Homework 1 Solutions

CS161 Computer Security, Spring 2008

1. **One-Time Pads**

   To communicate using a one-time pad, Alice and Bob would first need to share a secret, random ℓ-bit string $r \in \{0,1\}^\ell$ (e.g., established at an in person meeting). Then given their shared secret r, Alice could later privately send an ℓ-bit message $m \in \{0,1\}^\ell$ by sending Bob $s = m \oplus r$. Bob could then in turn compute the message as $s \oplus r = m$.

   Having just learned about the one-time pad, Alice is excited about the security it offers. However, she is concerned about the inconvenience of establishing the shared key r with Bob. The following protocol occurs to her.

   > When Alice is ready to send her message m, she randomly selects $r_1 \in \{0,1\}^\ell$ and sends Bob $s_1 = m \oplus r_1$. Bob then randomly selects $r_2 \in \{0,1\}^\ell$ and sends Alice $s_2 = s_1 \oplus r_2$. Next, Alice computes $s_3 = s_2 \oplus r_1$ and sends it to Bob. Bob may then compute the message as $s_3 \oplus r_2 = m$.

   This idea seems similar to the one-time pad, but does not require prior distribution of a shared key.

   (a) (1 point) Is Alice's protocol secure?
   Answer: No.

   (b) (2 points) If your answer is no, give an attack that breaks the protocol. If your answer is yes, briefly specify your attack model and state why you think the scheme might be secure under that model (no formal proof is necessary).

Answer: A passive eavesdropper can directly compute m from s_1, s_2, and s_3 as shown below.

$$s_1 \oplus s_2 \oplus s_3 = (m \oplus r_1) \oplus (m \oplus r_1 \oplus r_2) \oplus (m \oplus r_1 \oplus r_2 \oplus r_1) = m$$

2. **Block Ciphers**

   This question contains a small programming assignment that is designed to ensure that your named UNIX accounts are properly set up for CS161 while also illustrating some properties of block ciphers.

   We've published a code skeleton with a number of BMP image files[1] at `http://inst.eecs.berkeley.edu/~cs161/sp08/hw01code.tar.gz`. Untar the file by running `gtar zxf hw01code.tar.gz`, and run `make` to compile the program. Then type

   ```
   ./encrypt_ecb plaintext-medium.bmp out.bmp
   ```

   to to attempt to encrypt the image. The skeleton will pass a null pointer into `write()` and fail with a "bad address" error until you fill in the solution to the assignment.

   We've tested the code on Linux and on `cory.eecs.berekeley.edu`. We will use `cory.eecs.berkeley.edu` to grade the assignment, so make sure that your code builds and runs on that machine.

   The two bitmap files contain uncompressed versions of the same image. One contains a monochrome (1 bit per pixel) version of the picture, and the other contains a 200% scaled RGB (24 bits per pixel) version of the picture.

   When you've completed the question, submit the files `encrypt_ecb.c`, `plain.bmp`, `crypt.bmp`, and `encrypt_cbc.c`. Instructions for electronic submissions will be given on the website later this week.

   (a) (3 points) Edit `encrypt_ecb.c` so that it implements DES encryption in electronic codebook (ECB) mode. Encrypt the two included image files.

   Answer: We expect to see a discernable pattern for both images.

[1] From `http://xkcd.com/`.

(b) (3 points) The skeleton intentionally avoids encrypting the BMP file's header so you can use standard software to inspect the encrypted data. Open the two encrypted files in your favorite image editor. (We used the GIMP.) Was the encryption effective? Explain the difference between the two encrypted files.

Answer: No, ECB mode encryption always encrypts the all-white or all-black regions of the file as the same pattern. The encrypted version of the smaller, monochrome file format can store 64 pixels per ECB block, while only 64/24 pixels in the larger image fit in an ECB block. Therefore, the larger file is easily readable, while the smaller file's text is scrambled.

(c) (3 points) Find a small (under 1MB uncompressed) image file that would be more effectively encrypted using a block cipher in ECB mode. Explain your choice of image. What sorts of attacks is this scheme vulnerable to?

Convert the image to an uncompressed (not RLE encoded) 24-bit BMP format and encrypt it. Name the files `plain.bmp` and `crypt.bmp` and submit them.

Answer: We were looking for a photograph, or other image that is unlikely to contain identical runs of pixels. In addition to revealing repititions within a single image, this scheme is vulnerable to repititions across multiple images. Also, the image's blocks could be rearranged. CBC encryption is a much better choice for this type of application.

(d) (3 points) Edit `encrypt_cbc.c` so that it implements DES encryption in cipher block chaining (CBC) mode. For this exercise, a block of all 0's may be used as the initialization vector. Encrypt the two original image files and view the result.

How do the results differ when CBC is used instead of ECB? Briefly state why that is the case.

Answer: CBC's output should be indistinguishable from random noise because it encrypts each block of the file based upon the values of previous blocks.

(e) (extra credit, 4 points) Alice and Bob decided to use a modified version of the homework submission for secure communications. Alice tried to send her first encrypted email using one of the encryption programs:

From: Alice
To: Bob
Date: Feb 5th, 2008 12:34:56PM

Bob,

I just generated a key. Did it work?

-A

However, Mallory intercepted it before it reached Bob's computer. A few seconds later (far too soon to brute-force a 56-bit key), Mallory encrypted a completely different message using Alice's key, and sent it to Bob. Eve saw the forged message, and now everyone's upset. Mallory told you there's a bug in the implementations of `main()` that we provided to you. What happened?

Answer: The skeleton uses the real time clock's current value rounded to the nearest second. Mallory knew the current value of Alice's clock (within a minute or so). With this information, she knew Alice's key was one of a few hundred values.

She guessed Alice's key by feeding seeds close to the current time into the skeleton's key generation code. After a few hundred tries, her computer guessed the correct seed, computed the shared key and successfully decrypted the email. Then Mallory used the key to forge a new message, and sent it to Bob.

3. **Hash Functions and Collisions**

Let k be a positive integer and let $h : \{0,1\}^* \to \{0,1\}^k$. For this problem we will assume h is an idealized, perfectly random hash function. Specifically, assume h is selected uniformly at random from all functions mapping $\{0,1\}^*$ to $\{0,1\}^k$.

An attacker is interested finding collisions of h. Assume the attacker treats the hash function as a black box; i.e., the only operation they can perform is to compute the hash function on an input and observe the result. Further assume they can perform at most one million hash computations per second.

Hint: to solve the following two problems, you may need to use an

approximation, as direct computation of k may be difficult with a fixed precision calculator. For this question it is acceptable to look up and cite outside sources for any useful approximations. Briefly show how you arrived at your answers.

(a) (6 points) Suppose the attacker is interested in finding a preimage $x \in \{0,1\}^*$ which hashes to a particular value $y \in \{0,1\}^k$. What is the minimum value of k that ensures the attacker will have at most a 0.1% chance of succeeding in one year?

Answer: Note that the attacker can perform $n = 1,000,000 \cdot 60 \cdot 60 \cdot 24 \cdot 365.25 = 3.15576 \times 10^{13}$ hash computations per year. The attacker may try hashing sequential values or randomly chosen values. Provided all the values the attacker hashes are distinct (it would be pointless to repeat them), each trial will succeed with probability $\frac{1}{2^k}$, independent of the others.

The probability that the attacker succeeds is then

$$1 - \left(1 - \frac{1}{2^k}\right)^n \quad ,$$

and we require that this be at most 0.001. So we want to find the minimum k that satisfies the following inequality.

$$\left(\frac{2^k - 1}{2^k}\right)^n \geq 0.999$$

Computing k exactly. We computed this k explicitly using `bc` to do arbitrary precision calculations. The above condition holds iff

$$n \log \frac{2^k - 1}{2^k} \geq \log 0.999 \quad .$$

If you run `bc -l` and doing the following, you can see that $k = 55$ satisfies this inequality and $k = 54$ does not.

```
scale = 1000
n = 1000000 * 60 * 60 * 24 * 365.25
k = 55
n * l((2^k - 1)/2^k)
l(0.999)
k = 54
n * l((2^k - 1)/2^k)
```

Therefore the minimum k is exactly 55.

Another approach. Define random variable X to be the number of times the attacker guesses the right answer. Then X is a binomial random variable with parameters $p = \frac{1}{2^k}$ and $n = 3.15576 \times 10^{13}$. The probability of the attacker winning is $P(X \geq 1)$.

Many students at this point chose k so that the expected value of X at most 0.001. Since $E(X) = np$, this requires that

$$\frac{n}{2^k} \leq 0.001$$

The minimum k which satisfies that inequality is also 55. However, in general this is an incorrect approach to solving the problem, so we assigned half credit to students who simply stated the reasoning above. Specifically, $E(X) \leq 0.001$ is a distinct condition from $P(X \geq 1) \leq 0.001$, and it is essentially only a coincidence that $k = 55$ is the minimum value that satisfies both.

To see why using expectation and probability interchangably is problematic, consider two fair coin tosses. If X is the number of heads observed, we have $E(X) = 2 \cdot 0.5 = 1$, but $P(X \geq 1) = 0.75$.

The reason why this approach ends up with the right answer on this problem is that, when the probability of an individual trial succeeding is exceptionally small, $P(X \geq 1) \approx P(X = 1)$.

(b) (6 points) Suppose the attacker is interested in finding any collision, that is, preimages $x_1, x_2 \in \{0,1\}^*$ which both hash to the same value in $\{0,1\}^k$. What is the minimum value of k that ensures the attacker will have at most an 80% chance of succeeding in one year?

Answer: It doesn't seem to be possible to compute the answer to this problem explicitly, even using an arbitrary precision calculator. However, by using both upper and lower bounds the minimum value for k can be narrowed down (at least) to either 88, 89, or 90.

That is, we can derive that $k \geq 88$ is a *necessary* condition for the attacker to fail with probability at least 0.8, and that $k \geq 90$

is a *sufficient* condition for the attacker to fail with probability at least 0.8.

To show that $k \geq 88$ is necessary, we can use the inequality $1 + x \leq e^x$ for all $x \in \mathbb{R}$ that is suggested on Wikipedia. The probability that all n hashes are distinct is

$$\begin{aligned} & \left(1 - \frac{1}{2^k}\right) \cdot \left(1 - \frac{2}{2^k}\right) \cdot \left(1 - \frac{1}{3^k}\right) \cdot \cdots \left(1 - \frac{n-1}{2^k}\right) \\ \leq\ & e^{-\frac{1}{2^k}} \cdot e^{-\frac{2}{2^k}} \cdot e^{-\frac{1}{3^k}} \cdot \cdots e^{-\frac{n-1}{2^k}} \\ =\ & e^{-\frac{1+2+3+\cdots(n-1)}{2^k}} \\ =\ & e^{-\frac{(n-1)\cdot(n-2)}{2\cdot 2^k}} \ . \end{aligned}$$

The last expression is less than 0.2 for all $k < 88$, so $k \geq 88$ is a necessary condition for the original probability to be greater than or equal to 0.2.

We can use the union bound to get a sufficient condition on k. If $y_1, y_2, \ldots y_n$ are the computed hash values (taken as random variables), then the probability of a collision is

$$\begin{aligned} & P(y_1 = y_2 \wedge y_1 = y_3 \wedge \cdots y_1 = y_n \wedge y_2 = y_3 \wedge \cdots y_{n-1} = y_n) \\ \leq\ & P(y_1 = y_2) + P(y_1 = y_3) + \cdots P(y_{n-1} = y_n) \\ =\ & \frac{(n-1)\cdot(n-2)}{2} \cdot \frac{1}{2^k} \ . \end{aligned}$$

The last expression is less than 0.8 iff $k \geq 90$. So we know that any $k \geq 90$ will ensure that the attacker has at most a 0.8 chance of success.

Thus the minimum value of k which ensures the required property is either 88, 89, or 90.

4. **ElGamal and Chosen Ciphertext Attacks**

Recall that to use the ElGamal public key encryption scheme, Alice randomly selects a private key $x \in \mathbb{Z}_p$ and computes her public key as $y = g^x \mod p$, where g is a publicly known generator of $\mathbb{Z}_p^*$. To encrypt a message $m \in \mathbb{Z}_p^*$ for Alice, Bob randomly selects $r \in \mathbb{Z}_p$ and computes the ciphertext as $c = (g^r, m \cdot y^r)$.

(a) (3 points) Assume you are given an ElGamal public key y (but not the private key). Assume ciphertexts $c_a = (c_{a_1}, c_{a_2})$ and $c_b = (c_{b_1}, c_{b_2})$ are encryptions of some unknown messages m_a and m_b.

Show how you can construct a ciphertext which is a valid ElGamal encryption of the message $m_a \cdot m_b \mod p$.

Answer: The ciphertext may be constructed as follows, where all computations are done modulo p.

Since c_a and c_b are encryptions of m_a and m_b, there exist $r_a, r_b \in \mathbb{Z}_p$ such that $c_a = (g^{r_a}, m_a \cdot y^{r_a})$ and $c_b = (g^{r_b}, m_b \cdot y^{r_b})$. Now define $r_c = r_a + r_b$ and compute the following.

$$c_{c_1} = c_{a_1} \cdot c_{b_1} = g^{r_a + r_b} = g^{r_c}$$
$$c_{c_2} = c_{a_2} \cdot c_{b_2} = m_a \cdot m_b \cdot y^{r_a + r_b} = m_a \cdot m_b \cdot y^{r_c}$$

So $c_c = (c_{c_1}, c_{c_2})$ is a valid encryption of $m_a \cdot m_b$.

(b) (4 points) Show how the above property of ElGamal leads to a chosen ciphertext attack.

That is, assume you are given an ElGamal public key y and a ciphertext $c = (c_1, c_2)$ which is an encryption of some unknown message m and that you are furthermore given access to an oracle that will decrypt any ciphertext other than c. Based on these things, compute m.

Answer: (Again we implicitly assume computation modulo p.) Since c is encryption of m there exists an $r \in \mathbb{Z}_p$ such that $c = (g^r, m \cdot y^r)$.

Pick any $m' \in \mathbb{Z}_p^*, m' \neq 1$ and any $r' \in \mathbb{Z}_p, r' \neq 0$ and compute

$$c_1' = c_1 \cdot g^{r'} = g^{r+r'}$$
$$c_2' = c_2 \cdot m' \cdot y^{r'} = m \cdot m' \cdot y^{r+r'}$$

Define $c' = (c_1', c_2')$ and submit it to the oracle for decryption. Note that c' is a valid encryption of $m \cdot m'$ and $c' \neq c$, so the oracle will give us $m \cdot m'$ as the result. Given $m \cdot m'$, we may simple multiply by m'^{-1} to obtain m.

5. **Factoring and More**

Given an integer n, we say that m is a *non-trivial factor* of n if $m|n$, $m \neq n$, and $m \neq 1$.

(a) (4 points) Assume $n = pq$, where p and q are prime. It can be shown that for every $x \in \mathbb{Z}_n^*$, there exists a $y \in \mathbb{Z}_n^*$ such that $y \neq x$, $y \neq -x$, and $x^2 \equiv y^2 \mod n$.

Assume you are given such an x and y in $\mathbb{Z}_n^*$, that is, $x^2 \equiv y^2 \mod n$, $x \neq y$, and $x \neq -y$. Show that $\gcd(x - y, n)$ is a non-trivial factor of n.

Answer: First of all, n has only four divisors: 1, p, q, and pq. Since $\gcd(x - y, n)$ is a divisor of n and p and q are non-trivial factors of n, we only need to show that $\gcd(x - y, n) \neq pq$ and $\gcd(x - y, n) \neq 1$. To see that, we reason as follows.

Suppose $\gcd(x - y, n) = pq$. That would imply that $x - y$ is a multiple of n, i.e., that $x - y \equiv 0 \mod n$. This cannot be the case because $x \neq y$, so $\gcd(x - y, n) \neq pq$.

Now suppose that $\gcd(x - y, n) = 1$. This implies that neither p nor q are divisors of $(x - y)$, because either of those would be a greater divisor than 1. Next note that $(x - y) \cdot (x + y) = x^2 - y^2 \equiv 0 \mod n$. So $n|(x - y) \cdot (x + y)$; that is, the product $(x - y) \cdot (x + y)$ contains both a factor of p and a factor of q. If $(x - y)$ has neither p nor q as a factor, then $(x + y)$ must have both p and q as factors, in which case $n|(x + y)$. But that cannot be the case, because we know that $x \neq -y$ so $x + y \not\equiv 0 \mod n$. So $\gcd(x - y, n) \neq 1$.

Thus we have that either $\gcd(x - y, n) = p$ or $\gcd(x - y, n) = q$, both of which are non-trivial factors of n.

(b) (1 point) Suppose we are considering a function $h : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ for use as a hash function, where $h(x) = x^2 \mod n$. Does h satisfy the compression property of a hash function? No justification of your answer is required.

Answer: No. (The domain and range of h are the same.)

(c) (8 points) If we assume the difficulty of factoring, does h satisfy the preimage resistance (a.k.a. one-way) property of hash functions?

If your answer is no, give a probabilistic polynomial time algorithm that, when given n and y, will output an $x \in \mathbb{Z}_n^*$ such that $h(x) = y$ or abort if no such x exists.

If your answer is yes, show how such an algorithm could be used to factor n in expected polynomial time.

Answer: Yes.

Assume we have a probabilistic polynomial time algorithm A that, when given n and y, will output an $x \in \mathbb{Z}_n^*$ such that $h(x) = y$ or abort if no such x exists. Then we may factor n in expected polynomial time as follows.

Randomly select an $x \in \mathbb{Z}_n^*$ and compute $y = x^2 \mod n$. Based on the fact stated at the beginning of part (a), we know there exists an x' such that $x' \neq x$, $x' \neq -x$, and $x'^2 = y \mod n$. So y has at least four square roots modulo n: x, $-x$, x', and $-x'$.

Now run A on n and y. If A outputs x or $-x$, select a new x and try again. This failure case can only happen with probability at most $\frac{1}{2}$, because we selected x randomly and only gave y to A. That is, from A's perspective, this situation is indistinguishable the situation of us originally picking x' and then giving $y = x'^2 \mod n$ to A.

So eventually, A will output one of the other square roots of y, that is, an x' such that $x' \neq x$, $x' \neq -x$, and $x'^2 \mod n = y$. Specifically, the expected number of tries before A produces such an x' is 2.

Next compute $m = \gcd(x - x', n)$. Based on the answer to part (a), we know that m is a non-trivial factor of n. Specifically, m is either p or q. Thus, m and $\frac{n}{m}$ are the prime factors of n.

(d) (3 points) If we assume the difficulty of factoring, does h satisfy the second preimage resistance (a.k.a. weak collision resistance) property of hash functions?

If your answer is no, give a probabilistic polynomial time algorithm that, when given n and x, will output an $x' \neq x$ such that $h(x) = h(x')$ or abort if no such x exists.

If your answer is yes, show how such an algorithm could be used to factor n in expected polynomial time.

Answer: No. Let $x' = -x$ (i.e., $n - x$), then $h(x) = h(x')$.