Problem Set 2: Crypto

due by noon on Thu 9/22

Per the directions at this document's end, submitting this problem set involves submitting source code via `submit50` as well as filling out a Web-based, which may take a few minutes, so best not to wait until the very last minute, lest you spend a late day unnecessarily.

Be sure that your code is thoroughly commented
to such an extent that lines' functionality is apparent from comments alone.

Goals.

- Better acquaint you with functions and libraries.
- Allow you to dabble in cryptanalysis.
- Introduce you a bit early, perhaps, to file I/O.

Recommended Reading.

- Sections 11 – 14 and 39 of `http://www.howstuffworks.com/c.htm`.
- Chapters 7, 8, and 10 of *Programming in C*.

diff hacker2.pdf hacker2.pdf.

- Hacker Edition challenges you to crack actual passwords.

Academic Honesty.

All work that you do toward fulfillment of this course's expectations must be your own unless collaboration is explicitly allowed in writing by the course's instructor. Collaboration in the completion of problem sets is not permitted unless otherwise stated by some problem set's specification.

Viewing or copying another individual's work (even if left by a printer, stored in an executable directory, or accidentally shared in the course's virtual terminal room) or lifting material from a book, website, or other source—even in part—and presenting it as your own constitutes academic dishonesty, as does showing or giving your work, even in part, to another student or soliciting the work of another individual. Similarly is dual submission academic dishonesty: you may not submit the same or similar work to this course that you have submitted or will submit to another. Nor may you provide or make available solutions to problem sets to individuals who take or may take this course in the future. Moreover, submission of any work that you intend to use outside of the course (*e.g.*, for a job) must be approved by the staff.

You are welcome to discuss the course's material with others in order to better understand it. You may even discuss problem sets with classmates, but you may not share code. In other words, you may communicate with classmates in English, but you may not communicate in, say, C. If in doubt as to the appropriateness of some discussion, contact the course's instructor.

You may turn to the Web for instruction beyond the course's lectures and sections, for references, and for solutions to technical difficulties, but not for outright solutions to problems on problem sets or your own final project. However, failure to cite (as with comments) the origin of any code or technique that you do discover outside of the course's lectures and sections (even while respecting these constraints) and then integrate into your own work may be considered academic dishonesty.

All forms of academic dishonesty are dealt with harshly. If the course refers some matter to the Administrative Board and the outcome for some student is *Admonish*, *Probation*, *Requirement to Withdraw*, or *Recommendation to Dismiss*, the course reserves the right to impose local sanctions on top of that outcome for that student that may include, but not be limited to, a failing grade for work submitted or for the course itself.

Grades.

Your work on this problem set will be evaluated along four axes primarily.

Scope. To what extent does your code implement the features required by our specification?
Correctness. To what extent is your code consistent with our specifications and free of bugs?
Design. To what extent is your code written well (*i.e.*, clearly, efficiently, elegantly, and/or logically)?
Style. To what extent is your code readable (*i.e.*, commented and indented with variables aptly named)?

All students, whether taking the course Pass/Fail or for a letter grade, must ordinarily submit this and all other problem sets to be eligible for a passing grade (*i.e.*, Pass or A to D–) unless granted an exception in writing by the course's instructor.

Help!

☐ Surf on over to

`http://help.cs50.net/`

and log in if prompted. Then take a look around!

Henceforth, consider `help.cs50.net` *the* place to turn to anytime you have questions. Not only can you post questions of your own, you can also search for or browse answers to questions already asked by others.

It is expected, of course, that you respect the course's policies on academic honesty. Posting snippets of code about which you have questions is generally fine. Posting entire programs, even if broken, is definitely not. If in doubt, simply flag your discussion as "private," particularly if you need to show us most or all of your code. But the more questions you ask publicly, the more others will benefit as well!

Lest you feel uncomfortable posting, know that students' posts to the course's bulletin board are anonymized. Only the staff, not fellow students, will know who you are. Certainly don't hesitate to post a question because you think that it's "dumb." It is not!

Sanity Check.

☐ You should already have the CS50 Appliance installed, per Problem Set 1. But be sure that you have version 2.3. To check, launch VirtualBox (as by double-clicking its icon wherever it's installed), and in VirtualBox's lefthand menu should be **CS50 Appliance 2.3**. If you instead see an older version (*e.g.*, **CS50 Appliance 2.1**), head to `https://manual.cs50.net/FAQs` for instructions on how to upgrade to version 2.3.

Tips.

☐ If you have a computer that's a few years old or a netbook (in which case your CPU might be on the slower side and your RAM on the lower side), head to `https://manual.cs50.net/FAQs` for tips on how to improve the appliance's performance if you're finding it slow. If you're finding that the appliance is too slow to even be usable on your computer, email `sysadmins@cs50.net` to inquire about options.

☐ Realize that the CS50 Appliance is a computer, albeit a virtual one. For better or for worse (mostly worse), computers don't like to be forcibly shut down or otherwise interrupted while in the middle of something. Do take care, then, not to quit VirtualBox, shutdown your own computer, or even close your laptop's lid while the appliance is in the middle of something (*e.g.*, downloading or installing updates, submitting your work*, etc.*) Best to wait until the appliance isn't doing anything important, then shut it down (as via the green icon in the appliance's bottom-right corner). Bad things can happen, too, if your own computer runs out of disk space, so beware downloading big files on your own computer if you know you're low on disk space while the appliance is running.

☐ When running, the CS50 Appliance "borrows" some of your computer's own RAM and CPU cycles, which can slow down programs on your computer and vice versa. For maximum performance, try to launch VirtualBox and the appliance before launching other programs on your computer, and try to minimize the number of programs running on your computer while the appliance is running.

With that said, if you have lots of RAM (*e.g.*, 4GB) and lots of CPU cycles (*e.g.*, 2.0GHz), you might not need to give any of this a second thought!

☐ We've chased down and corrected almost all of the problems that folks ran into with the appliance during Problem Set 1. But at least one quirk remains, whereby if you double-click a `.c` file on your desktop (or somewhere in John Harvard's home directory), gedit might not actually open, though your cursor might start to spin. If you run into that issue, simply launch gedit via **Menu > Programming > gedit** (or via its icon in the appliance's bottom-left corner), then open the file in question via **File > Open…**.

☐ If you run into some technical difficulty with the appliance itself, unrelated to C code, do consult `https://manual.cs50.net/FAQs` first, followed by `http://help.cs50.net/`. Anytime you post to `help.cs50.net`, do take care to be as specific as possible, mentioning your OS (and version thereof), your symptoms, and what interventions you've already tried. "It's not working" isn't quite enough detail for us to know how we can help!

☐ Just to be safe, do get into the habit of backing up your `.c` files from time to time, whether to your own hard drive, to `dropbox.com`, or to CS50's servers. To back up your files to your own hard drive (outside of the appliance), see:

`https://manual.cs50.net/CS50_Appliance_2.3#How_to_Transfer_Files_between_Appliance_and_Your_Computer`

To synchronize with `dropbox.com`, see:

`https://manual.cs50.net/Appliance#How_to_Synchronize_Files_with_Dropbox`

To back up your files to CS50's servers, simply run `submit50` (per this document's end), as though you're submitting your work. You can re-submit as many times as you'd like (prior to the problem set's deadline), so you might as well submit as you go, just in case something goes wrong with your computer or appliance! If you need to retrieve your files from CS50's servers, consult `https://manual.cs50.net/FAQs`!

We'll soon introduce you to other techniques!

Getting Started.

☐ Alright, here we go!

Launch VirtualBox (as by double-clicking its icon wherever it's installed), and then boot the appliance (as by single-clicking it in VirtualBox's lefthand menu, and then clicking **Start**).

Upon reaching John Harvard's desktop, open a terminal window (remember how?) and type the below, followed by Enter:

```
sudo yum -y update
```

Input **crimson** if prompted for John Harvard's password. For security, <u>you won't see any characters as you type</u>. That command essentially does what **Menu > Administration > Software Update** does, but you'll see in more detail what's going on.

Realize that updating the appliance in this manner requires Internet access. If on a slow connection (or computer), it might take a few minutes to update the appliance. Don't worry if the process seems to hang if it decides to update a "package" called `cs50-appliance`; that one can take several minutes.

If you see messages like **Couldn't resolve host** or **Cannot retrieve metalink for repository**, those simply mean that the appliance doesn't currently have Internet access. Sometimes that happens if you've just awakened your computer from sleep or perhaps changed from wireless to wired Internet or vice versa. If your own computer does have Internet access (which you can confirm by trying to visit some website in a browser on your own computer) but the appliance does not (which you can confirm by trying to visit the same with Firefox within the appliance), try restarting the appliance (as by clicking the green icon in its bottom-right corner, then clicking **Restart**. If, upon restart, the appliance still doesn't have Internet access, head to `https://manual.cs50.net/FAQs` followed by `http://help.cs50.net/` for help!

Once the appliance has been updated, you should see **Complete!** in your terminal window. If there was nothing to update, you'll see **No packages marked for Update** instead.

☐ Juuuuuuuuuust to be sure that everything worked, go ahead and execute that very same command again in a terminal window (though not while its first invocation is still running):

```
sudo yum -y update
```

Again, input **crimson** if prompted for John Harvard's password. (If only a few minutes have passed since the last update, you might not even be prompted.) You should now see **No packages marked for Update**, which means that your appliance is now up-to-date! If you see some error instead, try once more, try to restart the appliance and then try once more, then head to `https://manual.cs50.net/FAQs` followed by `http://help.cs50.net/` as needed for help!

☐ Alright, here we go for real! Open a terminal window if not open already (whether by opening gedit via **Menu > Programming > gedit** or by opening Terminal itself via **Menu > Programming > Terminal**). Then execute

```
mkdir ~/hacker2
```

at your prompt in order to make a directory called `hacker2` in your home directory.[1] Take care not to overlook the space between `mkdir` and `~/hacker2` or any other character for that matter! Recall that `~` denotes your home directory, and thus `~/hacker2` denotes a directory called `hacker2` therein.

Now execute

```
cd ~/hacker2
```

to move yourself into (*i.e.*, open) that directory. Your prompt should now resemble the below.

```
jharvard@appliance (~/hacker2):
```

If not, retrace your steps and see if you can determine where you went wrong. You can actually execute

```
history
```

at the prompt to see your last several commands in chronological order if you'd like to do some sleuthing. You can also scroll through the same one line at a time by hitting your keyboard's up and down arrows; hit Enter to re-execute any command that you'd like. If still unsure how to fix, remember that `help.cs50.net` is your friend!

All of the work that you do for this problem set must ultimately reside in your `hacker2` directory for submission.

Passwords et cetera.

☐ On most, if not all, systems running Linux or UNIX is a file called `/etc/passwd`. By design, this file is meant to contain usernames and passwords, along with other account-related details (*e.g.*, paths to users' home directories and shells). Also by (poor) design, this file is typically world-readable. Thankfully, the passwords therein aren't stored "in the clear" but are instead encrypted using a "one-way hash function." When a user logs into these systems by typing a username and password, the latter is encrypted with the very same hash function, and the result is compared against the username's entry in `/etc/passwd`. If the two ciphertexts match, the user is allowed in. If you've ever forgotten some password, you may have been told that "I can't look up your

[1] If you decide to use `dropbox.com`, you can instead store your files in, say, `~/Dropbox/hacker2`.

password, but I can change it for you." It could be that person doesn't know how. But, odds are they just can't if a one-way hash function's involved.[2]

Even though passwords in /etc/passwd are encrypted, the crypto involved is not terribly strong. Quite often are adversaries, upon obtaining files like this one, able to guess (and check) users' passwords or crack them using brute force (*i.e.*, trying all possible passwords). Only in recent years have (most) system administrators stopped storing passwords in /etc/passwd, instead using /etc/shadow, which is (supposed to be) readable only by root.[3] Below, though, are some username:ciphertext pairs from some outdated (fake) systems.[4]

```
caesar:50zPJlUFIYY0o
chartier:50lMLvy/mlPIE
guest:50q.zrL5e0Sak
jharvard:50yoN9fp966dU
malan:50ym7PAOqs3Ko
rbowden:50WZ/Wy2GdA1Y
skroob:50Bpa7n/23iug
vigenere:505YXx3Mz50bg
```

Crack these passwords, each of which has been encrypted with C's DES-based (not MD5-based) crypt function. Specifically, write, in crack.c, a program that accepts a single command-line argument: an encrypted password.[5] If your program is executed without any command-line arguments or with more than one command-line argument, your program should complain and exit immediately, with main returning any non-zero int (thereby signifying an error that our own tests can detect). Otherwise, your program must proceed to crack the given password, ideally as quickly as possible, ultimately printing to standard output the password in the clear followed by \n, nothing more, nothing less, with main returning 0. The underlying design of this program is entirely up to you, but you must explain each and every one of your design decisions, including any implications for performance and accuracy, with profuse comments throughout your source code. Your program must be designed in such a way that it could crack all of the passwords above, even if said cracking might take quite a while. That is to say, it's okay if your code might take several minutes or days or longer to run. What we demand of you is correctness, not necessarily optimal performance. Your program should certainly work on inputs other than these as well; hard-coding into your program the solutions to the above is not acceptable.

So that we can automate some tests of your code, your program must behave per the below; highlighted in bold is some sample input.

```
jharvard@appliance (~/hacker2): ./crack 50q.zrL5e0Sak
password
```

[2] If you like this stuff, consider taking Computer Science 220r.
[3] Take a look at /etc/passwd in the appliance, for instance; wherever you see x a password once was.
[4] http://cdn.cs50.net/2011/fall/psets/2/hacker2/passwd
[5] In case you test your code with other ciphertexts, know that command-line arguments with certain characters (*e.g.*, ?) must be enclosed in single or double quotes; those quotation marks will not end up in argv itself.

Assume that users' passwords, as plaintext, are no longer than eight characters long. As for their ciphertexts, you'd best pull up the man page for `crypt` by executing

```
man crypt
```

in a terminal window so that you know how the function works. In particular, make sure you understand its use of a "salt." (According to the man page, a salt "is used to perturb the algorithm in one of 4096 different ways," but why might that be useful?) As implied by that man page, you'll likely want to put

```
#define _XOPEN_SOURCE
#include <unistd.h>
```

at the top of your file and link your program with `-lcrypt`. (If you use `make` to compile your code, that switch will be included automatically.)

You might also want to read up on C's support for file I/O, as there's quite a number of English words in `/usr/share/dict/words` in the appliance that might (or might not) save your program some time.

By design, `/etc/passwd` entrusts the security of passwords to an assumption: that adversaries lack the computational resources with which to crack those passwords. Once upon a time, that may have been true. Perhaps some still do. But when it comes to security, assumptions are dangerous. May that this problem set make that claim all the more real.

We should note that this problem set is no invitation to seek out other passwords to crack.[6] Do not conflate these Hacker Editions with "black hat" editions. We hope, though, that by understanding better the design of today's systems, you might one day build better systems yourself. Besides acquainting you further with C, this problem set urges you to start questioning designs, as vulnerabilities (if not regrets) often result from poor ones.

If you'd like to play with the staff's own implementation of `crack`, well, sorry! :-) Where'd be the fun in that?

Don't forget to back up your files (as to your own hard drive, to `dropbox.com`, or to CS50's servers with `submit50`)!

[6] In fact, do bear in mind the policies at
`http://www.fas-it.fas.harvard.edu/services/student/policies/rules_and_responsibilities`.

How to Submit.

In order to submit this problem set, you must first execute a command in the appliance and then submit a (brief) form online.

☐ Recall that you obtained a CS50 Cloud account (*i.e.*, username and password) for Problem Set 1. If you don't remember your username and/or password, head to `https://cloud.cs50.net/` look up the former and/or change the latter. You'll be prompted to log in with your HUID (or XID) and PIN.

☐ Just in case we updated `submit50` since you started this problem set, open a terminal window and execute the below, inputting **crimson** if prompted for John Harvard's password.

```
sudo yum -y update
```

If there was something to update, you should see **Complete!** after a few seconds or minutes. If there was nothing to update, you should instead see **No packages marked for Update**. If you see any errors, try the command once more, try to restart the appliance and then try once more, then head to `https://manual.cs50.net/FAQs` followed by `http://help.cs50.net/` as needed for help!

☐ To actually submit, first open a terminal window and execute:[7]

```
cd ~/hacker2
```

Then execute:

```
ls
```

At a minimum, you should see `crack.c`, capitalized and spelled exactly like that. If not, odds are you skipped some step(s) earlier! In particular, if you misnamed it (*e.g.*, as `Crack.c` instead of as `crack.c`), know that you can rename it with a command like

```
mv Crack.c crack.c
```

where `mv`'s first command-line argument is the file's current name and `mv`'s second command-line argument is the file's new name. If you don't see any or all of your files, you might have saved them somewhere else accidentally. Poke around John Harvard's desktop and home directory, and drag files as needed into the `hacker2` directory that should be in his home directory.

If everything is as it should be, you are ready to submit your source code to us. Execute:[8]

```
submit50 ~/hacker2
```

[7] Unless you decided to use `dropbox.com` and stored your files in, say, `~/Dropbox/hacker2`.
[8] *Ibid.*

When prompted for **Course**, input **cs50**; when prompted for **Repository**, input **hacker2**. When prompted for a username and password, input your CS50 Cloud username and password. For security, you won't see your password as you type it. That command will essentially upload your entire `~/hacker2` directory to CS50's repository, where your TF will be able to access it. The command will inform you whether your submission was successful or not. If provided with the URL of a PDF of your code (which further confirms its submission), right-click (or ctrl-click) the link, then choose **Open Link** from the menu that appears to open the PDF in Document Viewer.

You may re-submit as many times as you'd like; we'll grade your most recent submission. But take care not to submit after the problem set's deadline, lest you spend a late day unnecessarily or risk rejection entirely.

If you run into any trouble at all, let us know via `help.cs50.net` and we'll try to assist! Just take care to seek help well before the problem set's deadline, as we can't always reply within minutes!

☐ Head to the URL below where a short form awaits:

`https://www.cs50.net/psets/2/`

Once you have submitted that form (as well as your source code), you are done!

This was Problem Set 2.