UNIVERSITY OF MARYLAND
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENEE 459C
Computer Security
Instructor: Charalampos Papamanthou

Homework 3

Out: 10/17/14 Due: 10/31/14

Instructions

1. Strictly adhere to the University of Maryland Code of Academic Integrity.
2. **Type** and submit your solutions as a pdf document at Canvas. Include your full name in the solutions document. Name the solutions document as x-hw3.pdf, where x is your last name.
3. No extensions will be given. Everything will be due at 11.59pm on 10/31/14.

Problem 1 (20 points)

1. You are given that $3 \cdot 5 - 2 \cdot 7 = 1$. Use the Chinese remainder theorem to compute
$$3^{72} \mod 35\,.$$
2. The Blum-Micali generator outputs the next pseudorandom bit as follows: First it computes
$$x_i = g^{x_{i-1}} \mod p$$
and then it outputs the bit 1 if $x_i < \frac{p-1}{2}$, else it outputs the bit 0 (the procedure is repeated for $i = 0, 1, 2, \ldots$). Write a program that implements `get_random_bit()` using the above generator for $p = 2147483647$ (note that in real applications, p needs to be a lot bigger) and $g = 7$. Use $x_0 = 1$. Also, write a program that implements `get_random_number_BM()` and outputs a 32-bit pseudorandom number by calling `get_random_bit()` 32 times. Note that for implementing the exponentiation above, you should use the modular power algorithm (that runs in logarithmic time) that we discussed in class.

Problem 2 (20 points)

1. What is the difference between public-key encryption and secret-key encryption? What is a certification authority? What is a digital certificate?
2. Suppose Mallory is responsible for storing all the public keys of the students that are currently enrolled at the University of Maryland. Namely, it keeps a list of mappings of (d_i, pk_i), where d_i is the directory identifier of student i and pk_i is the public key of student i. Suppose Bob, Alice and Eve are all students at the university. Bob wants to send a confidential message to Alice and asks Mallory for Alice's public key. However Eve is really interested in learning what Bob wants to say to Alice. Eve happens to be very good friends with Mallory. Explain how Eve could collaborate with Mallory so that Eve eventually gets to decrypt Bob's message to Alice.

3. Instead of using Mallory for managing public keys, Bob and Alice decide to setup their public keys with Verisign, Inc., a trusted certification authority. In order to arrange a meeting for the weekend, Bob exchanges with Alice brief text messages, using RSA encryption, as described in slide 7 of Lecture 9 (click here to access Lecture 9). Let p_B and p_A be the public keys of Bob and Alice respectively. A message m sent by Bob to Alice is transmitted as $\mathsf{Enc}_{p_A}(m)$ and the reply r from Alice to Bob is transmitted as $\mathsf{Enc}_{p_B}(r)$. Although Eve cannot collaborate with Mallory anymore, she can still eavesdrop the communication and knows the following information:
   - Bob and Alice are meeting either on a Saturday or on a Sunday;
   - The set of 100 possible meeting places;
   - Alice is going to specify the time in the format HH:MM pm or HH:MM am.
   - Messages and replies are terse exchanges of the following form:

     Bob: When are we meeting?
     Alice: Saturday.
     Bob: Where are we meeting?
     Alice: Union station.
     Bob: What time?
     Alice: 8.30pm.

   Describe how Eve can learn the meeting day, time, and place of Bob and Alice.
4. As a result of the above attack, Bob and Alice decide to modify the protocol for exchanging messages. Describe two simple modifications of the protocol that are not subject to the above attack. The first one should use random numbers and the second one should use symmetric encryption.

Problem 3 (20 points)

In class we talked about some basic number theory and its significance for various cryptographic algorithms such as RSA and ElGamal. Answer the following questions:

1. Suppose $n = 43434563$. How many elements in $\mathbb{Z}_n - \{0\}$ have a multiplicative inverse modulo n? Name such an element (i.e., one that has a multiplicative inverse modulo n). You might want to visit this webpage to answer this question.
2. Consider an RSA key set with $p = 11$, $q = 29$, $n = pq = 319$, and $e = 3$. What value of d should be used in the secret key? What is the encryption of message $M = 100$? Why should e always be an odd number (to answer this question, assume p and q are always greater than 2)? You might want to visit this webpage to answer this question.
3. Prove that *all* elements in $\mathbb{Z}_n - \{0\}$ have a multiplicative inverse modulo n **if and only if** n is a prime. Note that you need to prove both directions.
4. Let now n be a positive integer and let $\mathbb{Z}_n^* = \{x \in [0, n] : \mathsf{GCD}(x, n) = 1\}$. Let also $k < n$ and $a \in \mathbb{Z}_n^*$. What is the multiplicative inverse of $a^k \in \mathbb{Z}_n^*$ when n is a prime? What is the multiplicative inverse of $a^k \in \mathbb{Z}_n^*$ in case that n is not a prime (and has p and q as prime factors)? You do not have to use the extended Euclidean algorithm to answer this question.
5. Prove that for all integers a, k and n, it is $\mathsf{GCD}(a, n) = \mathsf{GCD}(a + kn, n)$ (Hint: Use the same technique we used in class to prove that $\mathsf{GCD}(a, b) = \mathsf{GCD}(b, a \bmod b)$).
6. Prove that for all integers a, b and primes p it is $(a + b)^p = a^p + b^p \mod p$ (Hint: Use Fermat's little theorem).

7. Recall the extended Euclidean algorithm: Given two integers a and b, it outputs two other integers x and y such that $xa + yb = \mathsf{GCD}(a, b)$. Devise an algorithm that uses the extended Euclidean algorithm as a black box, such that, on input 3 integers a, b and c, it outputs three integers u, v and z such that $ua + vb + zc = \mathsf{GCD}(a, b, c)$ (Hint: Use the fact that $\mathsf{GCD}(a, b, c) = \mathsf{GCD}(\mathsf{GCD}(a, b), c)$).

Problem 4 (20 points)

In this question, you will learn how to use PGP, a popular format for public-key (asymmetric) email encryption. Plain email is insecure; it is easy to send forged email with a spoofed `From:` address, and it is often easy to intercept 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, and `gpg` is an open-source Gnu implementation.

In order to use `gpg`, you will need to have access to a Linux machine (there are options for Windows and OS X, but the instructions here pertain to Linux). As such, you may want to install VirtualBox or another VM tool to run Ubuntu 14.04 LTS on your own machine. You can get documentation on how to use `gpg` by typing `man gpg`.

1. 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).
2. 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. **Write down your fingerprint in your solution document.**
3. Get your TA's public key, and import it into your "keyfile". Your TA's public key is stored at `http://enee459c.github.io/homeworks/code/<ta-name>.asc`. (Replace `<ta-name>` with the first name of your TA.) You can import the public key into your `gpg` keyfile using the command `gpg --import <ta-name>.asc`.
   IMPORTANT: Check the fingerprint of the key you just imported using `gpg --fingerprint`. Here is the fingerprint for the TA's public key:
   `C9B6 FEB2 BD39 43C5 5CCD 5760 13AD 7DD9 46E7 458A`
   Answer in your solution document: Why is verifying the fingerprint important?
4. 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.
5. Compose, encrypt, and sign a message to your TA. Create a text file containing at least 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`.

6. Send an email to your TA with three files attached: `pubkey.asc, ta.asc, msg.asc`.

Problem 5 (20 points)

In this programming assignment, you will hack into a server running at IP `129.2.212.154` by figuring out the private key of a user! To prove to us that your attack succeeded, you are going to submit the contents of the file `/home/chell/cake.txt` that is stored at the server, as well as leave your own message in a file inside the same directory. We now give some guidelines:

SSH public-key authentication. SSH (secure shell) is a protocol that is used to establish secure connection to a server. To establish an SSH connection, a user typically generates a public/private key pair by executing command `ssh-keygen`, and then adds the produced public key to `.ssh/authorized_keys`. Then the user can use his private key to log into the server with SSH. You can read more about SSH here.

The bug. `openssl` is a cryptographic library that is used by `ssh-keygen`. In 2006, Debian, a popular Linux distribution, introduced a bug into `openssl` by commenting out code dealing with getting entropy for the pseudorandom number generator. You can read more details about the bug here. The effect of the change was that the only source of entropy for `openssl` was now the process's PID, which can be any value from 1 to `/proc/sys/kernel/pid_max` (usually 32768).

This meant that, if someone was using the buggy version of `openssl` to get randomness and generate key pairs, instead of generating one of the possible $\sim 2^{2048}$ key pairs, he would be generating only one of the 32768 key pairs.

How to hack it. You know that the user `chell@129.2.212.154` is using a key pair generated with a vulnerable version of `openssl`. Your goal is to log into his account and read the file `/home/hackme/file.txt`.

To achieve that, write a program to generate all 32768 key pairs and try logging into the server with them until you find the correct one. To generate the vulnerable keys, you can download from here an `.iso` image of an old version of Ubuntu that uses the vulnerable version of `openssl` and use virtualization software (e.g., VirtualBox, visit webpage here) to boot the `.iso`. Then, in the virtual machine, run `ssh-keygen` with PID from 1 to 32768 to generate all the possible keys.

The following code will run `ssh-keygen` with a PID of 1, and will try to log into the server with the generated key. First, since it is hard to make a program run with a chosen PID, we instead use LD_PRELOAD to intercept calls to `getpid()`, so that `ssh-keygen` eventually runs with our chosen PID. Save the following code to a file named `fakepid.c`:

```
#include <stdlib.h>
int getpid() {
    /* instead of returning the actual PID, return whatever
     * number is in the environment variable FAKEPID */
    return atoi(getenv("FAKEPID"));
}
```

Now, compile `fakepid.c` as a shared library, so that we can use LD_PRELOAD to tell the linker to link `ssh-keygen` with our version of `getpid()` instead of the normal one:

```
$ gcc -shared -fPIC fakepid.c -o fakepid.so
```

Now, run `ssh-keygen`, using LD_PRELOAD and `fakepid.so` to execute `ssh-keygen` with the PID we want (in Linux, `'ENVIRONMENT_VARIABLE=value command'` runs `command` with the environment variable set to `value`):

```
$ FAKEPID=1 LD_PRELOAD="./fakepid.so" ssh-keygen -N "" -f key_for_pid_1
```

Finally, try to log into the server using the key you just generated:

```
$ ssh -i key_for_pid_1 hackme@107.20.43.191
```

Submit the contents of the file `/home/chell/cake.txt`, along with any code you wrote for the problem, and leave a message containing at least your name.