IT and Network Security
WiSe 2020/21

Practical Assignment No. 3

Nils Aschenbruck

Leonhard Brüggemann, Alexander Tessmer, **Stefanie Thieme**

Till Zimmermann

Release	24.11.2020
Submission	07.12.2020

General information for the practical assignment:

- Read the assignment sheet **thoroughly from top to bottom**, before you start working!
- For the admission to the exam, you have to acquire at least 8 points.
- You have to work on all assignments **on your own**.
- **(Code) plagiarism will not be tolerated!**
  This includes the use of code from other sources without reference or unusual amounts of code with references and applies to all parties involved.
- Submission is **no later than 9:00 a.m.** on the respective submission date via **StudIP**. Related files must be packed as a ZIP archive and uploaded in the assignment area of the lecture. The archive name must conform to the following pattern:
  ITS_WiSe_202021_PA3_*MatrNo*

  where ***MatrNo*** has to be replaced with your matriculation number. On unpacking the archive, a **subfolder with the same naming pattern** must be created automatically. Please note that potential changes to this submission method can occur for individual assignments.
- **Comply with all stated specifications** for file names, folder structure, program parameters, etc. Otherwise the time for assignment evaluation is unnecessarily extended and the result announcement will be delayed.
- Please use the **mailing list** of the lecture for questions of general interest. Alternatively you can attend the **Q&A tutoring session**. An application by mail to the teaching assistant (**thieme@uos.de**) is optional, but recommended for extensive questions.

Practical Assignment 3: Pretty Good Privacy (3 Points)

In the lecture, you have learnt about PGP as a practical application of cryptographic algorithms like RSA, AES or 3DES. To get the full hands-on experience, this assignment is twofold: First, you must use PGP for secure mail communication. Second, you must implement a trusted log server that utilises PGP.

3.1: Encrypted E-Mail Communication (1 Point)

This assignment is to show you the benefit of PGP for your day-to-day life. Learn how to encrypt your mail traffic!

a) Install the OpenPGP implementation GnuPG [1] and generate a PGP key pair for your *@uos.de* address, if you have not done so yet. Use the encryption algorithms of your choice and, if desired, add further mail addresses (IDs) to your key pair, which is automatically added to your local keyring. This is especially helpful for assignment 3.2, where you have to access said keyring from code.

b) To communicate with others using PGP, they need your public key. Therefore, you have to upload it to a *key server*. Even though these servers are synchronized, the actual exchange of keys can take some time. Hence, we urge you to upload your public key to `pgp.mit.edu`. In order to be able to revoke your key from those servers, you should generate a revocation key as well.

c) With the aid of your private key, you can sign your mails. Using your public key that was uploaded in the previous step, your communication partners can verify this signature. Beyond that, you can now receive encrypted mails. Of course, you can also send encrypted mails, if you have the recipient's public key. Search for and import the key of our address *sys-lehre@uni-osnabrueck.de* (key ID *64E2C6380E941015*). Make sure that the fingerprint from the retrieved key matches *8633 3255 F291 4643 17E3 DE03 64E2 C638 0E94 1015*.

d) Write a signed and encrypted mail with the following text to *sys-lehre@uos.de*, where *Name* and *Matriculation number* have to be replaced with your data:

    Kind regards!
    Name
    Matriculation number

   The subject is *ITS Greetings*

General Information & Helpful Hints

- Plugins such as Enigmail [2] offer PGP integration for mail applications and therefore simplify the usage of PGP greatly. For users of Thunderbird 78 mail encryption based on OpenPGP became even easier as those functionalities are already built into this version of the mail client.

- If we are unable to decrypt and verify your mail with the corresponding public key, or if there does not exist such a key on `pgp.mit.edu`, you will not receive the point for this assignment. Therefore, we advise you to check up on the successful upload of your key in time.
- Besides GnuPG[1]/OpenPGP[3] for Linux, macOS and Windows, there also exist apps for Android and iOS that aid you in signing and encrypting mails from your smartphone.

3.2: Trusted Log Server (2 Points)

Imagine that after your IT security audit, the company with the vulnerability from the previous assignment sheet has employed you to design a simple client-server architecture for logging the thermostat's temperature data. This redevelopment is to remedy the problems of the former system that you have exploited. Use PGP to sign and encrypt the thermostat's messages!

a) Implement a simple TCP client that sends signed and encrypted messages to a server. Use the python module python-gnupg [4]. Initialise the interface so that it uses the standard GnuPG home directory. In order to identify the correct PGP key, add a user input request. This provides the client with access to the key pair that was generated in assignment 3.1. (If you have chosen a directory other than the standard during the GnuPG key generation, you have to modify all instructions accordingly.) On the client side, communication looks like this:

   - The client establishes a connection to the server.
   - The client sends his public key to the server as plain text. In return, it receives the server's public key.
   - The client accepts command line input. Each message is signed with the private key from assignment 3.1 and encrypted with the server's public key, before it is sent.
   - For all messages that were sent according to the previous step, the client receives a status message from the server. If it is one of `UNTRUSTED` or `ERROR`, it should display an appropriate error message and terminate.

b) Implement a simple TCP logging server that checks and prints encrypted and signed messages. Use a different user account on your Kali machine with a keyring that must be empty on the first server start, containing no keys.
   On the server side, communication looks like this:

- The server checks if it owns a personal key pair. If it does not, it generates one for the fictional address *trustedlogs@server.com* and refrains from choosing a passphrase (which should never be done in reality).
- The server accepts one TCP connection at a time on port 4711.
- A clients connects to the server and sends its public key. The server adds this to its keyring and answers with its own public key in plain text.
- The server receives further messages, which it tries to decrypt and verify.
  - If decryption fails, the server answers with the status `ERROR` in plain text and prints an appropriate error message.
  - If the verification of the signature fails, the server answers with the status `UNTRUSTED` and prints the message and an appropriate warning.
  - If decryption and verification are successful, the server answers with `TRUSTED` and prints the message to the terminal.

General Information & Helpful Hints

- Your program should not only work if everything goes well, but also if problems occur. At least check with messages that lack a signature or encryption.
- Make sure that your programs terminate gracefully, even in the presence of connection failures, interrupts (e.g. SIGINT via CTRL-C) or the like.
- Do not confuse the packet that has to be used (which is called *python*-gnupg [4]) with the similar named packet gnupg! Although the latter is a derivative of the former, the documentations differ in parts. Watch out!
- Please note the note in the official python-gnupg documentation concerning key validation [5]!

Happy Coding!

Submission:

- ZIP-archive named *ITS_WiSe_202021_PA3_MatrNo.zip* that automatically creates the corresponding folder *ITS_WiSe_202021_PA3_MatrNo* on unpacking (cf. general information on page 1).
- This folder must contain all relevant source files (.py); at least:
  - the client: *encrypt_sign_client.py*
  - the trusted log server: *trusted_log_server.py*
- send the PGP-**encrypted** and **-signed** mail according to 3.1
- upload the zip-archive to **StudIP**: *IT- und Netzwerksicherheit → Tasks → PA3*

Literatur

[1] `https://www.gnupg.org/`

[2] `https://www.enigmail.net/index.php/en/`

[3] `http://openpgp.org/`

[4] `http://pythonhosted.org/python-gnupg/`

[5] `https://pythonhosted.org/python-gnupg/#encryption`

[6] `https://xkcd.com/1181/`