CS 161 Computer Security — Fall 2006 — Joseph/Tygar — HW 1

Due Friday, September 22 at 11am

Please include the following at the top of the first page of your homework solution:

Your full name
Your login name
The name of the homework assignment (e.g. hw3)
Your section number/time
Names of students you worked with

Staple all pages together and put them in the CS 161/Fall 2006 slot of drop box #2 in 283 Soda. *No credit will be given after 11am on the due date. If you have not finished, turn in what you have for partial credit.*

Homework exercises:

1. **(1 pts.) Any questions?**
   What's the one thing you'd most like to see explained better in lecture or discussion sections? A one-line answer would be appreciated.

2. **(4 pts.) Getting started**
   (a) Read the course web page. Write on your homework, immediately after your name, the following sentence: "I understand and will comply with the academic integrity policy."
   (b) In your own words, what is the course policy regarding working on homework in groups?
   (c) Register with the grading system. These are the instruction for becoming registered for the class:
      i. Login to your instructional named account (cory account)
      ii. Set your environment variable $MASTERDIR to /home/ff/cs161
         - if your are using tcsh or csh: `setenv MASTERDIR /home/ff/cs161`
         - if you are using bash: `export MASTERDIR=/home/ff/cs161`
      iii. Run `register`

      That's it. If you want to, you can put the environment setting for the MASTERDIR variable in your .cshrc or .bashrc, so you don't have to set it every time you login.
   (d) What is Todd Kosloff's favorite security-related book? The answer is found on the course newsgroup, `ucb.class.cs161`. Look for the post from Todd Kosloff titled "The answer to question 2(d)," and write down the answer you find there. Instructions on how to access the newsgroup may be found on the course web page.
      (Why are we having you do this? The class newsgroup is your best source for recent announcements, clarifications on homeworks, and related matters, and we want you to be familiar with how to read the newsgroup.)

3. (20 pts.) PGP

In this question, you will learn how to use PGP, a popular format for public-key (asymmetric) email encryption. Plain email is totally insecure: it is easy to send forged email with a spoofed `From:` address, and it is often easy to intercept (eavesdrop on) email. PGP was first built by Phil Zimmerman as a way for activists to communicate securely over the internet. Over time, it has evolved into an open standard, called OpenPGP. There are several programs that support this message format: PGP is made by PGP Corporation; `gpg` is an open-source Gnu implementation. They can all interoperate (at least in theory). The instructional machines have `gpg` installed, so the following instructions assume that you will use `gpg` on the instructional machines (feel free to use another OpenPGP-compatible program, but you will have to work out the details yourself). You can get documentation on how to use `gpg` by typing `man gpg`. The instructional web pages also have a more detailed tutorial at `http://www-inst.eecs.berkeley.edu/cgi-bin/pub.cgi?file=gpg.help`.

The only part of this problem that requires a written answer on your solution set is part (b). You will also have to send an email to your TA with certain files attached as part of this solution. You may send the email anytime before the homework is due.

(a) Generate a new key pair. With `gpg`, you can use `gpg --gen-key`. This will construct a public key and a private key. Make sure your keysize is at least 1024 bits. Then export your public key into ASCII-armored format, and save the result to a file called `pubkey.asc`. If this is the first time you've used `gpg`, you can use `gpg --export --armor > ~/pubkey.asc`, but if you have other keys in your keyring you'll need to identify the key you wish to export (see the man page).

(b) Extract the fingerprint of your public key. `gpg` prints out the fingerprint when you generate the key, or you can use `gpg --fingerprint`. The fingerprint is a string of 40 hex digits that can be used to uniquely identify your public key. It is generated by applying a collision-resistant hash function to your public key, so no two public keys will have the same fingerprint. This is very useful: if I want to send you my public key, I can email you my public key (or send it to you over any insecure channel), and then we can compare fingerprints over a secure channel. For instance, I might print my fingerprint on my business card, or you can telephone me and I can read off my fingerprint out loud. Write down your fingerprint as the answer to this question.

(c) Get your TA's public key, and import it into your "keyfile". Your TA's public key will be stored in `~cs161/pubkeys/<ta-name>.asc` on the instructional machines. (Replace `<ta-name>` with the last name of your TA.) You can import the public key into your `gpg` keyfile using the command `gpg --import ~cs161/pubkeys/<ta-name>.asc`.

IMPORTANT: Check the fingerprint of the key you just imported. You can view the fingerprint of the key you just imported using `gpg --fingerprint`. Here are the fingerprints for the TA's public keys:

`DAA8 8B35 B33E DE13 38A6  C336 7C2C A3D0 D71F C73C`	Marco Barreno
`BF0E 3565 A12C DE0C D76B  4A37 9E30 F623 4E36 124D`	Todd Kosloff

Verifying fingerprints is important; otherwise, Mallet (the malicious attacker) might be able to trick you into accepting a key pair she generated as your TA's key, and then Mallet would be able to read all the encrypted mail you send to your TA.

(d) Once you have verified your TA's public key, sign it. You can use `gpg --sign-key name`, where `name` is a string that identifies your TA's public key (such as your TA's first or last name). Export the signed version of your TA's public key and save the result into the file `ta.asc`. You can use `gpg --armor --export name > ~/ta.asc`, where again `name` is a string that identifies your TA's public key.

(e) Finally, compose, encrypt, and sign a message to your TA. Create a text file containing a very short message to your TA, including your name. This is the cleartext. Now encrypt it with your TA's public key, sign it with your private key, and save the (ASCII-armored) result in `msg.asc`. `gpg --encrypt --sign --armor --recipient name <msg >msg.asc`, where `name` is a string identifying your TA, will generate the ciphertext for a cleartext called `msg`.

(f) Send an email to your TA with three files attached: `pubkey.asc`, `ta.asc`, and `msg.asc`.

4. (45 pts.) Repeated DES

You might think that one way to strengthen the use of the DES is to encrypt messages twice (using different keys). But double encryptions are subject to a "meet in the middle attack."

Let's use the notation C = E(P, K) to indicate that plaintext P is encrypted under cipher symmetric encryption function E by key K to produce ciphertext C. We'll also use D as the cipher symmetric decryption function, so P = D(C, K).

Now, the idea of double encryption is to use C = E(E(P, K1), K2). But the problem is that if we have a known-plaintext attack (where we know P and C) we can compute a table E(P, K) for all possible values K and also a table D(C, K) for all possible values K. We then look for a collision. This is the "meet in the middle attack."

(a) Show that performing a meet in the middle attack takes approximately twice as long[1] as the worst-case exhaustive search of a single encryption and four times as long as the expected-case exhaustive search of a single encryption.

(b) What if we use a triple encryption, that is C = E(E(E(P, K1), K2), K3)? How much does time does a known-plaintext attack take using meet in the middle? Give the details of the attack and explain your performance.

(c) Generalize a rule: if we use n-encryption with n keys, how much time does a known-plaintext attack take?

5. (15 pts.) Access Control

One of the requirements of the Health Insurance Portability and Accountability Act (HIPAA) is that health care organizations must protect patient files from unauthorized access. Would it be better to use a mandatory or a discretionary access control system to control access to patient files? Explain your answer.

6. (15 pts.) Access Control Lists and Capabilities

Give an example of a situation where an access control list (ACL)-based access control system is more efficient than a capability-based access control system. Give another example for where a capability-based system is more efficient.

[1] For this question, our notion of time is the number of times an encryption or decryption is performed.