Weaver Fall 2020 — CS 161 Computer Security — Optional Lab 2

Due: Friday, December 11, 2020, 11:59 PM PT

Most recent update: November 15, 2020

Starter files

Based on a true story

One of Nick's greatest failures as an instructor was when one of his former CS161 students implemented an encryption scheme to distribute online exams for a Berkeley CS class. Even after the evil horror of project 2, they tried to write their own crypto...

In this lab, you will decrypt Python code snippets that were encrypted insecurely. Collaboration is allowed and encouraged, but please credit all collaborators and write up your final submission in your own words.

You can use your overall homework grade to replace your grade on this lab, so it is effectively optional. However, EvanBot is offering a small incentive to participate in this hivemind data reconstruction effort: if the class can collectively create an algorithm that perfectly decrypts the entire encrypted exam, EvanBot will give 5 points of extra credit to everyone on the final exam.

Exam format

The day before the exam, each student would receive a set of encrypted questions. Then, when the exam started they would receive the decryption key for their first question. After time expired for question 1, they would receive the decryption key for question 2. After time expired for question 2, they would receive the decryption key for question 3, and so on.

To make it harder to cheat, each student received the questions in random order, and each question could have different versions with different keywords or variable names.

To make distributing keys easier, everyone's nth question would be encrypted with the same key, even though different students received different questions as their nth question.

As an example, consider the following scenario with 2 students and a 3-question exam:

Student	Question 1	Question 2	Question 3
Alice	Enc(k_1, Trees Version A)	Enc(k_2, Recursion)	Enc(k_3, Linked lists)
Bob	Enc(k_1, Recursion)	Enc(k_2, Linked lists)	Enc(k_3, Trees Version B)

Each student receives the questions in a random order, but everyone's first question is encrypted with k_1, everyone's second question is encrypted with k_2, and everyone's third question is encrypted with k_3. Also, notice that Alice receives Version A of the Trees question, and Bob receives Version B of the Trees question.

Encryption format

To encrypt the questions, course staff imported the `pycrypto` library and used it to build a simple encryption program, `encrypt.py`, that accepts a key and a series of files and either encrypts or decrypts them as appropriate. Here is the encryption function:

```
def encrypt(name):
    f = open(name, mode='r')
    data = f.read()
    ctr = Counter.new(128)
    cipher = AES.new(key, AES.MODE_CTR, counter=ctr)
    out = open(outname, mode='w')
    out.write(cipher.encrypt(data))
    f.close()
    out.close()
```

It simply uses CTR mode to encrypt, following the instructions from the python library. Unfortunately this library has an, umm, limitation: you don't need to specify an IV or initial counter state. Now the person coding the exam-distribution program through this would mean it would simply chose a random IV and prepend it to the message.

Unfortunately it was the opposite: if you failed to specify an IV or counter start they would start at 0. This means that every question on the exam was encrypted with a constant IV of 0.

Question format

In cryptanalysis, a powerful strategy is using information you know about the structure of the encrypted text to narrow down the possibilities.

The exam questions are small fill-in-the-blank Python programs with some portions of the code replaced with 6 underscores (`_____`). The Python programs are all written in English with the standard character set (no special characters or emojis).

Each file starts with this sequence of characters:

```
email = '{EMAIL-address}'

def
```

Alice's email address is `alice@alice.com`, and Bob's email address is `bob@robert.com`.

Your task

First, download the Starter files if you haven't already.

We provide the following files:

- `encrypt.py`: the Python script used to encrypt files
- `q{N}.{STUDENT}.py.out` (8 files): Two encrypted 4-question exams. `{N}` is the question number (1 to 4) and `{STUDENT}` is the name of the student (Alice or Bob).
- `encrypt.py.out`: the result of encrypting the `encrypt.py` script
- `encrypt.py.out.decrypt`: an example of a partially decrypted `encrypt.py.out`. Note that there are some Xs where the decryption algorithm couldn't predict the decryption.

Your goal is to decrypt as much of the exam as possible. Note that for some questions you won't be able to decrypt much, but others will be effectively fully decryptable. You do not need to decrypt everything for full credit on this assignment.

This assignment is purposely very open-ended: you can use any programming language you want to write up your solution. You can also use any cryptanalysis tools you find online, and share and discuss your ideas with others in the class. The only thing you may not do is ask someone outside the class to decrypt the exam for you (e.g. posting this assignment on Chegg or Stack Overflow).

Submission

On Gradescope, submit your 8 decrypted exam questions to the Lab 2 Autograder. Make sure to name your files `q{N}.{STUDENT}.py.out.decrypt`, replacing `{N}` and `{STUDENT}` appropriately.

If you used any code in decrypting, submit your code to the Lab 2 Code assignment. Please include a README with a high-level description of the strategies you used in decrypting. Make sure to credit anything that is not your original work. These will be graded on completion–since it's a new assignment, we'd like to get an idea of what decryption strategies people are using.

Finally, since the assignment is new, we'd love to hear any feedback you have for how to improve the assignment for future semesters. Please submit any comments you have about the lab in the README. Your feedback will not affect your grade in any way.