6.033 Computer System Engineering

Spring 2009

Hands-on 6: Cryptography and Certificates

This hands-on assignment is due at the beginning of Recitation 23.

The goal of this hands-on is to give you an introduction to mathematics and the algorithmic building blocks of moderns cryptographic protocols. Before attempting this hands-on, you should read Chapter 11 of the class notes.

Part 1: Big Numbers and Brute-Force Attacks

One way to unseal a sealed message is to try every possible key. This kind of attack is known as a *brute-force attack* or a *key search* attack. The longer the key, the harder the attack.

Keys are almost always represented as blocks of binary data. Some cryptographic transformations use a fixed number of bits, while others allow a variable number. The table below lists some common cryptographic transformations and the key sizes that they use:

Cipher	Key Size
The Data Encryption Standard (DES)	56 bits
RC-2	40-1024 bits
RC-4	40-1024 bits
Advanced Encryption Standard (AES)	128, 192 or 256 bits

Although there are many factors that come into play when evaluating the strength of a cryptographic transformation, the length of the key is clearly important. This is because an attacker who is in possession of a sealed message can always mount a brute-force attack. Since longer keys have more possible values than shorter keys, longer keys are more resistant to brute-force attacks. (Note: this does not mean that cryptographic transformations that use longer keys are more secure. It may still be possible to attack a flaw in the algorithm, rather than simply testing every possible key. For example, the obsolete LUCIFER cryptographic transformation uses a 128-bit shared secret, while the DES uses a 56-bit key. But LUCIFER, unlike the DES, was susceptible to a particular kind of attack called differential cryptanalysis. This attack method was not publicly known at the time that the DES was published, but it was secretly known by the National Security Agency mathematicians who worked on DES.)

In this problem we will explore the impact of different key lengths on system security.

In general, because a key of n bits can have 2^n possible values, there can be at most 2^n different keys. For example, a 16-bit key can have 2^{16} or 65,536 different values. If you had a computer that could try 100 of these keys every second, it would take 654 seconds or roughly 11 minutes to try all possible keys. (If you are cracking many keys, the expected time to crack any given key is half that, as on average you will need to try half of the keys before you find the right one. Of course you could get lucky and try the key on your first attempt, or you could be unlucky and have to try nearly every single key.)

In your study of cryptography, you will discover that many products have the ability to use either so-called "weak" or "export-grade" encryption and "strong" or so-called "domestic" or "military-grade" cryptography. Usually the "weak" cryptography is limited to an effective secret key length of just 40 bits. The 40-bit restriction dates from a time when the United States government had regulations prohibiting the exportation of products containing strong cryptographic technology. Even though these regulations were largely eliminated more than five years ago, their legacy lives on today. For example, Microsoft Office XP uses a 40-bit RC2 key to seal documents that are given a "password."

With clever programming, a modern desktop computer can try over a million RC2 keys every second.

Question 1.1: What is the maximum amount of time that it would take for a computer that can try 1 million RC2 keys every second to do a brute-force attack on a Microsoft Office document sealed with a cryptographic transformation that uses a 40-bit shared secret?

Question 1.2: Microsoft Office 2003 uses the AES cryptographic transformation with a 128-bit shared secret to control access to documents controlled by Windows Rights Management technologies. If AES keys can be tried with the same speed as RC2 keys using the computer described in Question 1.1, what is the maximum amount of time that it would take for a brute-force attack on a single document sealed with the Windows Rights Management technology?

Question 1.3: How does your answer to question 1.2 compare to the age of the Universe, currently estimated at somewhere between 13.5 billion and 14 billion years?

With advances in technology it may be possible at some point in the future to have billions of high-speed computers in a very small volume.

Question 1.4: If you upgrade your computer to system that has a billion processing elements, each of which can try a billion keys in a second, is your secret still safe from attack?

In fact, computers are getting faster every year. Moore's law is commonly believed to hold that computers are doubling in speed every 18 months. What's more, faster techniques are being developed for reversing cryptographic transformations. Thus, simple estimates for the lifetime of a sealing key that do not take into account the relentless march of technology are inherently flawed.

Question 1.5: If you start with a computer today that can try 1 million keys every second and every 18 months you throw away that computer buy another for this project that is twice as fast, how long will it be until you have tried all possible 128-bit AES sealing keys?

One of the challenges in mounting a successful brute-force attack is that your program needs to be able to have some way of recognizing when it has guessed the correct key. Sometimes such recognitions are easy: the sealed text decrypts to English or another human-readable language. Your program can do a letter-frequency analysis on the resulting text and determine if the entropy is low or high; low-entropy indicates that the unsealing operation was successful. Recognizing a correct key becomes trivial if the decrypted message includes a checksum or message authentication code (MAC). In general, the

longer the ciphertext, the easier it is to recognize when a correct key is guessed.

Extra Credit: The AES standard allows for key lengths of 128 bits, 192 bits and 256 bits. Can you give a practical reason why a 192-bit or a 256-bit shared secret would provide more security than a 128-bit shared secret?

Part 2: Cryptographic Hashing

This section explores some properties of cryptographic hashing functions. For more information on cryptographic hashes, see section 11.2.3 of the notes. Cryptographic hashes are used as building blocks of many security primitives, including digital signatures (see section 11.3 of the notes).

The SHA-1 cryptographic hash function produces a 160-bit value, called a residue or a hash, for any given input. Since 160 bits is 20 bytes, there must be many files that have the same hash. (For example, given a file that is 21 bytes in length, there should be approximately 255 other 21-byte files that have the same hash. This is a simple application of the pigeonhole principle.) Nevertheless, no two files have yet been found that have the same SHA-1 hash, known as a "collision".

Note: While no SHA-1 collisions have been found, security flaws have been identified in it, and new hash standards are currently under development.

A version of the SHA-1 is built into the openssl command-line program that is available on Athena (`/usr/athena/bin/openssl`) and MacOS X (`/usr/bin/openssl`). When **openssl** is run with the **sha1** argument and one or more filenames, the program computes the SHA-1 hash of each file and prints the result as a hexadecimal string. If no files are presented, the program calculates the SHA-1 of any input presented on standard input.

For example, to compute the SHA-1 of the file `/etc/motd` you could use this command:

```
athena% openssl sha1 /etc/motd
SHA1(/etc/motd)= 1f3a70355ed8d34c5cc742fd64c2eca42b0d1846
athena%
```

Notice that this 160-bit output is encoded in 40 hexadecimal digits.

To calculate the SHA-1 of the string "MIT", you could use this command:

```
athena% echo "MIT" | openssl sha1
7bf26f2a41bb62f30b10f8a740df2508f86023e6
athena%
```

(In fact, the code that is shown above is actually the SHA-1 of the three characters M, I and T followed by a newline character.)

Question 2.1: Compute the SHA-1 of the string "Massachusetts Institute of Technology" (either with or without the newline).

Question 2.2: Estimate the chance that there another file on any computer at MIT that has the same SHA-1 value that you calculated in question 2.1. Show your work. To do this problem, you will need a rough estimate of the number of computers at MIT, and the number of unique files that each of those

computers contains.

Question 2.3: Compute how long it would take to find a string with the same SHA-1 hash as your answer in Question 2.1, using today's computers.

Part 3: GPG, Signatures and Certificates

This part of the hands-on assignment uses GNU Privacy Guard (GPG), a message security program that is based on the program Pretty Good Privacy (PGP). The GPG program implements a full suite of signing and sealing algorithms, bulk cryptographic transformation algorithms, and routines for the management of keys. It also interoperates with network of so-called *key servers* on the Internet on which people can publish their public keys.

GPG operates under the web of trust model. In this model you trust a certificate because it is vouched for by someone whose identity you already trust. You may contrast this with the certificate authority model, where a centralized third party, such as Verisign, vouches for the authenticity of a certificate.

To complete this part of the hands-on you will need a copy of GPG. You can log in to Project Athena and type `add gnu`, as there is a copy of GPG in the Athena "gnu" locker. Alternatively, you can download a copy of GPG from GNU privacy guard and install it on your own computer.

To learn about GPG's commands type `man gpg` and `gpg --help`. Note: it is a good idea to try both of these commands, because each one will teach you something different about GPG.

You can use GPG's `--version` command to discover the transformation algorithms, ciphers, and hashes that it supports.

The following questions make use of GPG Key A66F3DB0 which was created especially for this course and uploaded to the GPG key server `pgp.mit.edu` You will need to obtain the public key A66F3DB0 and add it GPG's database of public keys, which the program calls its *key ring*. You can either download the key from the key server using the GPG command `gpg --keyserver pgp.mit.edu --recv-keys A66F3DB0` or else you can copy the key below, run GPG with the `--import` command, and paste the key into GPG's standard input.

Here is the GPG key:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.9 (GNU/Linux)

mQGiBEnyLyQRBADeYRikleBYPFVknLgcU++1r2ByasAyFKFdGJOOQNeYOKYg7kfb
QItN81QEisi8192a/m90W/YkXdWBvWSSf7B2U38swt6phJclyR477/+NfijV7boc
p9EUGv4VT80IKFlmGgRPH4W26LEK1yyXMbd7YomdaRomhd1+diwXRk7ZNwCguMIp
nBByzXSohhywpY1YM2fJJqkD/RS34ZEEfTCipkrTUEjMhy8xWfkfLyS2CeY++K5/
S9K3wYwFFWquP4WVS32ZhlBQI5WsSUPiFFS58H+zUuPZuom2RL6nbjWlqepsKhpS
7MjkNBs3PlF6mEeRPyVdohdBf7xWRazDkeWWNyxFrx0re/G2B5lq+PdmfEvQgQJP
rg57BAC9wY3Us62/e8ts6t+La9BwBiuC+BKsQM3vhG7ZIK8/fCTAzIoXBmOCztTd
FbyM3WmEgKZUO3rRBq9YLvG+esModuxgrfP6a/FtVNHxtH4nM2FbG8IorRaH+gmR
YL1l/q7Xk8bU7eHdR2htpprhYEi25nQgnD2jRiLWLDTej8E0h7RKTUlUIDYuMDMz
IFNwcmluZyAyMDA5IChLZXkgZm9yIGhhbmRzLW9uICM2IC0gQ3J5cHRvKSA8Ni4w
MzMtc3RhZmZAbWl0LmVkdT6IZgQTEQIAJgUCSfIvJAIbAwUJABpeAAYLCQgHAwIE
```

```
FQIIAwQWAgMBAh4BAheAAAoJEBaHOXumbz2wfo4An3xUSPpxBa3WGFNjKufBImk7
MLuKAJ40zfj7jZpZsGs23xui/BnJxOUNHIhGBBARAgAGBQJJ87O7AAoJEPsKSMOO
yaTlhAUAn2ii1jCJztV8G0B62AhgYu6Pc+rIAJ0Y9hdHjzqu8RBA372YwbU0Kf+U
2Q==
=4zp8
-----END PGP PUBLIC KEY BLOCK-----
```

GPG keys can be used for signing documents or for signing other keys. By convention, signatures on documents are used to verify the document's author and to demonstrate that the document has not been modified since it was signed. Signatures on keys mean that the person signing the key is making an affirmative statement that a given public key really belongs to the person whose name is embedded inside the key. Keys with signatures binding them to names are called *certificates*. (See sections 11.5.1 and 11.7.4 of the notes.)

GPG supports supports a signature format called *clear signature* in which the signature appears at the bottom of the message. The course locker contains two messages that have GPG clear signatures at the bottom. Both documents were signed with the key that we created for this course, but one of the documents were modified.

Document #1:	
Filename:	`/mit/6.033/www/assignments/gpg-message1.txt.asc`
URL:	http://web.mit.edu/6.033/www/assignments/gpg-message1.txt.asc

Contents:	`-----BEGIN PGP SIGNED MESSAGE-----`
	`Hash: SHA1`
	
	`This is a message that was signed with GPG!`
	`-----BEGIN PGP SIGNATURE-----`
	`Version: GnuPG v1.4.9 (GNU/Linux)`
	
	`iEYEARECAAYFAkn14o4ACgkQFoc5e6ZvPbB7hgCgt74rbbP7MPmcNlDgNVBwZALm`
	`QPUAn3ggZLHC/Z/35QvuAkm6+tefu5Ll`
	`=qgXk`
	`-----END PGP SIGNATURE-----`

Document #2:	
Filename:	`/mit/6.033/www/assignments/gpg-message2.txt.asc`
URL:	http://web.mit.edu/6.033/www/assignments/gpg-message2.txt.asc

Contents:	`-----BEGIN PGP SIGNED MESSAGE-----`
	`Hash: SHA1`
	
	`This is not a message that was signed with GPG!`

```
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.9 (GNU/Linux)

iEYEARECAAYFAkn14q0ACgkQFoc5e6ZvPbDsCgCdFQ8nZmaCm0atWTaNAbAGW5lU
pGIAn2Rn94eH1tbvvmZxEniu9zst1KkK
=Ipaf
-----END PGP SIGNATURE-----
```

Question 3.1: In three sentences or less, describe how signing is different from sealing.

Question 3.2: Check that you have imported the key by using the GPG `--list-keys` command. What is the email address associated with the key?

Question 3.3: One of the above documents were modified after it was signed. Tell us which one! Show the output from GPG that proves your assertion.

Question 3.4: Given your knowledge of signing, explain what information *must* be stored in the PGP signatures of the above messages?

Question 3.5: Someone has signed key A66F3DB0, verifying its itentity. Whose signature is on this key? (Hint: You may need use `--list-sigs` and to access the key server to download additional keys to answer this question.) How do you know that the signature is legitimate? Can you trust that key A66F3DB0 was really made for this course? Why or why not?

Question 3.6: Download key 0B72EB0F from the key server. Whose key is this? Can you trust this key because you got it from the official MIT key server? Why or why not?