Shi Su
shis

Problem #1 : Using PGP

Email: shis@andrew.cmu.edu OpenPGP keyID: 58053E5D Fingerprint: 2BA3 6514 F478 CF9F 5C06 0C04 A6CB 1835 5805 3E5D Public key is available on https://pgp.mit.edu/

1. Based on this information, how do you verify that the public key you got from the web page is valid, i.e., that no one has modified it?

By comparing the downloaded key's fingerprint with the supposedly correct fingerprint provided in the assignment handout, it there?re the same, the key is authentic. Because the fingerprint is a hash of the user?s certificate [1], if the key get modified, the hash (fingerprint) will change as well, and the course handout downloaded is from a trusted resources, so we can assume the fingerprint is correct.

3. If you had to send Mahmood an attachment, would you use PGP/MIME or PGP/inline, knowing that the various email client(s) he uses support both methods? Justify your answer.

PGP/MIME, because PGP/inline only encrypt the plain text inside the email message, while PGP/MIME allows you to encrypt attachment together with the message body. [2]

[1] Pgpi.org,. (2015). How PGP works. Retrieved 5 October 2015, from http://www.pgpi.org/doc/pgpintro/#p11

[2] Enigmail.net,. (2015). Enigmail: Enigmail Configuration Manual. Retrieved 5 October 2015, from https://www.enigmail.net/documentation/per-account.php

Problem #2 : Padding attack

Python script available in **shis_hw1/MD5_collisions/**

Decryption Function
shis_decticket.py

When running without arguments, the script decrypt the cipher provided in the homework.

Use ”-c” to provide a different cipher
Use ”-h” for more information

Encryption Function
shis_goldticket.py

This python script is importing function from shis_decticket.py, so please keep them under same directory.

When running without arguments, the script encrypt the following ticket
{”username”:”shis”,”is_admin”:”true”,”expired”:”2020-01-31”}

Use ”-u” to provide a different username, e.g. ”instructor”
Use ”-a” to provide admin setting, ”true” or ”false”
Use ”-e” to provide expiration date, e.g. ”2016-01-31”
Use ”-h” for more information

Problem #3 : Finding MD5 collisions

1. A succinct description of how in practice the attack works. Do not delve into the cryptographic details, simply assume the result of Wang and Yu is well known.

1. Write a script file contains 2 methods perform different tasks, compile it with a main body contains a switch to choose between the methods using 2 placeholder ”crib” strings.
2. Read the first block and get the MD5 initial vector of the executable generated from step 1.
3. Given the initial vector, find the MD5 collision pair using the method of Wang and Yu.
4. Copy the executable file from step 1 to create 2 executable, at the same time replace the ”crib” string with one of the collision pair into each new executable.

Thus most of the content of the 2 executables are the same, except the collision pair. Because of the switch in the main function, they have different behavior, and according to Wang and Yu?s work, they have same MD5 hash.
3. Answers to the following questions:

- What is the linearity property that makes the attack work for files of arbitrary length?

  According to Wang and Yu's attack, giving arbitrary IV, it's able to find two pairs of blocks M,M' and N,N', which makes f(f(s, M), M') = f(f(s, N), N?), f is the hash function applied to each block. Because of the function f's input is only the output of previous block, this linearity property ensure that after MD5 function processed these 2 blocks, it generates same output for next block. Thus, as long as the blocks before/after M,M' and N,N' are the same, they can have arbitrary length.

- We saw here how to use this attack on an executable. How would you go about implementing such attack on a document? Which feature must the document format have? Is it possible to carry out such an attack in a completely stealth manner on an ASCII document?

  Selinger's attack requires each file contains both version of content, to implement this attack on document, the document format must support hidden contents. So it's not possible carry this attack on an ASCII document. In an ASCII document, there's no way to hide the extra data especially the ascii characters.

Problem #4 : Public Key Cryptography

1. Calculate the private key d:

When knowing p = 23, q = 17, e = 3, we can get: N = pq = 23 * 17 = 391
And d is the modular multiplicative inverse of e, from ed = 1 (mod (p-1)(q-1)):
ed - k(p-1)(q-1) = 1
3d - 1 = 352k
3d - 352k = 1
Use Euclid's Algorithm to compute gcd(352,3)
352 = 117 * 3 + 1
3 = 3 * 1 + 0
Then we get: 1 = (-117)*3 + 352, to get a positive d, add 352*3 - 3*352 into equation
1 = (-117)*3 + 352*3 - 3*352 +352
1 = 235*3 - 2*352
So private key d = 235
2. Describe which party (keyboard/SecApp) knows which keys (public key/private key), and show the steps of encryption and decryption of the exemplary keyboard input ?B?.

Keyboard knows the public key in order to encrypt the input, only SecApp knows the private key, so only it can decrypt the message. When sending ?B? (ascii: 66), keyboard encrypted it with public key (n,e)=(391,3), according to $m^e = c$(mod n) $66^3 = c(mod 391)$, c = 111, so 'B' is encrypted to 111 When SecApp decrypting the message with private key d, according $c^d = m(mod n)$ $111^2 35 = m(mod 391)$, so after the app can calculate m = 66. (in python, pow(111,235,391))

3. Assuming the key distribution is secure, list one advantage and one disadvantage for picking e=3 as the public key? Use e=3 such a small exponent as the public means less calculation which can make the encryption or signature verification process faster. However the small exponent may also be less secure in some cases, for example when the message is not padded simply or the private key is partially exposed, etc. [1]

4. Another company needs your technical input to beat this product. Please list 3 different arguments for why you think this design is not good. p,q are too small, which results in a small n=pq, making the prime factorization of it much easier. When p,q got revealed, it?s simple to calculator private key d with ed mod(q-1)(p-1)=1 Because of the information is sending from keyboard, suppose it sends each key typed right away, that means for each encryption the length of plain text is 1 and is likely subject to short plain text attack. Also it only has limited variation of inputs, so if the algorithm doesn?t have a good padding scheme, an attacker may collect large amount of cipher texts and apply frequency analysis attack on them. If the keyboard encrypts bunch of keys together, it can lead to input latency. Asymmetric encryption method like RSA is generally much slower to calculator than those symmetric ones, and keyboard is a real time input device, the delay for encrypting and decrypting may largely sacrifice the user experience.

[1] Boneh, D. (1999). Twenty years of attacks on the RSA cryptosystem. Notices of the AMS, 46(2), 203-213.