Homework 5: Hash Function and Public Key Cryptography

1 Overview

The learning objective of this homework is for students to get familiar with the concept and application of one-way hash functions and Public-Key cryptography. To achieve the objective, the students will do both theory (paper-and-pencil problems) and practice (programming and lab). After finishing the homework, in addition to gaining a deeper understanding of the concepts, students should be able to use tools and write programs to apply the learned knowledge.

2 Paper-and-Pencil Problems [40 pts]

There are a total of 4 paper-and-pencil problems from the KPS textbook ($X.Y.$ means number Y homework in Chapter X). (For your convenience, we provide the full description of the problems.) Please make sure to read KPS textbook closely before answering these questions.

- 5.2. (8 pts): Message digests are reasonably fast, but here's a much faster function to compute. Take your message, divide it into 128-bit chunks, and xor all the chunks together to get a 128-bit result. Do the standard message digest on the result. Is this a good message digest function?
- 5.14. (12 pts): For purposes of this exercise, we will define random as having all elements equally likely to be chosen. So a function that selects a 100-bit number will be random if every 100-bit number is equally likely to be chosen. Using this definition, if we look at the function "+" and we have two inputs, x and y, then the output will be random if at least one of x and y are random. For the following functions, find sufficient conditions for x, y and z under which the output will be random:

  $\sim x$

  $x \oplus y$

  $x \vee y$

  $x \wedge y$

  $(x \wedge y) \vee (\sim x \wedge z)$ [the selection function]

  $(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$ [the majority function]

  $x \oplus y \oplus z$

  $y \oplus (x \vee \sim z)$
- 6.2. (8 pts): In [KPS] textbook, it states that encrypting the Diffie-Hellman value with the other side's public key prevents the man-in-the-middle attack. Why is this the case, given that an attacker can encrypt whatever it wants with the other side's public key?
- 6.8. (12 pts): Suppose Fred sees your RSA signature on m_1 and on m_2 (i.e., he sees m_1^d mod n and m_2^d mod n). How does he compute the signature on each of m_1^j mod n (for positive integer j), m_1^{-1} mod n, $m_1 m_2$, and in general $m_1^j m_2^k$ mod n (for arbitrary integers j and k)?

3 Lab and Programming Tasks [60 + 10 bonus pts]

We will use the same OpenSSL lab environment as before. OpenSSL Command-Line HOWTO can be found here `http://www.madboa.com/geek/openssl/`.

3.1 Task 1: Generating Message Digest and MAC [7 pts]

In this task, we will play with various one-way hash algorithms. You can use the following `openssl dgst` command to generate the hash value for a file. To see the manuals, you can type `man openssl` and `man dgst`.

```
 % openssl dgst dgsttype filename
```

Please replace the `dgsttype` with a specific one-way hash algorithm, such as `-md5`, `-sha1`, `-sha256`, etc. In this task, you should try at least 3 different algorithms, and describe your observations. You can find the supported one-way hash algorithms by typing "`man openssl`".

3.2 Task 2: Keyed Hash and HMAC [9 pts]

In this task, we would like to generate a keyed hash (i.e. MAC) for a file. We can use the `-hmac` option (this option is currently undocumented, but it is supported by `openssl`). The following example generates a keyed hash for a file using the HMAC-MD5 algorithm. The string following the `-hmac` option is the key.

```
% openssl dgst -md5 -hmac "abcdefg" filename
```

Please generate a keyed hash using HMAC-MD5, HMC-SHA256, and HMAC-SHA1 for any file that you choose. Please try several keys with different length. Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

3.3 Task 3: The Randomness of One-way Hash [9 pts]

To understand the properties of one-way hash functions, we would like to do the following exercise for MD5 and SHA256:

1. Create a text file of any length.
2. Generate the hash value H_1 for this file using a specific hash algorithm.
3. Flip one bit of the input file. You can achieve this modification using `ghex`.
4. Generate the hash value H_2 for the modified file.
5. Please observe whether H_1 and H_2 are similar or not. Please describe your observations in the report. You can write a short program to count how many bits are the same between H_1 and H_2.

3.4 Task 4: Hash Collision-Free Property [20+10 bonus pts]

In this task, we will investigate hash function's collision-free properties. We will use the brute-force method to see how long it takes to break these properties. Instead of using `openssl`'s command-line tools, you are required to write your own C programs to invoke the message digest functions in `openssl`'s crypto library. A sample code can be found from `http://www.openssl.org/docs/crypto/EVP_DigestInit.html`. Please get familiar with this sample code.

Since most of the hash functions are quite strong against the brute-force attack on those two properties, it will take us years to break them using the brute-force method. To make the task feasible, we reduce the length of the hash value to 24 bits. We can use any one-way hash function, but we only use the first 24 bits of the hash value in this task. Namely, we are using a modified one-way hash function. Please design an experiment to find out the following:

1. How many trials it will take you to find two messages with the same hash values using the brute-force method? You should repeat your experiment for multiple times, and report your average number of trials.
2. How many trials it will take you to find a message has the same hash value as a given/known message's hash value using the brute-force method? Similarly, you should report the average.
3. Based on your observation, which case is easier to break using the brute-force method?
4. (10 Bonus Points) Can you explain the difference in your observation mathematically (i.e., a formal proof)?

3.5 Task 5: Performance Comparison: RSA versus AES [8 pts]

In this task, we will study the performance of public-key algorithms. Please prepare a file (`message.txt`) that contains a 16-byte message. Please also generate an 1024-bit RSA public/private key pair. Then, do the following:

1. Encrypt `message.txt` using the public key; save the the output in `message_enc.txt`.
2. Decrypt `message_enc.txt` using the private key.
3. Encrypt `message.txt` using a 128-bit AES key.
4. Compare the time spent on each of the above operations, and describe your observations. If an operation is too fast, you may want to repeat it for many times, and then take an average.

After you finish the above exercise, you can now use `OpenSSL`'s `speed` command to do such a benchmarking. Please describe whether your observations are similar to those from the outputs of the `speed` command. The following command shows examples of using `speed` to benchmark `rsa` and `aes`:

```
% openssl speed rsa
% openssl speed aes
```

3.6 Task 6: Create Digital Signature [7 pts]

In this task, we will use `OpenSSL` to generate digital signatures. Please prepare a file (`example.txt`) of any size. Please also prepare an RSA public/private key pair. Do the following:

1. Sign the SHA256 hash of `example.txt`; save the output in `example.sha256`.
2. Verify the digital signature in `example.sha256`.
3. Slightly modify `example.txt`, and verify the digital signature again.

Please describe how you did the above operations (e.g., what commands do you use, etc.). Explain your observations. Please also explain why digital signatures are useful.

4 Submission

You need to submit a detailed homework report (as well as any required program) to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising. In your report, you need to answer all the questions listed in this homework.