UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Winter 2017**
Florian Kerschbaum

ASSIGNMENT 3

Assignment due date: **Friday, March 31st, 2017 3:00 pm**

Total Marks: 44
Written Response Questions TA: Nikita Volodin nikita.volodin@uwaterloo.ca
(Office hours: Tuesday 3:30 pm-4:30 pm in DC 3332)
Programming Questions TA: Navid Nasr Esfahani nnasresfahani@uwaterloo.ca
(Office hours: Friday 9:30 am-10:30 am in DC 3332)

Please use Piazza for all communication. Ask a private question if necessary. The TAs' office hours are also posted to Piazza for reference. You are expected to follow the expected Academic Integrity requirements for Assignments; you can find them here: `https://uwaterloo.ca/library/get-assignment-and-research-help/academic-integrity/academic-integrity-tutorial` Strict penalties will be enforced on students for any Academic Integrity violations.

Written Response Questions [24 marks]

Note: For written questions, please be sure to use complete, grammatically correct sentences where appropriate. You will be marked on the presentation and clarity of your answers as well as the content.

1. [8 marks total] **Two-time Pad**

   Two ciphertexts, called `ciphertext1` and `ciphertext2`, have been provided for you in your home directory (`/home/`*`USERID`*) on your assigned ugster machine. The original plaintexts are fragments of text taken from the English-language Wikipedia. These fragments have had references (notations like "[4]") removed, and contain ASCII characters only in the range from 0x20 (space) to 0x7e (tilde). In particular, there are no control characters, such as newlines, in the plaintexts. The plaintexts were then truncated to exactly 300 bytes.

   To produce the ciphertexts, a random 300-byte pad was generated, and each plaintext was encrypted *using the same pad* by XORing the plaintext and the pad.

   The plaintexts, ciphertexts, and pad are unique to you (your classmates have been given different ciphertexts generated from different plaintexts using different pads).

(a) [2 marks] What is the XOR of the two original plaintexts? Submit it as a 300-byte file called `xor`.

(b) [6 marks] Determine the two original plaintexts. (Tips: “man ascii”. You may search Wikipedia.) Submit them as two 300-byte files called `plaintext1` and `plaintext2`. Explain in detail how you got your answer. If you used or wrote any software to help you, describe how the software works.

2. [8 marks total] **GnuPG**

A GnuPG public key for nikita.volodin@uwaterloo.ca is provided along with the assignment on the course website (nikita.volodin.asc). Perform the following tasks. You can install GnuPG on your own computer, or use the version we have installed on the ugster machines.

~~(a) [2 marks] Generate a GnuPG key pair for yourself. Use the RSA and RSA algorithm option, your real name, and an email address of your-userid@uwaterloo.ca. Export this key using ASCII armor into a file called **key.asc**. [Note: older versions of GnuPG might not have the RSA and RSA algorithm option, so check that the version you are using has this option. The ugster machines have a new enough version, but the student.cs machine may not.]~~

(b) [2 marks] Use this key to sign (not local-sign) the nikita.volodin@uwaterloo.ca key. Its true fingerprint can be found on the course website (nikita.volodin.fpr). Export your signed version of the nikita.volodin key into a file called **nikita.volodin-signed.asc**; be sure to use ASCII armor. [Note: signing a key is not the same operation as signing a message.]

~~(c) [2 marks] Create a message containing your userid and name. Sign it using the key you generated, and encrypt it to the nikita.volodin key. You should do both the encryption and signature in a single operation. Make sure to use ASCII armor, and save the output in a file called **message.asc**~~.

(d) [2 marks] Briefly explain the importance of fingerprints in GnuPG. In particular, explain how users should check fingerprints and what type of attacks are possible if users do not follow this procedure properly.

3. [8 marks total] **Diffie-Hellman**

Diffie-Hellman key exchange protocol is designed to establish common secret between two parties while happening over open channel.

(a) [2 marks] Assume Alice and Bob agree to use modulus $p = 83$ and base $g = 50$. Then Alice chooses secret parameter $a = 23$ and Bob chooses secret parameter $b = 12$. What are the public values that Alice gives to Bob and Bob gives to Alice? What is the resulting secret key that is generated as a result of DH protocol?

(b) [2 marks] Assume Alice and Bob agreed to use modulus p, and base g. During the key exchange, Mallory observed these values as well as public parameters sent by Alice and Bob: $A = g^a \pmod p$ and $B = g^b \pmod p$. Can Mallory recover original secret values a or b given public values? Explain.

(c) [4 marks] If Mallory behaves as active MITM attacker, how can she abuse DH protocol to obtain all of the plain-text communications between Alice and Bob? How can this be prevented?

Programming Question [20 marks]

A hospital wants to make a database of infectious diseases in the region available to researchers. The corresponding file, `Fake_Disease_Record.csv`, is provided along with the assignment on LEARN. For every person living in Fake Region, this database contains at most one record. A record lists the person's name, gender, phone, postal code, date of birth, and diagnosis. Understanding that it is important to preserve the privacy of these records, the hospital hires you as a consultant to anonymize the database.

Your job is to anonymize the records based on k-anonymity. In general, k-anonymizing a database such that the amount of remaining information is maximized is NP-hard. Therefore, we ask you to implement two different anonymization strategies, each consisting of four stages.

- Strategy 1:
  1. In every record, the name and phone number of the person are each replaced with the * character.
  2. Based on a person's gender, postal code, and date of birth, it might be possible to re-identify the person (e.g., by correlating this information with a public polling list from a recent provincial election). k-anonymity can help here. Assume that there is a set of records that all have the same gender, postal code, and date of birth. If there are fewer than k records in this set, the day in the date of birth is replaced with the * character for each record in this set. Repeat this process for each possible set.
  3. The second stage is repeated, but this time, the month in the date of birth is replaced with the * character for all sets of records that still have fewer than k records.
  4. The second stage is repeated, but this time, the second part in the postal code, from right to left and one character/digit at a time, is replaced with the * character, for all sets of records that still have fewer than k records.
- Strategy 2:
  1. In every record, the name and phone number of the person are each replaced with the * character.
  2. Consider the sets discussed in the second step of strategy 1. However, this time while the set has fewer than k records keep replacing the the next character/digit from right up to four times. Repeat this process for each possible set.
  3. The second stage is repeated, but this time, the day in the date of birth is replaced with the * character for all sets of records that still have fewer than k records.
  4. The second stage is repeated, but this time, the month in the date of birth is replaced with the * character for all sets of records that still have fewer than k records.

Your program should calculate and the maximum value of ℓ for which the $\ell-$diversity holds, and write the value in the the first column of last row of the corresponding anonymized file.

It should also produce two files `<input_file>_anonymized1.csv` (for strategy 1) and `<input_file>_anonymized2.csv` (for strategy 2), containing the anonymized records. Furthermore, you should produce a second pair of files, `<input_file>_non-kanonymized1.csv` and `<input_file>_non-kanonymized2.csv`, containing all the records from the original database for which k-anonymity does not hold after the anonymization with each strategy. When generating these two files, include column labels (as in the examples), and do not reorder records.

Testing and Marking

You may use any programming or scripting language available on the ugster machines. If your preferred language is not available, we may be able to accommodate requests. For evaluation, this file will be extracted and we will attempt to run a script at the top level with `./kanon` *`k input_filename`*. This script could be your program itself, or it could be a script that compiles and then runs your program. In either case it should start with an appropriate shebang line. Your submission should not contain any compiled executables. The invocation of your program will be in the following format:

```
./kanon k input_filename
```

1. Your submission files will be extracted into the (initially empty) current working directory.
2. `kanon k Fake_Disease_Record.csv` will be run.
3. Your produced files will be verified.
4. The previous two steps will be repeated on different test input files and values of k.

- You may assume k is an integer such that $1 \leq k \leq 100$.
- `<input_filename>` is a properly formatted csv file with identical structure, headers, and column order as the sample `Fake_Disease_Record.csv`
- To allow for partial marks, your code should write output after each stage of anonymization:
  - `<input_file>_anonymizedx.csv.1` — Output after first stage of anonymization using strategy `x`.
  - `<input_file>_anonymizedx.csv.2` — Output after second stage of anonymization using strategy `x`.

- `<input_file>_anonymizedx.csv.3` — Output after third stage of anonymization using strategy `x`.
- `<input_file>_anonymizedx.csv` — Output after final stage of anonymization using strategy `x`.
- `<input_file>_non-kanonymizedx.csv` — The *complete* records for which k-anonymity does not hold using strategy `x`.

Example ($k = 2$)

Assume the database file `test` consists of the following four records (plus header):

Name,Gender,Date of birth,Telephone,Postal code,Disease
Alice Alisson,F,1-11-1911,519 123 3435,N2L 3G1,Yellow fever
Bob Bobson,M,4-12-1945,519 537 8756,N2L 4F2,Mumps
Carol Carson,F,8-11,1911,519 635 5285,N2L 3G1,Conjunctivitis
Dave Davidson,M,3-4-1978,519 274 4345,N2L 3G0,Hamburger disease

Then `test_anonymized1.csv` would look as follows ($k = 2$):

Name,Gender,Date of birth,Telephone,Postal code,Disease
$*$,F,$*$-11-1911,$*$,N2L 3G1,Yellow fever
$*$,M,$*$-$*$-1945,$*$,N2L $* * *$,Mumps
$*$,F,$*$-11-1911,$*$,N2L 3G1,Conjunctivitis
$*$,M,$*$-$*$-1978,$*$,N2L $* * *$,Hamburger disease
1

Note that the resulting database is not strictly k-anonymous. Namely, there is only one record with year 1945. Same for 1978. Therefore, `test_non-kanonymized1.csv` would look as follows:

Name,Gender,Date of birth,Telephone,Postal code,Disease
Bob Bobson,M,4-12-1945,519 537 8756,N2L 4F2,Mumps
Dave Davidson,M,3-4-1978,519 274 4345,N2L 3G0,Hamburger disease

Using the second strategy will generated `test_anonymized2.csv`, which would look as follows ($k = 2$):

Name,Gender,Date of birth,Telephone,Postal code,Disease
$*$,F,$*$-11-1911,$*$,N2$* * * *$,Yellow fever
$*$,M,$*$-$*$-1945,$*$,N2$* * * *$,Mumps

```
*,F,*-11-1911,*,N2****,Conjunctivitis
*,M,*-*-1978,*,N2****,Hamburger disease
1
```

, and `test_non-kanonymized2.csv` would be identical to `test_non-kanonymized1.csv`. Note that in both cases $\ell = 1$.

What to hand in

Using the “submit” facility on the student.cs machines (**not** the ugster machines or the UML virtual environment), hand in the following files:

a3.pdf: A PDF file containing your answers for all written questions. It must contain, at the top of the first page, your name, UW userid, and student number. **-3 marks if it doesn’t!** Be sure to “embed all fonts” into your PDF file. Some students’ files were unreadable in the past; if we can’t read it, we can’t mark it.

kanon.tar : A tarball containing your source code for the programming portion, and a Makefile (if necessary). To create the tarball, `cd` to the directory containing your code, and run the command

```
tar cvf kanon.tar .
```

(including the `.`).