UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Fall 2018**

Ian Goldberg
Florian Kerschbaum

ASSIGNMENT 3

Assignment due date: **Monday, December 3rd, 2018 4:00 pm**

Total Marks: 53
Written Response Questions TA: Bailey Kacsmar bkacsmar@uwaterloo.ca
(Office hours: Mondays 11:00 am–12:00 pm in DC 3333)
Programming Questions TA: Sajin Sasy ssasy@uwaterloo.ca
(Office hours: Friday 11:00 am–12:00 pm in DC 3333)

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 [27 marks]

Note: Please ensure that written questions are answered using complete and grammatically correct sentences. You will be marked on the presentation and clarity of your answers as well as on the correctness of your answers.

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

   Two ciphertexts, called `ciphertext1` and `ciphertext2`, have been provided for you through infodist. 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] **Diffie-Hellman**

   Diffie-Hellman (DH) key exchange allows Alice and Bob to exchange a secret key using an insecure channel. See `http://mathworld.wolfram.com/Diffie-HellmanProtocol.html` for details.

   Note: For the purpose of this assignment, the parameters chosen for this question are very small. Real Diffie-Hellman would use much larger numbers.

   (a) [2 marks] Assume Alice and Bob agree to use modulus $p = 83$ and base $g = 5$. Then Alice chooses secret parameter $a = 23$ and Bob chooses secret parameter $b = 58$. 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) [3 marks] Assume that Alice and Bob agree to use modulus p, and base g. During the key exchange, Eve observes these values as well as the public parameters sent by Alice and Bob: $A = g^a \pmod{p}$ and $B = g^b \pmod{p}$. Can Eve recover the original secret values a or b given public values? **Explain**.

(c) [3 marks] If Mallory behaves as an active Man-In-The-Middle (MITM) attacker, how can she manipulate the DH protocol to obtain all of the plaintext communications between Alice and Bob? How can this be prevented? Explain.

3. [5 marks total] **Variants of RSA**

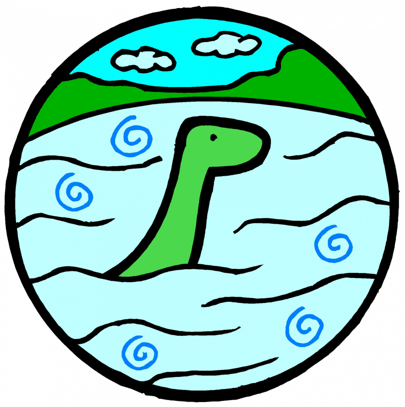
In RSA, the public key is a pair of integers (n, e), where $n = pq$ for large primes p and q. The private key is the triple (p, q, d) where $de \equiv 1 \mod (p-1)(q-1)$. In a simplified form of RSA, called "textbook RSA", the encryption of a message m to yield the ciphertext c is $c = m^e \mod n$ and the decryption is $m \equiv c^d \mod n$.

A key part of making RSA secure is to choose appropriate primes numbers. It is often stated that textbook RSA is weak because there are no restrictions on how to choose these primes (and for a number of other reasons as well). In this problem, we will investigate a variation of RSA that puts restrictions on the primes p and q.

Let us assume the genius security expert, Nessie, thinks that the following way of generating prime would be a good idea: choose the numbers g, b, and a such that $p = 2ga + 1$ and $q = 2gb + 1$ are prime. Nessie first puts an implicit restriction that the greatest common divisor of a and b is 1 (greatest common divisor of two integers is the largest integer that divides both the numbers).

She also adds the restriction that the term $2gab + a + b$ should be prime (in retrospect, this decision proves important for the security of the encryption scheme). Much like in textbook RSA, she then sets $n = pq$. During a round of beer, she boasts to her arch enemy Kingstie that her scheme is secure (that it is impossible to factor n into primes p and q in polynomial time).

Kingstie never took this (or any security) course and blindly believes that Nessie has devised an "unbreakable" encryption scheme. To impress another security expert, Manipogo, Kingstie gets involved in a bet that this scheme is unbreakable, even if given the values of a and b. Manipogo is intelligent, so after seeing the encryption scheme closely, she asks Kingstie to give her any two values of a and b that fit the scheme. Blinded by incomplete knowledge, the Kingstie gives Manipogo the values of a and b.



We will see how Manipogo can factor n in time that is polynomial in the bit length of n and win the bet.

(a) [2 marks] Write the number n in terms of g, a and b.

(b) [2 marks] Use the above relation to give a closed-form expression for g.

(c) [1 marks] Write in one sentence, how the above information helps you in finding the factors of n.

4. [6 marks total] **Inference Attacks**
The University Human Resource has a table of size N, `Employee`, in its database. Below is an excerpt (not the entire table):

Name	Gender	Occupation	Postal Code	Salary
James	M	Department Head	A0D 1Y7	150,000
Miguel	M	Instructor	B7S 7S8	55,325
Christine	F	Secetary Worker	G3R 9N1	50,721
Nicholas	M	Dean of Faculty	V3K 5H2	225,000
Stacey	F	Assistant Professor	Q9D 7D9	91,426
Cori	F	Administrative Officer	N7E 8P4	73,908
Matthew	M	Custodian	U4G 6W8	30,923
Natalie	F	Career Advisor	N2R 5Y7	58,129
Rachel	F	Student	P0D 5U8	12,907

Table 1: Part of `Employee` Table

To deter people learning other people's salary, the database is set up to suppress the `Salary` field in the output of queries. However, users can execute queries of the form `SELECT SUM(Salary) FROM Employee WHERE ...` where queries that match fewer than k or more than $N-k$ (but not all N) records are rejected. We will use $k = \frac{N}{8}$.

(a) [3 marks] Use a tracker attack, as defined in class, to design a tracker and a set of three queries based on this tracker that will let you infer Cori's salary. Both the tracker and the three queries need to be of the form `SELECT SUM(Salary) FROM Employee ...` Assume that employee's names are unique and not known to the attacker (apart from Cori's) and that the attacker has no additional information about Cori (not even her gender). For the tracker attack to succeed, the attacker does need to make an assumption about the distribution of (some of) the values stored in the database. This assumption should be realistic so that your tracker attack (likely) would also work on a different set of records, not only on the set shown above. In your solution, you should give 1) your assumption, 2) your tracker, and 3) the set of three queries.

(b) [3 marks] The University Human Resource becomes aware of the tracker attack and forbids `SUM(Salary)` queries. Instead the University Human Resource allows only queries of the form `SELECT COUNT(*) FROM Employee WHERE Q`. Note that `Q` may include testing the value of the Salary field. (Again, queries that match fewer than k or more than $N-k$, with the exception of N, records are ruled out.) How would you use queries of this type to learn Rachel's salary? You may assume that no one in the database makes more than $1 million and that all salaries are non-negative if that simplifies your attack.

Programming Questions [26 marks]

Background

With the large volumes of data we deal with today, hosting one's databases on cloud servers is a normal practice. However, maintaining the security of one's database hosted on such untrusted servers, while achieving functionality of queries over it, intuitively seem like orthogonal goals. Encrypting a database naively strips away querying functionality. For instance in the naive case, to query for a particular keyword in an encrypted database, one has to decrypt the entire database and then search for the keyword. In order to attain certain classes of querying functionality while preserving security, a technique used in practice is to leverage *property-preserving encryption* schemes. Securing databases through encryption while preserving the ability to perform query functionality on it is an area of active research today.

Hackers from another universe DON.MAC and Mr. Slippery are engaged in a serious debate about this very same conundrum of encrypting databases. DON.MAC proudly claims that he leverages two particular property-preserving schemes that can thwart any adversary that tries to breach the confidentiality of his data, while still retaining his querying functionality. The first is a *deterministic encryption* scheme; a deterministic encryption scheme is one for which any fixed key always produces the same ciphertext for a given input plaintext to the encryption function. DON.MAC uses a 128-bit AES-ECB as the particular instantiation of his deterministic encryption. The second is an *order-preserving* encryption (OPE) scheme; the ciphertexts of such a scheme retain the order properties of the corresponding input plaintexts (which must be, for example, numbers); i.e., if x and y are two plaintext numbers with $x < y$, then the order-preserving encryptions of x and y ($E(x)$ and $E(y)$ respectively) will be numbers such that $E(x) < E(y)$. Note that order-preserving encryption is also deterministic: equal plaintexts will always produce equal ciphertexts.

Mr. Slippery on the other hand is very skeptical about the security of such property-preserving encryption schemes, and insists that DON.MAC is in fact vulnerable to several *leakage attacks* of such encryption schemes. In order to prove his point, Mr. Slippery persuaded DON.MAC to let him attempt an attack on DON.MAC's encrypted data. Mr. Slippery's only requested resource in addition to the encrypted data, is some background knowledge about the data for the deterministic encryption scheme. Being the cautious hacker that he is, DON.MAC did not want to let Mr. Slippery near his own private data, hence he created a set of challenges for Mr. Slippery by generating encrypted datasets and corresponding background knowledge files from a publicly available dataset,[1] for Mr. Slippery to demonstrate his attack on.

[1] The Texas Inpatient Public Use Data records, which are anonymized and made publicly available by Texas Health Care Information Collection Center for Health Statistics.

Help Mr. Slippery and DON.MAC evaluate the security of property-preserving encryption schemes over the following questions:

Questions

1. [8 marks] For his first challenge, DON.MAC provides Mr. Slippery with an encrypted dataset (**p1_encrypted.csv**), and a background knowledge file (**p1_background.csv**). As with the distribution of letters and bigrams discussed in class, this background knowledge has a similar distribution of keywords as the encrypted dataset, because it was drawn from a similar source. Help Mr. Slippery implement an attack that can extract the ciphertext-to-plaintext mapping for the encrypted data.

2. [10 marks] To perform range queries on his encrypted data, DON.MAC uses a home-brewed order-preserving encryption scheme with the following two functions:

$$\text{Encryption}(i, (s, t)) = i \cdot s + t = \bar{i}$$
$$\text{Decryption}(\bar{i}, (s, t)) = (\bar{i} - t)/s$$

   - i: an input integer in the domain $0 \leq i < 2^{32}$
   - $\bar{i}$: the encryption of integer i
   - (s, t): the secret key of the order-preserving encryption scheme, with $s > 0$, and $0 \leq t < s$.

   DON.MAC generates a new key (s, t) for his aforementioned OPE scheme and provides Mr. Slippery with a file of encryptions of random inputs (**p2_encrypted**). Help Mr. Slippery attack DON.MAC's scheme by producing a file of corresponding plaintexts for DON.MAC's encrypted file.

3. [8 marks] After having successfully broken DON.MACs futile attempts at securing his data, Mr. Slippery informed DON.MAC about secure order-preserving encryption schemes that he uses for his databases; the order-preserving encryption scheme that the cryptographers of Earth have been working on,[2] and strongly recommended him to use that instead of the home-brewed insecure scheme. As DON.MAC was tinkering with his new-found tool, it suddenly struck him that even this scheme had its own leakages. In order to make his point to Mr. Slippery, DON.MAC used this secure order-preserving encryption scheme to encrypt a sampling of the black (x,y) pixel coordinates of a secret black-and-white image. (Hint: the image will be of a short sequence of alphanumeric characters.) No x-coordinate will be sampled twice, and similarly for the y-coordinates. Since the coordinates are unique, they are not susceptible to the earlier attacks demonstrated by Mr. Slippery (as in Question 1), and they also use an underlying secure order-preserving scheme hence aren't

[2] https://eprint.iacr.org/2012/624.pdf

susceptible to attacks against the encryption scheme itself (as in Question 2). However DON.MAC claims that it is still vulnerable to a form of information leakage, and hence provided Mr. Slippery with this OPE-encrypted dataset of coordinates. Help Mr. Slippery extract the image content through the "information leak" of this secure OPE scheme. You will be provided (through infodist) a file of OPE-encrypted (x,y) coordinates. You are expected to provide the decrypted secret (the alphanumeric characters) for this question in your written response, as well as a description of how you determined your answer. You do not need to submit your code for this question.

Output Format

For Part 1, each line of your `output_file` should be a .csv format file[3] with each line containing a unique ciphertext from the input `encrypted_file` and its corresponding plaintext. The ordering of (ciphertext,plaintext) pairs do not matter; the marking script will handle that.

For Part 2, each line of your `output_file` should be just the plaintext corresponding to the ciphertext in that position in the input `encrypted_file`.

Testing and Marking

Your hand in programs must be called **p1_attack** and **p2_attack** respectively. They will be invoked in the following manner:

./**p1_attack** *background_file encrypted_file output_file*

./**p2_attack** *encrypted_file output_file*

For marking, we will compile and execute your attacks in a virtual machine with no access to the Internet. The *background_file* and *encrypted_file* will be placed in the same folder as your code when we execute it, and your code is expected to produce the output in *output_file* in the same directory. Note that for Question 1 we will run your program against input files that will have a different number of records and columns than the sample dataset provided to you, and your program is expected to be able to handle that. Similarly for Question 2 we will provide input files with varying numbers of input ciphertexts. The following steps will be performed:

1. Your submission files will be extracted into an initially empty working directory.

[3] To be clear, you do not have to add .csv to the file name provided in the command line, just the format of your output should be that of a standard .csv file

2. If `Makefile` is found, then `make` will be executed to compile your code.
3. Depending on the attack being evaluated, we will execute either of the following:
   - ./**p1_attack** *background_file encrypted_file output_file*
   - ./**p2_attack** *encrypted_file output_file*
4. The output of step 3 will be compared to the expected output for the test case
5. If there are more tests to run, go to step 3. Otherwise, derive a mark based on the program outputs.

At some point before the assignment deadline, we will provide you with access to a system where you can submit your files to ensure that they will compile successfully in the marking environment. We will make an announcement through LEARN and/or email when this system becomes available. You will be expected to ensure that your code compiles in this environment before your final submission.

Programming Languages

You may implement your solution in several of the most popular programming languages. You may choose any of the languages supported by our marking system to use for your attack implementations.

The marking system runs Ubuntu 14.04; your submission must operate in this environment. The following table enumerates the supported languages and, for interpreted languages, the shebang (the line starting with `#!`) that you should include as the first line of your source file:[4]

Language	**Version**	**Shebang**
C	gcc 4.8.4	*(Makefile)*
C++	g++ 4.8.4	*(Makefile)*
Java	1.7.0 (OpenJDK)	*(Makefile)*
Go	1.2.1	*(Makefile)*
Perl	5.18.2	`#!/usr/bin/env perl`
PHP	5.5.9	`#!/usr/bin/env php`
Python	2.7.6	`#!/usr/bin/env python`
Python3	3.4.3	`#!/usr/bin/env python3`
Ruby	1.9.3	`#!/usr/bin/env ruby`
Tcl	8.6	`#!/usr/bin/env tclsh`

[4]Make sure that the shebang line doesn't have trailing spaces at the end.

Special Notes for Programming Languages: If you choose to implement your solution using any of the following languages, you should consider the special notes:

- **Go:** Your submission is extracted into the `$GOPATH`. Consequently, you will need to have `bin`, `pkg`, and `src` subdirectories inside your submission. However, you will still need to include a `Makefile` that copies your final executables to `$GOPATH/p1_attack` and `$GOPATH/p2_attack`.

What to hand in

All assignment submission takes place on the `student.cs` machines (not ugster or the virtual environments), using the `submit` facility. In particular, log in to the Linux student environment (`linux.student.cs.uwaterloo.ca`), go to the directory that contains your solution, and submit using the following command: `submit cs458 3 .` (dot included). CS 658 students should also use this command and ignore the warning message. Hand in the following files:

a3.pdf: A PDF file containing your answers for all written questions and programming questions when requested. 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.

xor, plaintext1, plaintext2: For the first question of the written part of the assignment.

src.tar: Your source files for your attacks, in your supported language of choice, inside a tarball. To create the tarball, `cd` to the directory containing your code, and run the command

```
tar cvf src.tar .
```

(including the `.`). If you are using an interpreted language, your source should include executable scripts named `p1_attack` and `p2_attack` that runs your code using the proper shebang. For compiled languages, include a `Makefile` in your source with a default target that builds executables named `p1_attack` and `p2_attack`.