Information Security & Privacy Cryptography

Exercises (Weeks 2 and 3), 8 hours

Exercise 1: Probabilities

Imagine that you just sent an inappropriate e-mail message, M, and you want to revert the situation.

For integrity reasons the MD5 hash value of M is stored on another server (the integrity server). However, to save storage space, only the first N bytes ($N \leq 16$) of the e-mail hash are stored on the server (denote these by H). Recall that MD5 is always 128 bits (= 16 bytes = 32 hexadecimal digits) long. You have access to the e-mail server (to change your message) but **not** to the integrity server.

Your goal is to find another message, M' to substitute to M. For the substitution to be plausible and undetectable, the first N bytes of the hash of M must be the same as those of the hash of M.

Example: The message "F*ck you!" has the MD5 hash value of

`04``c48580c76bf1c7e6efa05048203db2`.

If you replace it with the message "I love you <3" the hash will be

`12``7da3a57a439387bdbd334f51866376`.

The change will be detected for $N = 1$ because `04` $\neq$ `12`.

If you want to go undetected you can use brute force do create a message such as "I adore information security and privacy." With the hash value of

`04``fad44c38966bff0390d01b805611ec`

The new message hash matches on the first byte (that is the first two hexadecimal digits). In the case of $N = 1$, the replacement would go undetected.

Question 1

What is the probability that a random e-mail message has the first N bytes of its MD5 hash equal to H?

Question 2

How many random (uniformly distributed) messages, on average, do you have to try before you find the first message whose hash collides with M (this means that their first N bytes are the same)?

Question 3

How many random messages (uniformly distributed) do you have to try to ensure that at least one of them collides with M with probability at least 99%? Assume $N = 4$.

Question 4
Write a Python program that computes the sha256 hash of an input text. The program should ask the user to input a text in the terminal (through the `input` function[1]). Once entered, the hash of the text should be computed and displayed to the user.

Hint:
Use the `hashlib` package to compute the hash function of a string (**Attention**: the hash function operates on raw *byte arrays* (not *strings)*; therefore, you must use the `encode` functions of the string object[2] to transform the text to binary). The most appropriate encoding parameter is probably 'utf8'. You can verify that in Preferences… > Editor > File Encodings. To display the result in hexadecimal form, `hashlib` contains a `hexdigest` method.
https://docs.python.org/3/library/hashlib.html

Question 5 [Homework: deadline 17.10.19, 6pm]
Consider the original M = 'F*ck you, dear professor!'. Write a short Python program to produce a different e-mail string, M', that has the same first N bytes of the MD5 hash function as M. Note that M must look like a genuine message, containing words and sentences that flow and are grammatically correct, and not be a random sequence of characters or a message padded with some characters. Start with $N = 1$ bytes and then try for higher values.

Hint:
As you saw in previous questions, the average number of tries to get one success is quite large, so think of a clever trick (e.g., use synonyms) to easily generate many random messages that look genuine, as asked.

Deliverable:
Upload on Moodle a text file named `h1_I.txt`. This file should have the following format:
- on the first line the string M' you suggest as replacement for the original e-mail.
- on the second line, the MD5 hash value of M'
- on the third line, the value of N (in bytes). Of course, we expect only one solution for the highest value of N that you could find.

Also upload a Python script file containing your code named `h1_I.py`.
This is a group exercise, so we only need one solution and one version of the code.

✓ **Correction criteria:**
0: The script does not work **and/or** no matching hexadecimal characters is found.
0.5: The script works and the first byte matches.
1: The script works and 2 or more bytes matches.

Challenge:
This is a challenge question. The group with the best result will get a reward. In the case of a tie, the group that submitted his answer first will be the winner. The challenge is to get to the highest N possible. (**Hint**: if you really want to win, try using a tool such as `hashcat` which will fully take advantage of the computing resources on your computers such as your GPU https://hashcat.net/)

[1] https://docs.python.org/3/library/functions.html#input
[2] https://docs.python.org/3/library/stdtypes.html#str

Exercise 2: Hashing

A common use of hashes is to check the integrity of downloaded files. The server provides the expected hash of the file, and the downloader computes the hash of the file he downloaded from the server and compares its hash with the expected one. A match indicates the download was successful.

Download the file Transmission-2.94.dmg from https://transmissionbt.com/download/, compute the hash of the downloaded file and verify that it matches the expected hash displayed on the website.

Try to download the file directly using Python (check out the `urllib`, `requests` or `wget` packages).
Alternatively, use the terminal with the `shasum` command to compute the hash. Type `shasum -h` to learn how to use it. On Windows, the `certutil` command[3] is able to compute hash of a file.

Exercise 3: Symmetric cryptography

In this exercise, we will use the cryptography library (conveniently named `cryptography`[4]) installed last week (either in the terminal with the pip command or directly from the dedicated menu in PyCharm).
Noé sent you an image encrypted with a symmetric key. You will have to load the key and the encrypted image in python in order to see what you received. The image was encrypted using AES encryption as a *block cipher* and the Electronic Codebook as a *mode of operation*. Download the “encrypted image + key” file on moodle.

> **Hint:**
> To open a file with Python, use the built-in `open` function[5]. You will need to read the file as binary. When using the `cryptography` library, all the key and the data have to be in binary format.
> You should have a look at the cryptography library documentation about symmetric encryption.[6]

Exercise 4: Secure e-mail communication using PGP

OpenPGP is a non-proprietary protocol for encrypting e-mail communication using public key cryptography. It is based on the original PGP (Pretty Good Privacy) software. The OpenPGP protocol defines standard formats for encrypted messages, signatures, and certificates for exchanging public keys. Source: openpgp.org
This protocol can be used by journalist or human right activists to communicate in a secure manner.

Step 1 – GnuPG:

GnuPG is a complete and free implementation of the OpenPGP standard as defined by RFC4880 (also known as PGP). GnuPG allows you to encrypt and sign your data and communications; it features a versatile key management system, along with access modules for all kinds of public key directories. GnuPG, also known as GPG, is a command line tool with features for easy integration with other applications.
GnuPG is one of the tools that Snowden used to reveal the secrets of the NSA. Source: gnupg.org

Install GnuPG

For macOS, open the terminal and type: `brew install gnupg2 pinentry-mac`. And then type the following command in the terminal:

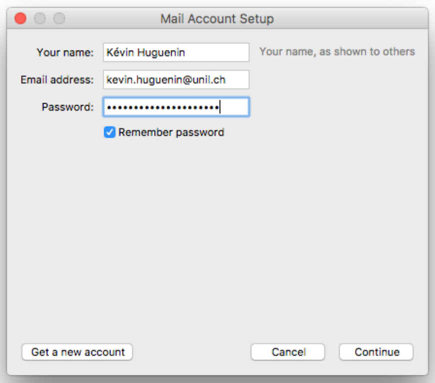
[3] https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/certutil#BKMK_hashfile
[4] https://cryptography.io/en/latest/
[5] https://docs.python.org/3/library/functions.html#open
[6] https://cryptography.io/en/latest/hazmat/primitives/symmetric-encryption/

```
echo -e "\npinentry-program /usr/local/bin/pinentry-mac" >> ~/.gnupg/gpg-agent.conf
```

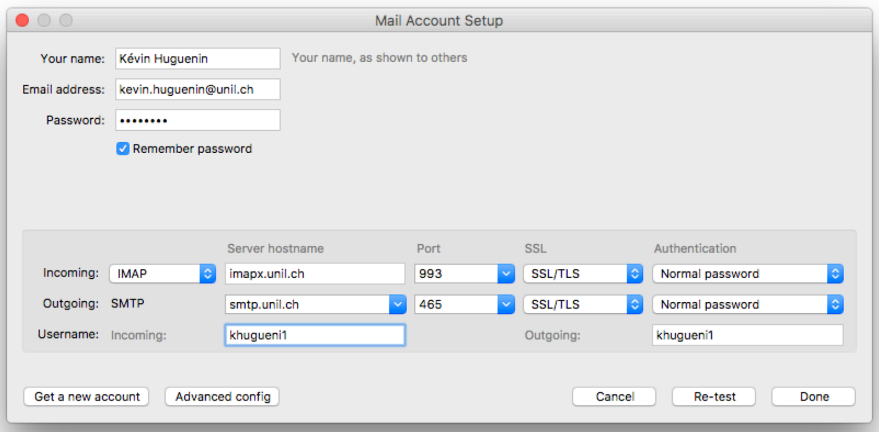
For Windows go to https://gnupg.org/download/ and follow the instruction.

Step 2 –Thunderbird:

Download and install the Thunderbird mail client (`brew cask install thunderbird` for macOS, or https://www.thunderbird.net/) and configure it as described at (use **your** username!): https://www.unil.ch/ci/home/menuinst/catalogue-de-services/email-et-calendrier/messagerie/documentation/email/generic-e-mail-clients.html



Your name: Kévin Huguenin — Your name, as shown to others
Email address: kevin.huguenin@unil.ch
Password: ••••••••••••••••••••
[✓] Remember password



Your name: Kévin Huguenin — Your name, as shown to others
Email address: kevin.huguenin@unil.ch
Password: ••••••••
[✓] Remember password

		Server hostname	Port	SSL	Authentication
Incoming:	IMAP	imapx.unil.ch	993	SSL/TLS	Normal password
Outgoing:	SMTP	smtp.unil.ch	465	SSL/TLS	Normal password
Username:	Incoming:	khugueni1		Outgoing:	khugueni1

Step 3 –Enigmail:

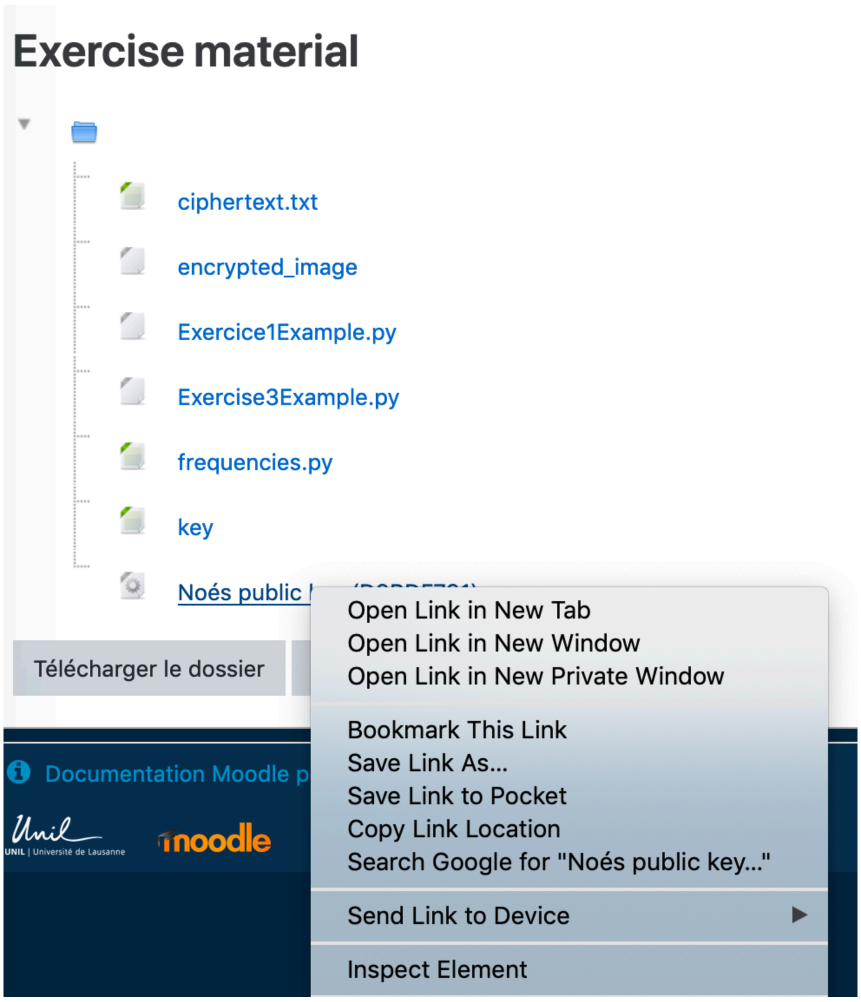
Follow this tutorial to add Enigmail to Thunderbird: https://www.enigmail.net/index.php/en/user-manual/installation-of-enigmail

Restart Thunderbird and choose Tools > Account Settings..., then click on Enable OpenPGP support.
In the menu bar, select Enigmail > Setup Wizard. Follow the instruction (you can skip the GPGtools part as you already installed the tools with brew): https://www.enigmail.net/index.php/en/user-manual/quick-start

⚠ **Attention**: Use a passphrase to protect your PGP key.

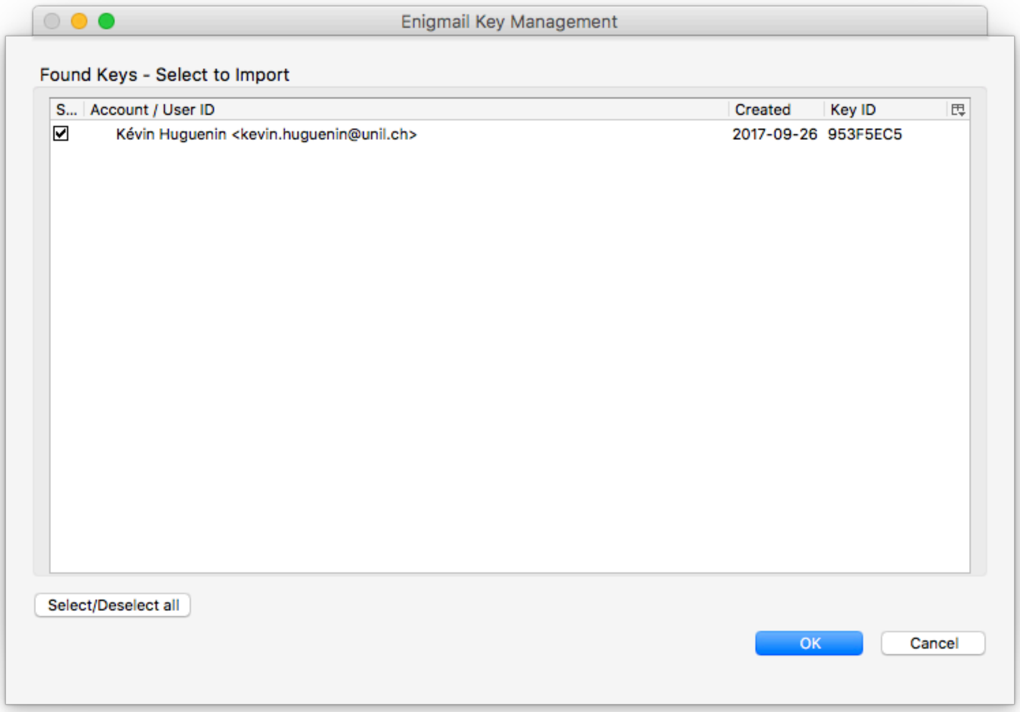
Step 4 - Exercise:

Exchange key – from file:
Next, go on Moodle and download Noé's public key which is saved in the file 'Noés public key (D9BDF791).asc'.



Then, import this file form the location where you saved it into Thunderbird (see https://www.enigmail.net/index.php/en/user-manual/key-management#Importing_public_keys).

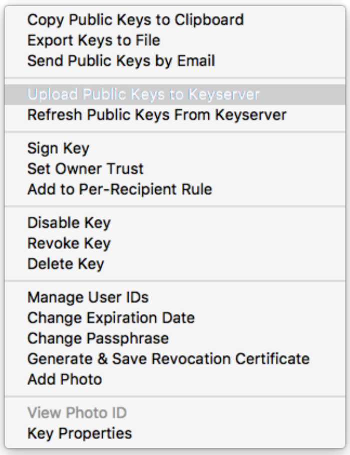
Exchange Key – from PKI:
You will now try a different method to exchange public keys. In Thunderbird, go to the Enigmail menu and select 'Key Management'.


Enigmail Key Management

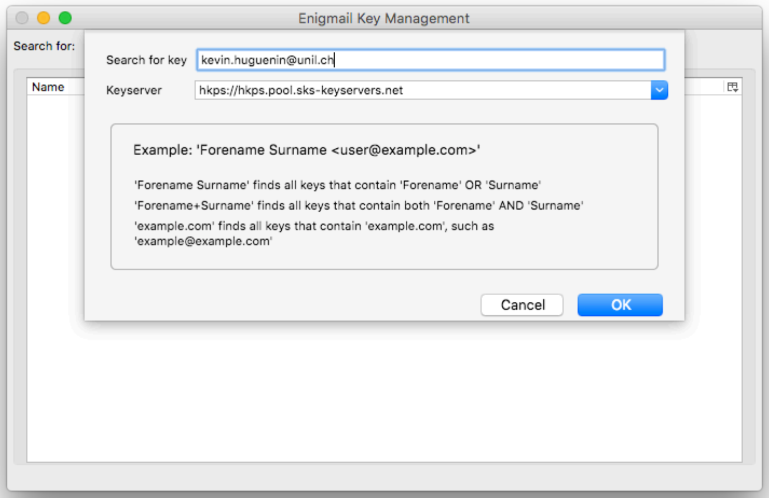
Found Keys - Select to Import

S...	Account / User ID	Created	Key ID
☑	Kévin Huguenin <kevin.huguenin@unil.ch>	2017-09-26	953F5EC5

First, you will upload your own public key to the PKI server. Select your key, right click it and select “Upload public keys to Keyserver”.


Copy Public Keys to Clipboard
Export Keys to File
Send Public Keys by Email
Upload Public Keys to Keyserver
Refresh Public Keys From Keyserver
Sign Key
Set Owner Trust
Add to Per-Recipient Rule
Disable Key
Revoke Key
Delete Key
Manage User IDs
Change Expiration Date
Change Passphrase
Generate & Save Revocation Certificate
Add Photo
View Photo ID
Key Properties

Next, import Prof. Huguenin’s key from the server Keyserver menu > Search for Keys, searching by his e-mail.


Enigmail Key Management

Search for:

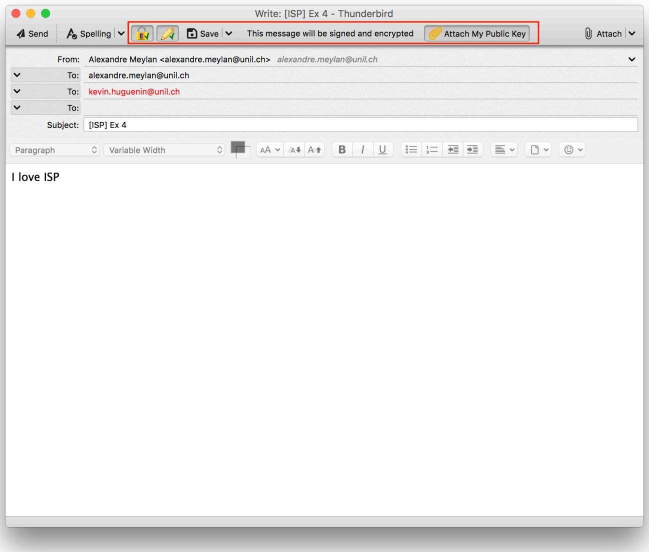
Search for key: kevin.huguenin@unil.ch

Keyserver: hkps://hkps.pool.sks-keyservers.net

Example: 'Forename Surname <user@example.com>'

'Forename Surname' finds all keys that contain 'Forename' OR 'Surname'
'Forename+Surname' finds all keys that contain both 'Forename' AND 'Surname'
'example.com' finds all keys that contain 'example.com', such as 'example@example.com'

Send an e-mail [Homework: deadline 17.10.19, 6pm]
Write a signed and encrypted e-mail (with the content of your choice) and send it to noe.zufferey@unil.ch **and** kevin.huguenin@unil.ch. To do that, in Thunderbird, create a new message, and make sure the encrypted, the sign and the attach my public key buttons are pressed. If you don't have the options to select them, you can add the Enigmail toolbar by right clicking on the top of the new e-mail window and selecting Customize... or you can go in the Enigmail menu and select what you need. You might get an error saying there is no attachment, but choose to send anyway. If you succeeded, Noé will reply to your e-mail with an encrypted answer containing a code that you will have to submit to Moodle.



This message will be signed and encrypted

From: Alexandre Meylan <alexandre.meylan@unil.ch> *alexandre.meylan@unil.ch*
To: alexandre.meylan@unil.ch
To: kevin.huguenin@unil.ch
To:
Subject: [ISP] Ex 4

I love ISP

Note that from this point on, you could easily communicate in an encrypted fashion. Downloading and sending the respective public keys only needs to happen only once for the secure communication channel to be established!

To convince yourself that this communication is indeed secure and that the e-mail server does not see these exchanges, log into your e-mail account on https://owa.unil.ch and look at the messages. You will not see any of the text body or the attachments, but only encrypted content. Note that the subject (unfortunately) and the recipient (naturally), however, are in clear.

Deliverable:
Upload on Moodle a file named `h1_IV.txt` with two lines, one for each group member. Each line will have the following format:

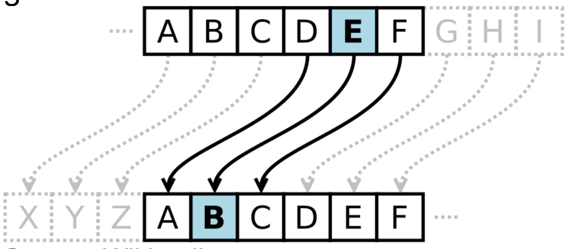
- your e-mail followed by a comma ','
- your decrypted response received from Noé

✔ Correction criteria:
0: The token is not given back **and/or** does not match yours.
1: The token is correct.

⚠ **Attention**: This is an **individual** exercise, so we need both group members to contribute with their respective received responses

Exercise 5: Caesar Cipher

The Caesar cipher, also known as a shift cipher, is one of the simplest forms of encryption. It is a substitution cipher where each letter in the original message (called the plaintext) is replaced with a letter corresponding to a certain number of letters up or down in the alphabet. In this way, a message that initially was quite readable, ends up in a form that cannot be understood at a simple glance.



Source: Wikipedia.org

For example, here's the Caesar Cipher encryption of a message, using a right shift of 3.
Plaintext:
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

Ciphertext:
QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD

Now consider that you want to decrypt the content of the `ciphertext.txt` file which you find on Moodle.

Structure your code as follows: a method to encrypt each letter of the English alphabet, a method to encrypt or decrypt a string with a given shift value and the main method that reads the ciphertext from file and performs the attack.

Question 1:

Write a Python program to decrypt the given ciphertext, using a simple version of the *frequency analysis method*, knowing that the frequency (in %) of letters in English is given in the frequencies.py file that you find on Moodle.

Your program should output the most likely value of n, which you then use to also produce the corresponding plaintext.

You need to find the shift value of "n" which makes the frequency of each letter in the cypher text as close as possible to the plaintext frequency table. For instance, in the given ciphertext, letter G has a frequency of 14.52% and letter V one of 11.52%. This indicates that these are the best candidates for the encryption of the most common letter - E (either with a shift of $n = 2$, or $n = 17$). What mathematical formula could you use to measure this distance for all the letters at the same time?

Hint:
To start with, just determine the value of n by considering only the most frequent letter.
What is the maximum number of values for n you should test in a brute force attack?

Question 2 [Homework: deadline 17.10.19, 6pm]

Write a Python program to decrypt the given ciphertext, using the *dictionary method*. In PyCharm, you can install the package `pyenchant` (pip or PyCharm packet manager) to test if a string is a word in English, as in the sample code below:

```python
import enchant
d = enchant.Dict("en_US")
candidate_word = "cookie"
print(d.check(candidate_word)) # True
candidate_word = "lskvd"
print(d.check(candidate_word)) # False
```

Your program should output the most likely value of n, which you then use to also produce the corresponding plaintext.

Hint:
The key insight is to choose the shift value n that maximizes the number of valid English words (from the available dictionary) in the candidate plaintext.

Deliverable:
Upload on Moodle a text file named `h1_V.txt`. The file should contain two lines:

- on the first line, the decrypted plaintext. Make sure the plaintext contains the original punctuation from the ciphertext.
- on the second line an integer representing the value of the shift with which the plaintext was encrypted

Also upload a Python script file, named `h1_V.py`, containing your code (for both the frequency and the dictionary method).
See the file `h1_V.txt` (available on Moodle) for an example.
This is a group exercise, so we only need one solution and one version of the code.

✓ Correction criteria:
0: The text is not deciphered **and/or** the script does not work.
0.5: The text is deciphered but **neither** the letter case **nor** the punctuation matches.
0.8: The text is deciphered, the letter case **or** the punctuation matches.
1: The text matches **exactly** to the original one.