storing and later forwarding a message. How can that be added to the protocol? (1 T-credit)

Task 2 Passwords and computational effort (4 T-credits)

The goal of this task is to deepen your understanding of passwords and the computational effort involved in cracking them.

a) The recommended password length for offline applications (e.g. disk encryption) is significantly higher than that for online applications (e.g. Web-based login): at least 12-14 characters versus at least 8-10 characters (given a Latin alphabet, plus numbers and special characters). What is the crucial difference between the two applications that makes it possible to make do with shorter passwords in the online case? (1 T-credit)

b) Given the same underlying alphabet and randomly chosen password characters, which password is stronger: (1) a password of 8 characters consisting of upper case and lower case letters, numbers, special characters. etc. or (2) a password of 12 characters consisting only of lower case letters? Give the general formula to determine the password's strength and explain which factor in the formula is the most important one in making a password strong? (1 T-credit)

c) A *passphrase* is a sequence of words from a vocabulary, e.g. `correct_horse_battery_staple`. A popular method to create a passphrase is *diceware*¹: given an indexed vocabulary of words that occur in a natural language, a user uniformly and randomly chooses n words from the list and concatenates them (separated by spaces).

1. Assume an attacker who tries brute-force and knows which method you have used: password over a certain alphabet or a passphrase using a given diceware vocabulary.

Determine which is stronger: (1) a passphrase of 7 words, chosen using a variant of diceware with 5 dice and a vocabulary of corresponding size (!) or (2) a password of 10 characters over an alphabet of 64 characters (your computation must be part of your solution). (1 T-credit)

2. Assume the attacker can brute-force at a rate of 2,000,000 tries per second. How long does he on average need to find the password? How long to find the passphrase? (1 T-credit)

Task 3 Hash functions and passwords (4 T-credits)

a) Many multi-user systems do not store user passwords as plain text. Instead, they compute $p' = \text{SHA1}(r|p)$, i.e. a SHA1 hash of a random string r to which the password is concatenated. They then store (r, p') . What is achieved with this? (1 T-credit)

b) Why is it not a problem if r is known to an attacker? (1 T-credit)

c) Assume you own a company *Business Doping Inc.*, or *B-Dope* for short. Two engineers turn up for a job interview. You ask both the same question: 'How should I store user passwords?' Engineer A says: 'Store the passwords as hash values (as described in a)).' B says: 'Choose a global key of length 256 bits and use AES to encrypt the passwords individually, using ECB mode.' Following sound cryptographic considerations, which engineer should you hire – and why? Give 2 (cryptographic) reasons. (2 T-credits)

¹<http://en.wikipedia.org/wiki/Diceware>

Task 4 Implementing RSA (20 P-credits)

We will implement an inefficient (and not particularly secure) version of the RSA algorithm here. We accept submissions in Python and Java.

You find a skeleton program for Python on our homepage. Independently of your choice of language: **read** the skeleton code and comply with the constraints mentioned therein! In particular, the choice of e **must** be the smallest possible e (otherwise our machine-verification code will fail!). Make sure you do not change the function signatures nor their output data types, nor the format of the output. Please call the file you submit `rsa.py/java`.

We also accept Java submissions. If you choose Java, the input values to the program **must** be on the command line and of the form shown below. The first two parameters are the prime numbers to use. The third parameter is the message to encrypt (we use an integer representation!).

The output of the program **must** go to standard output (**not** a file). It **must** conform to the output defined in our Python skeleton.

Here are two sample invocations and input/output for you to test with. Correct code will produce the same values! If you find you are able to encrypt and decrypt but do not get the same values, check if your choice of e is really the smallest possible e !

```
ralph@firenze:$ ./rsa.py 43 103 1000
Pubkey:4429,5
Phi:4284
PrivKey:857
m:1000->790
c:790->1000
```

```
ralph@firenze:$ ./rsa.py 13 23 212
Pubkey:299,5
Phi:264
PrivKey:53
m:212->296
c:296->212
```

Here are some instructions. Happy hacking!

a) (10 P-credits) First, we create a key pair. We restrict our users to primes $p, q < 300$ to save ourselves some computation time. Hints:

- You will need to implement the Extended Euclidean Algorithm, EEA².
- A function `gcd()` can use EEA's output.
- Find e with $\text{gcd}(e, \phi(n)) = 1$ by simply looping over prime numbers $0 < e < 300$ and testing with `gcd()`. Choose the smallest e you find!

b) (5 P-credits) Compute the private key. Write a function `computePrivKey(e, $\phi(n)$)` that makes use of the EEA. The output is an integer d .

c) (5 P-credits) Now write the functions for encryption and decryption. Verify you get the same output as above.

²http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm