4/15/2018

Hash Function and Public Key Cryptography

CSCE 465 Homework 5

Nicholas Charchenko

Book Problems

1. 5.3: It should take around 2^33 attempts to find a collision. We know that the probability that a given Type 2 message has the same message digest as one of the 2^32 Type 1 messages is roughly 2^32/2^64, equivalent to 1/2^32, so it is likely that one of the 2^32 Type 2 messages matches one of the 2^32 Type 1 messages by the birthday problem.
2. 5.4: The digest size will remain 64 bits. Therefore, the iterations will remain the same, and if a palindrome exists within the message space, then the message will be the same forward and backward, leading to a collision when the two messages are hashed.
3. 5.14: For ~x, if x is random, then the output will be random. For x XOR y, if either x or y is random, then the output will be random. For x OR y, a fixed 1 bit will lead to non-random output, while a fixed 0 bit or both x and y being random will lead to a random output. For x AND y, the output will be random if there is either a fixed 1 bit or if both x and y are random. For the selection function, if any two of x,y, and z are random, then the output will be random. For the majority function, if any two are random, then the output will be random. For x XOR y XOR z, at least two of x,y, and z must be random. For the last function, if either y, or both x and z are random, then the output will be random. **Note: It is assumed that if any fixed variable can tamper with the results, then the result is not truly random.**
4. 6.2: The attacker will not be able to decrypt the Diffie-Hellman values sent to him and so will not be able to compute the shared secrets.
5. 6.8: We can compute m_1^j mod n for any positive integer j by this formula: $\left(m_1^d\right)^j$mod n = $\left(m_1^j\right)^d$ mod n. We apply this principle and then compute the signature of m_1^x using the fact that the signature of m_1^{-1} is $(m_1^{-1})^d$ mod $n = (m_1^{-d})$ mod $n = (m_1^d)^{-1}$ mod n. We can similarly compute the signature of m_2^k for any k. Now we know how to compute the signatures of m_1^j and m_2^k. We can compute the signature of the product with the formula $m_1 * m_2 = (m_1 * m_2)^d$ mod $n = ((m_1)^d$ mod $n) * ((m_2)^d$mod $n)$ mod n. Another property to note is that if $j = 0$, then the signature of $m_1^j = m_1^{-j}$.

Task 1: Generating Message Digest and MAC

For this task, I created a file called example.txt with the sentence “This is an example.” Then, I used OpenSSL to implement MD5, SHA-1, and SHA-256 algorithms.

```
[04/11/2018 15:18] seed@ubuntu:~/Desktop$ openssl dgst -md5 example.txt
MD5(example.txt)= 46edc6541babd006bb52223c664b29a3
[04/11/2018 15:18] seed@ubuntu:~/Desktop$ openssl dgst -sha1 example.txt
SHA1(example.txt)= a6f153801c9303d73ca2b43d3be62f44c6b66476
[04/11/2018 15:19] seed@ubuntu:~/Desktop$ openssl dgst -sha256 example.txt
SHA256(example.txt)= c80a97041f15ba166b9a3e8fc2b09726d778bc3bd9338d4befe34b46707
ebeec
```

Figure 1: MD5, SHA-1, and SHA-256 hash outputs

I can observe that MD5 gives a 32-bit hash length, SHA-1 gives a 40-bit hash length, and SHA-256 gives a 64-bit hash length. This proves that MD5 has a 128-bit hash value, SHA-1 has a 160-bit hash value, and SHA-256 has a 256-bit hash value.

Task 2: Keyed Hash and HMAC

I created a separate file called hmacexample.txt and ran MD5, SHA-1, and SHA-256 hashing algorithms with three different keys: “a”, “abcd”, and “abcdefghijklmnopqrstuvwxyz” to show keys of varying lengths. The key length does not need to be fixed, and the HMAC lengths are the same as the lengths of the hash codes from Task 1.

```
[04/14/2018 14:35] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac "a" hmacexampl
e.txt
HMAC-MD5(hmacexample.txt)= 22e9f0b7c3c0deb895c67ae34ea9dae6
[04/14/2018 14:59] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac "abcd" hmacexa
mple.txt
HMAC-MD5(hmacexample.txt)= 404295b1a68aeb9efc083c917ce802df
[04/14/2018 14:59] seed@ubuntu:~/Desktop$ openssl dgst -md5 -hmac "abcdefghijklm
nopqrstuvwxyz" hmacexample.txt
HMAC-MD5(hmacexample.txt)= 4eff411ef91f3a801df0ab4dd48fbb13
[04/14/2018 15:00] seed@ubuntu:~/Desktop$ openssl dgst -sha1 -hmac "a" hmacexamp
le.txt
HMAC-SHA1(hmacexample.txt)= 7b2455572759eeb983a73a6656063fc8c60574b1
[04/14/2018 15:02] seed@ubuntu:~/Desktop$ openssl dgst -sha1 -hmac "abcd" hmacex
ample.txt
HMAC-SHA1(hmacexample.txt)= 14872a0bb8b463c10a6b3dddf102cc3e6aee68c7
[04/14/2018 15:04] seed@ubuntu:~/Desktop$ openssl dgst -sha1 -hmac "abcdefghijkl
mnopqrstuvwxyz" hmacexample.txt
HMAC-SHA1(hmacexample.txt)= 75d247f29199d754dc6bc16970e572246a610cbb
[04/14/2018 15:05] seed@ubuntu:~/Desktop$ openssl dgst -sha256 -hmac "a" hmacexa
mple.txt
HMAC-SHA256(hmacexample.txt)= e64fa62d78708340525cd981815fe26e7d5004804f8ef5796c
89f956e70dd486
[04/14/2018 15:05] seed@ubuntu:~/Desktop$ openssl dgst -sha256 -hmac "abcd" hmac
example.txt
HMAC-SHA256(hmacexample.txt)= eb1803f7a886db56394e62c08875f4d6c973b7fd22b2810ca6
19b4dcc525f3cf
[04/14/2018 15:05] seed@ubuntu:~/Desktop$ openssl dgst -sha256 -hmac "abcdefghij
klmnopqrstuvwxyz" hmacexample.txt
HMAC-SHA256(hmacexample.txt)= 9061dd4106f2e3a76fdd3e0ecef91d06ae21dfee8fdc658b71
b680ab6c194130
[04/14/2018 15:05] seed@ubuntu:~/Desktop$
```

Figure 2: Output for Task 2: Keyed Hash and HMAC

Task 3: The Randomness of One-Way Hash

For this task, I created a file called task3.txt with the sentence, “This is an example for Task 3.” (no quotes). I then did the hashing with MD5 and SHA-256 for H1. For H2, I changed “Task 3” to “task 3” using Ghex and ran the algorithms again. I observe that flipping this one bit causes the entire hash to change for both MD5 and SHA-256 algorithms. I compared the outputs for MD5 and only 1 bit was shared while SHA-256 had 6 shared bits between the two outputs. This shows that there is a strong avalanche effect with hash algorithms, similar to secret key encryption algorithms.

```
[04/14/2018 15:39] seed@ubuntu:~/Desktop$ openssl dgst -md5 task3.txt
MD5(task3.txt)= e92b190b4959481ad1421c158f26f5aa
[04/14/2018 15:39] seed@ubuntu:~/Desktop$ openssl dgst -sha256 task3.txt
SHA256(task3.txt)= 62b78c62465534f1f7e2817c4ce3c0fb032e35aaa4a32aba034d34f71b5cb46a
[04/14/2018 15:39] seed@ubuntu:~/Desktop$ openssl dgst -md5 task3.txt
MD5(task3.txt)= 5c6fa2ad50ccf4d98bf05907edb444a4
[04/14/2018 16:06] seed@ubuntu:~/Desktop$ openssl dgst -sha256 task3.txt
SHA256(task3.txt)= 45489a677d87472c6f7fe9f7b683f0bb73abe0a477e6542b7ae92026881026c9
[04/14/2018 16:06] seed@ubuntu:~/Desktop$
```

Figure 3: Changing one bit of task3.txt makes for a significant change in the hash.

Task 4: One-Way vs Collision-Free Properties

For this task, I designed a program using C to create experiments to perform attacks against both the one-way hash property and the collision-free hash property. To make it easier, I set it up so that we only had 24-bit values, since otherwise it could take a very long time to brute-force. I used random number generation to create messages to be hashed and then repeatedly made hashes using loops to run 15 trials for breaking the collision property, and 5 trials for breaking the one-way property because breaking the one-way property proved to take much longer, outputting the average number of attempts. I used both SHA-256 and MD5 algorithms in this task. From my observations, it is clear that the one-way property is more difficult to break.

```
Terminal
[04/14/2018 19:32] seed@ubuntu:~$ cd Desktop
[04/14/2018 19:32] seed@ubuntu:~/Desktop$ ./task4 md5
Collision after 10805 tries! Same digest d41d8c
Collision after 26350 tries! Same digest 008f0d
Collision after 3717 tries! Same digest 003017
Collision after 44724 tries! Same digest 008814
Collision after 941 tries! Same digest d41d8c
Collision after 26849 tries! Same digest d41d8c
Collision after 39279 tries! Same digest d41d8c
Collision after 30269 tries! Same digest d41d8c
Collision after 1715 tries! Same digest 00dece
Collision after 9705 tries! Same digest 005baa
Collision after 11182 tries! Same digest d41d8c
Collision after 8115 tries! Same digest 005c47
Collision after 5092 tries! Same digest d41d8c
Collision after 7288 tries! Same digest d41d8c
Collision after 18823 tries! Same digest d41d8c
Average number of attempts cracking collision-free property: 16323
One-way property cracked after 23623610 tries! Same digest feb8c1
One-way property cracked after 38272206 tries! Same digest 83ed98
One-way property cracked after 9915555 tries! Same digest f467cd
One-way property cracked after 6984570 tries! Same digest 4487de
One-way property cracked after 21860127 tries! Same digest 78dd3f
Average number of attempts cracking one-way property: 20131213
[04/14/2018 20:20] seed@ubuntu:~/Desktop$
```

```
Terminal
[04/14/2018 19:32] seed@ubuntu:~$ cd Desktop
[04/14/2018 19:32] seed@ubuntu:~/Desktop$ ./task4 sha256
Collision after 26581 tries! Same digest e3b0c4
Collision after 75739 tries! Same digest e3b0c4
Collision after 93906 tries! Same digest 008819
Collision after 28367 tries! Same digest e3b0c4
Collision after 32928 tries! Same digest 0028d4
Collision after 59081 tries! Same digest 002dcf
Collision after 49489 tries! Same digest 00d71d
Collision after 8901 tries! Same digest 00ee62
Collision after 48483 tries! Same digest 004b9e
Collision after 28240 tries! Same digest e3b0c4
Collision after 20644 tries! Same digest e3b0c4
Collision after 855 tries! Same digest 006e43
Collision after 21533 tries! Same digest 00a30d
Collision after 30968 tries! Same digest e3b0c4
Collision after 524 tries! Same digest e3b0c4
Average number of attempts cracking collision-free property: 35082
One-way property cracked after 4637439 tries! Same digest a034ef
One-way property cracked after 48422956 tries! Same digest 0e97e4
One-way property cracked after 32151829 tries! Same digest 853e7e
One-way property cracked after 17488374 tries! Same digest cfe562
One-way property cracked after 10484588 tries! Same digest 4afaf2
Average number of attempts cracking one-way property: 22637037
[04/14/2018 20:23] seed@ubuntu:~/Desktop$
```

```
Terminal
dl
[04/14/2018 19:25] seed@ubuntu:~/Desktop$ ./task4 md5
Collision after 8213 tries! Same digest d41d8c
Collision after 97445 tries! Same digest 0026fa
Collision after 1628 tries! Same digest d41d8c
Collision after 25973 tries! Same digest 00fd0c
Collision after 51400 tries! Same digest d41d8c
Collision after 115104 tries! Same digest 000600
Collision after 5143 tries! Same digest 00deba
Collision after 3646 tries! Same digest 001fec
Collision after 400 tries! Same digest d41d8c
Collision after 27496 tries! Same digest d41d8c
Collision after 79167 tries! Same digest d41d8c
Collision after 34380 tries! Same digest d41d8c
Collision after 8107 tries! Same digest d41d8c
Collision after 10373 tries! Same digest d41d8c
Collision after 10493 tries! Same digest 00b0ce
Average number of attempts cracking collision-free property: 31931
One-way property cracked after 2243621 tries! Same digest 83580e
One-way property cracked after 4220590 tries! Same digest a471b5
One-way property cracked after 263731 tries! Same digest f40235
One-way property cracked after 39625771 tries! Same digest b243fe
One-way property cracked after 15360544 tries! Same digest 329346
Average number of attempts cracking one-way property: 12342851
[04/14/2018 19:53] seed@ubuntu:~/Desktop$
```

```
Terminal
[04/14/2018 19:26] seed@ubuntu:~$ cd Desktop
[04/14/2018 19:26] seed@ubuntu:~/Desktop$ ./task4 sha256
Collision after 5789 tries! Same digest e3b0c4
Collision after 61966 tries! Same digest e3b0c4
Collision after 96721 tries! Same digest e3b0c4
Collision after 84237 tries! Same digest 00342a
Collision after 24681 tries! Same digest 00e5d8
Collision after 21199 tries! Same digest e3b0c4
Collision after 97974 tries! Same digest e3b0c4
Collision after 45064 tries! Same digest 00d897
Collision after 27188 tries! Same digest e3b0c4
Collision after 20871 tries! Same digest 005ad8
Collision after 21059 tries! Same digest e3b0c4
Collision after 27422 tries! Same digest e3b0c4
Collision after 30075 tries! Same digest 003acc
Collision after 19479 tries! Same digest 008174
Collision after 100437 tries! Same digest 005998
Average number of attempts cracking collision-free property: 45610
One-way property cracked after 5520744 tries! Same digest 39e600
One-way property cracked after 1662320 tries! Same digest a78753
One-way property cracked after 11373303 tries! Same digest a846b9
One-way property cracked after 42467107 tries! Same digest 315c8a
One-way property cracked after 21619426 tries! Same digest c924cd
Average number of attempts cracking one-way property: 16528580
[04/14/2018 20:09] seed@ubuntu:~/Desktop$
```

Figure 4: Brute-Force attacks on Collision and One-Way properties of both MD5 and SHA-256

We can think about this in context of the birthday problem. The birthday problem states that the probability that at least one student was born on a specific day is $1 - \left(\frac{364}{365}\right)^n$ for a class of n students. For $n = 30$, this probability is around 7.9%. **This is the same as breaking the one-way hash property.** This is because breaking the one-way hash property requires that, given a message m and a hash H(m), we find a message m' such that H(m')=H(m), like finding a student born on a specific day. However, it is much easier to find a student that shares a birthday with any other student. This probability is equivalent to $1 - \frac{365!}{(365-n)!*365^n}$, which is equal to around 70% for $n = 30$. **This is the same as breaking the collision hash property.**

Task 5: RSA vs AES

I used the following commands for Task 5:

- openssl genrsa -out privatekey.pem 1024
  - Generate private key
- openssl rsa -in mykey.pem -pubout -out mypubkey.pem
  - Generate public key
- time openssl rsautl -encrypt -in message.txt -pubin -inkey mypubkey.pem -out message.enc
  - Timing RSA encryption
- time openssl rsautl -decrypt -in message.enc -inkey mykey.pem -out message_decrypted.txt
  - Timing RSA decryption
- time openssl enc -aes-128-cbc -in message.txt -out message.aes \-K a \-iv a
  - Timing AES-128-CBC encryption

The RSA decrypt averaged around 0.004 seconds, the RSA encrypt and AES-128-CBC encryption averaged around 0.003 seconds. Running the speed benchmark on RSA for 1024-bit keys yielded that signing on average took 0.001135 seconds and verifying took on average 0.000053 seconds. For AES speed benchmarks, I got the following results:

```
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128 cbc      79778.77k    89718.42k    90975.36k   176637.59k   168611.77k
```

Figure 5: Speed tests on AES-128 CBC encryption

Task 6: Create a Digital Signature

The following screenshot shows the commands and steps used for Task 6:

```
[04/14/2018 20:14] seed@ubuntu:~$ cd Desktop
[04/14/2018 20:14] seed@ubuntu:~/Desktop$ openssl genrsa -des3 -out myrsaCA.pem
1024
Generating RSA private key, 1024 bit long modulus
......++++++
...++++++
unable to write 'random state'
e is 65537 (0x10001)
Enter pass phrase for myrsaCA.pem:
Verifying - Enter pass phrase for myrsaCA.pem:
[04/14/2018 20:14] seed@ubuntu:~/Desktop$ openssl rsa -in myrsaCA.pem -pubout -o
ut myrsapubkey.pem
Enter pass phrase for myrsaCA.pem:
writing RSA key
[04/14/2018 20:17] seed@ubuntu:~/Desktop$ openssl dgst -sha256 -out example.sha2
56 -sign myrsaCA.pem example.txt
Enter pass phrase for myrsaCA.pem:
[04/14/2018 20:17] seed@ubuntu:~/Desktop$ openssl genrsa -des3 -out myrsaCA.pem
1024
Generating RSA private key, 1024 bit long modulus
........................++++++
.........++++++
unable to write 'random state'
e is 65537 (0x10001)
Enter pass phrase for myrsaCA.pem:
Verifying - Enter pass phrase for myrsaCA.pem:
[04/14/2018 20:17] seed@ubuntu:~/Desktop$ openssl dgst -sha256 -signature exampl
e.sha256 -verify myrsapubkey.pem example.txt
Verified OK
[04/14/2018 20:20] seed@ubuntu:~/Desktop$ openssl dgst -sha256 -signature exampl
e.sha256 -verify myrsapubkey.pem example.txt
Verification Failure
[04/14/2018 20:21] seed@ubuntu:~/Desktop$ openssl dgst -sha256 -signature exampl
e.sha256 -verify myrsapubkey.pem example.txt
Verified OK
[04/14/2018 20:33] seed@ubuntu:~/Desktop$ openssl dgst -sha256 -signature exampl
e.sha256 -verify myrsapubkey.pem example.txt
Verification Failure
[04/14/2018 20:33] seed@ubuntu:~/Desktop$ █
```

Figure 6: Task 6-Creating Digital Signature

I noticed that even the slightest change in example.txt would cause the subsequent verification to fail. This is because I first made a significant change (adding a single word), and got a failure. Then, I added a whitespace character and got a verification failure. To create the RSA public/private key pair, I followed the same process as in Task 5 for generating the key pair.

Digital signatures are important in security because they ensure integrity and non-repudiation. From performing Task 6, the tampered document fails to verify. Using this, we can tell if a document was tampered, and we know not to use it. If the verification succeeds, then we know that the document is legitimate and trustworthy, improving our security. Furthermore, digital signatures help ensure delivery and helps track who interacts with the information.