CSE 484 / M584 Lab 2:
Cryptography Did Not Solve My Problem :(

1 Structure

Due Dates

- **All components**: Wednesday April 30th, 11:59pm

Handin All parts on Gradescope, see Deliverables. There are 2 required assignments + 2 extra credit ones.

Groups

- **Programming parts:** Partners or individual
- **Programming questions + Short answers:** Individual

Grading

All written up components (short answers and the programming short answers) MUST be done individually: do not collaborate on the writeups.

Points: 37 + EC

- 5 per programming problem:
  - 2-does-not-work
  - dont-count-on-counters
  - goldmine-of-ps-and-qs
- 1 per programming problem question (total of 8)
- Varies per short answer question (total of 14)
- 7 for Extra Credit (confession-by-compression: 3, written problems: 1 each)

2 Formatting Notes for this Handout

- Programs, files, or tools are stylized like `this`.
- Commands for programs are stylized like `this`.
- Code, assembly, and strings are stylized like `this`.

3 Before you start

You can work on either `umnak.cs.washington.edu` or on a local machine that has python3 installed. We can provide some assistance with a local setup, but if you have any problems just work on `umnak`.

4 Overview

- The goal of this assignment is to get a hands-on feel for how various cryptographic constructions do and don't work.
- There are 3 required programming problems and 1 extra credit.
- Each programming problem has several discussion questions.
- Finally, there are a series of short answer questions with no associated programming component.

4.1 Setup

First, you will need to make a fork of the lab2 gitlab:
https://gitlab.cs.washington.edu/dkohlbre/crypto-lab-25sp
Please make this fork *private* so that other students don't find it! Share this fork with your partner if you have one. Then clone your fork to wherever you are working (we encourage umnak.)
Once you have done that, you will need to run the `setup.sh` script in the `setup/` directory to setup the python environment.

Testing You can test a simulation of the autograder with `./test_locally.sh`. You need the autograder on Gradescope (using different keys) to pass for credit.
If you want to directly call `python` for testing your scripts (e.g. `python des_cracker.py`), you will need to run `source python-venv/bin/activate` once per shell.

4.2 Deliverables

Required:

- Python scripts for 2-does-not-work, dont-count-on-counters, and goldmine-of-ps-and-qs. These must pass the Gradescope autograder.
- Short answers to all programming problem questions, see Gradescope.
- Other short answer questions, see Gradescope.

Extra Credit:

- Python script for confession-by-compression, must pass autograder.
- Short answer questions for compression-by-confession.
- Short answer extension questions for other programming problems.

5 Programming problems

Each problem is unique, but has the same setup for you to work from: a python skeleton and a set of local tests you can run with `./test_locally.sh`.

Each problem has a detailed README.md that explains the specifics of the problem and any notable other bits. You will need to read those READMEs to solve the problems.

Our solution for each problem is <20 lines of code. If yours is significantly longer, you may want to rethink your approach.

2-does-not-work Your friend is using two chained DES operations with different keys, can you recover their keys from only seeing a single ciphertext and plaintext pair?
NOTE: To make this problem reasonable to run, we'll use DES with (effective) 14-bit keys. See the README for details.

dont-count-on-counters A server lets you download books, but the encryption and protocol it uses is not well thought out *and* they have a problem with how they generate "random" initialization vectors (IVs). Given only the ciphertexts of someone downloading a book, can you recover some of the plaintext of the book?

goldmine-of-ps-and-qs Not everyone generates RSA keys the way they should, and (true story!) sometimes there are problems with real devices where they can choose *partially* the same RSA keys. Given many *public* keys generated in this way, can you recover some of the *secret*(private) keys?

Extra Credit: confession-by-compression Encryption and compression don't play nicely together. When a message is first compressed, and *then* encrypted, something surprising can be revealed. If you get to insert some content into that message, can you recover the rest of the message?

6 Short answer questions

All written components are to be done individually.

Misc

1. There are several written questions for this lab, which are unrelated to the programming problems. See the Gradescope assignment for the questions.

2-does-not-work

1. What is the problem with 2DES?
2. At most, how many total 2DES operations (encryptions + decryptions) are needed to recover the original keys of 2DES? Assume keys are 14 bits.
3. Why does 3DES solve the vulnerability in 2DES?

dont-count-on-counters

1. What was the book in the test case we gave you?
2. How were you able to decipher the plaintext that the server sent to Alice?
3. Why couldn't we decode **all** content in each ciphertext that the server sent to Alice?

goldmine-of-ps-and-qs

1. How many RSA keys did you factor for the example we gave?
2. How did you factor these keys efficiently, and why did it work?

7 Extra Credit questions

All questions here are bonus, and should be submitted to the Lab2 Extra Credit Gradescope component.

Extra credit: dont-count-on-counters extension

Let's say that Alice no longer sends ready messages to the server, but you come across the following ciphertexts from the server (values are also in the README):
`95afdf64ac58d516626107be975c6b5caad9da1cb3971310b9c93ff1785a2f8a1b4539ce59b5a618a4d31c89a21de2b443dff3c45e42c93749aa81873ac1f0135f7923b69cb0a3a76849c93ab678d1a8cf20dfe68268bd5ba761228cb4205802c3b3a33cc89cc19509c769ef0cc4cb7f01101b9cb4e3521d86a0d4d55f133e44`
and
`95afdf64ac58d516626107be975c6b5ca9df8e5aab8d5209fb963fcc3052318a034525c41cbdb31ea5d30a82b900b6f90db2e49c1b63d13748bc9d8a36dab910557d77f087aaeca6204ec829e53bcbacd12a8bf7c42dd40ca367208cb4235f43dcb7eb708988dac705c179ab0ecb866965510edfe6a27116c4b595ff52503a15b1f55bd7e72477e1b60a9b34f5143078899b80f76f00247334e67e3f9858abfb`
Given your knowledge of the book and AES CTR mode, what were the plaintexts?

Extra credit: goldmine-of-ps-and-qs extension

- What is the expected number of keys you will be able to factor, if you are given 300 keys generated as described in the problem statement?
- How did you compute that?
  - Assume that when generating keys we ensure that all public keys are distinct and no key is a perfect square.
  - You can have very small inaccuracies if you recognize them in your answer and justify why they are very small.
  - Use double dollar-signs to enter equations into Gradescope.

Extra credit: confession-by-compression

1. How did you recover the password efficiently? Why did it work?

2. Briefly describe one real-world compression side channel attack (2–3 sentences). Please include a link to your source.
   Any source can work (e.g. news article, research paper, Wikipedia), as long as it is detailed enough to understand how the attacker got access to the side channel and how they used the information from the side-channel.

8 Miscellaneous

8.1 Credits

This project was originally designed by Sela Navot, Evan Lam, Nirvan Tyagi and David Kohlbrenner.