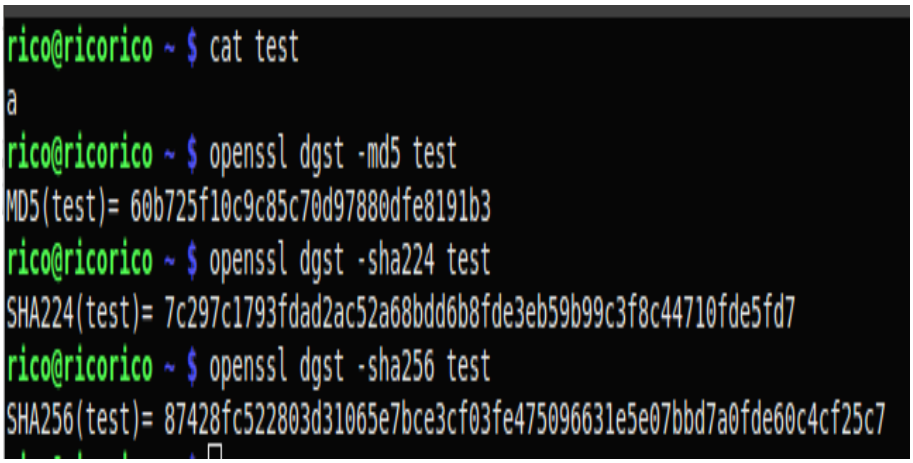
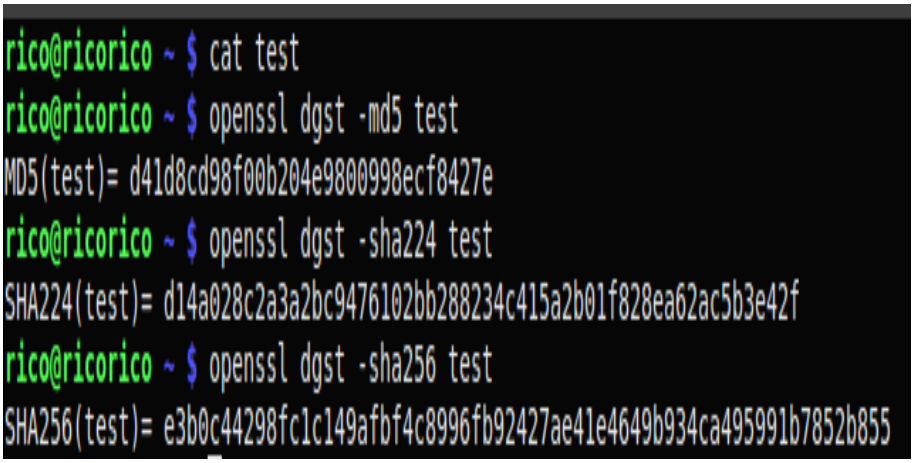
CPS713 - APPLIED CRYPTOGRAPHY - LAB 2

MATTHEW FRANCIS AND REAZ HUQ

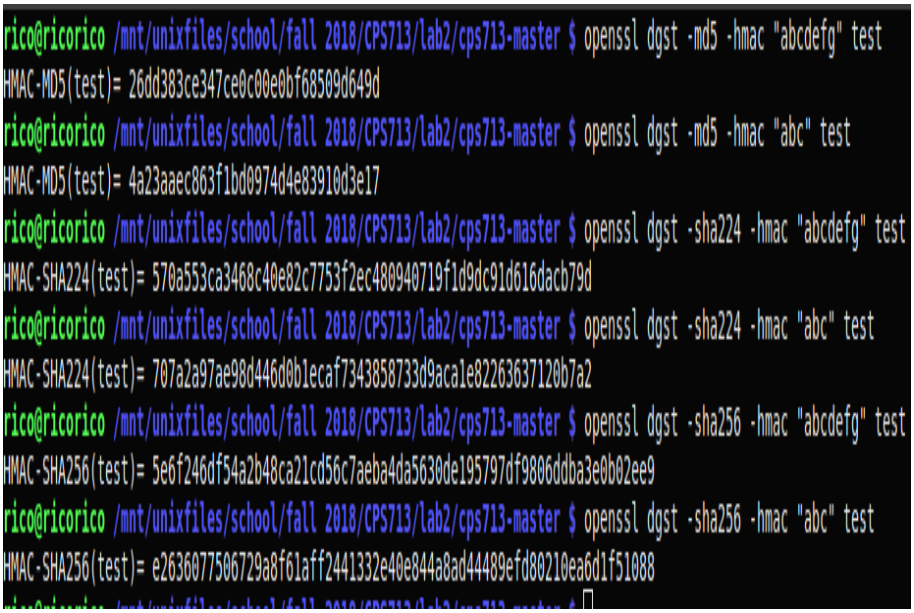
1. Part 1

1.1. Looking at the hashes generated by the 3 different hash functions (md5, sha224, sha256) 2 immediate observations can be made: First, when the file is non-empty, using a different hash function will result in a completely different hash, with no obvious correlation between the different outputted hashes visible to an external user. However, when the file is empty, both the sha-224 and md5 hash functions produce hashes that contain the characters d, 1, and 4 in their first three characters. In addition, each of the above hash functions creates a seemingly random and unpredictable hash from the input, making it impossible to predict an input given the hash (One-way property). Secondly the output of each hash function is a string of characters one quarter the length of their defined bit output. This is because these outputs are not chars they are hexadecimal numbers, where each hexadecimal digit represents 4 binary bits. Therefore each hash function outputs a hash the exact length of its defined output bit length (128 bits for md5, 224 bits for sha-224, and 256 bits for sha-256).





1.2. The HMAC-key can be of any length. The reason for this is that the key itself becomes hashed, and hashes operate by compressing data into blocks. However, it is not recommended to input a key that has a smaller length than the hashing algorithms predefined output hash length. This is because the key will have to be zero padded by the MAC to produce the required length, thereby reducing the entropy of the hashing function. Likewise it is unnecessary to provide a key longer than the hashing functions output length, as the extra bits will simply be unused. [1]



1.3. In the image below, an empty file was hashed and flipping the last bit had no effect, a markedly different case than when a non-empty file was hashed.

```
rico@ricorico /mnt/unixfiles/school/fall 2018/CPS713/lab2/cps713NEW/build $ cat test
rico@ricorico /mnt/unixfiles/school/fall 2018/CPS713/lab2/cps713NEW/build $ ./task3 test
Filename given: test
MD5 hash for input file [H1] --> d41d8cd98f00b204e9800998ecf8427e
Flipping Last bit ...
MD5 hash for input file [H2] --> d41d8cd98f00b204e9800998ecf8427e
Count of similar bits: 128
rico@ricorico /mnt/unixfiles/school/fall 2018/CPS713/lab2/cps713NEW/build $ echo a >> test
rico@ricorico /mnt/unixfiles/school/fall 2018/CPS713/lab2/cps713NEW/build $ cat test
a
rico@ricorico /mnt/unixfiles/school/fall 2018/CPS713/lab2/cps713NEW/build $ ./task3 test
Filename given: test
MD5 hash for input file [H1] --> 60b725f10c9c85c70d97880dfe8191b3
Flipping Last bit ...
MD5 hash for input file [H2] --> 0d95b2ba90e1939704982df868614a4b
Count of similar bits: 0
rico@ricorico /mnt/unixfiles/school/fall 2018/CPS713/lab2/cps713NEW/build $
```

1.4. The mean number of attempts, after 5 attempts, to break the collision property was 8242327.8 To break the one-way property, the mean number of attempts was 7271973. Thus, the one-way property was observed to require less attempts. However, the variance in the number of attempts made to find the collisions for both properties was very large, and therefore its is safe to assume that they both require a nearly equal number of trials to break via brute force. From a mathematical/logical perspective it would be easier to break the collision free property because of the birthday paradox; It is simply easier to find any two inputs to hash to the same value, in comparison to finding an input that matches the exact hash of our original input.

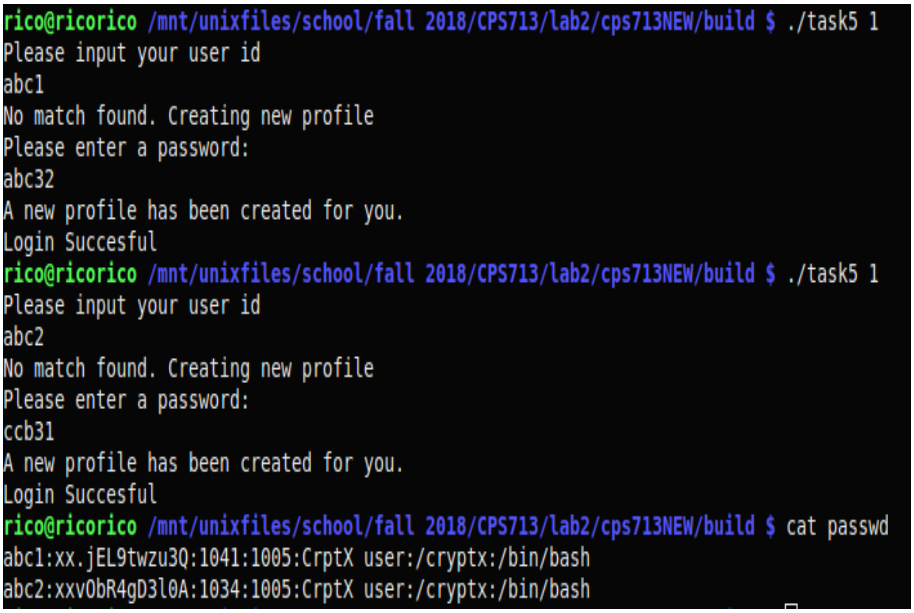
```
rico@ricorico /mnt/unixfiles/school/fall 2018/CPS713/lab2/cps713NEW/build $ ./task4 1f8a24
====Testing Collision free property====
Finding a collision can take a while, please be patient.
Initial random string cuihshktjhuqobpsognttfbbwascmy
SHA1 hash for initial string --> 864f
Found string with matching hash: bnvftxviprubyyhkrabxpwxwbrpnau : 864f
Number of trials to find a hash collision: 9140866

====Testing one way property====
Looking for hash: 1f8a24

Found plain text with matching hash: rsydwrgfkdsqwsqlxuetrhgfnseyqvf : 1f8a24
Number of trials to find a hash collision: 16929537
```

2. Second Part

2.1. 5. Passwords were selected within the range of values: [aa111,aaa112,...,ccc332,cc333] to limit the time and space-complexity of tasks 6 and 7. However, we felt that the range of values was sufficiently large to ensure that finding the password would be a non-trivial pursuit by hand and would thus necessitate the use of some sort of automated approach.



2.2. 6. The command used was:

crunch 5 5 abc + 123 -t @@@%% -wordlist.txt

5 5 Indicates that the minimum length of a word in the list is 5 and that the maximum length is also 5. Thus, all words generated by crunch 5 characters in length.
abc + 123 Indicates that the words are to be composed of the alphanumeric characters a, b, c, 1, 2, and 3.
-t @@@%% Indicates that the words take the general form of starting with 3 letters and ending with two numbers.
-wordlist.txt Saves the generated words in a file called wordlist.txt

```
rico@ricorico /mnt/unixfiles/school/fall 2018/CPS713/lab2/cps713NEW/build $ cd /mnt/unixfiles/crunch-3.6/
rico@ricorico /mnt/unixfiles/crunch-3.6 $ ./crunch 5 5 abc + 123 -t @@@%% -o /mnt/unixfiles/school/fall\ 2018/CPS713/lab2/cps7
13NEW/build/wordlist.txt
Crunch will now generate the following amount of data: 1458 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 243

crunch: 100% completed generating output
rico@ricorico /mnt/unixfiles/crunch-3.6 $ cd /mnt/unixfiles/school/fall\ 2018/CPS713/lab2/cps713NEW/build/
rico@ricorico /mnt/unixfiles/school/fall 2018/CPS713/lab2/cps713NEW/build $ cat wordlist.txt | more
aaa11
aaa12
aaa13
aaa21
aaa22
aaa23
aaa31
aaa32
aaa33
```

2.3. 7. The command used was:

john –wordlist=wordlist.txt passwd

–**wordlist=wordlist.txt** Initiates the wordlist function of John the Ripper using wordlist.txt. It took John the Ripper less than one second to find the correct passwords.

```
rico@ricorico /mnt/unixfiles/school/fall 2018/CPS713/lab2/cps713NEW/build $ john --wordlist=wordlist.txt passwd
Loaded 2 password hashes with no different salts (descrypt, traditional crypt(3) [DES 128/128 SSE2-16])
Press 'q' or Ctrl-C to abort, almost any other key for status
abc32            (abc1)
ccb31            (abc2)
2g 0:00:00:00 100% 16.66g/s 2025p/s 2025c/s 3091C/s bbc13..ccc33
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

3. Word Problems

3.1. One way that a cryptographic hash function differs from a non-cryptographic hash function is that both the one-way and two-way properties are explicitly set as criteria for cryptographic hash-functions. The minimum requirement for a hash-function is to produce a fixed-size value given a message. It need not be the case that, given a hash-value h, it is difficult to find a message m such that, given a non-cryptographic hash-function H, H(m) = h. Non-cryptographic hash functions could be very good for ensuring data integrity in certain situations, such as when the attack surface of a system is very small (for instance,

suppose I am the only one accessing my computer, I can make a very basic non-cryptographic hash that functions by XOR-ing the bits. In such a case, should the hash change, I could surmise that my data storage hardware has been compromised.).

3.2. CryptE could be improved by allowing or requiring for additional characters to be used; For instance, it could be required that passwords contain at least one upper case leat, one lower-case letter, and one number. CryptE could also be improved by using a newer hash algorithm; Even SHA1 would be a marked improvement over the DES function. CryptE could also be improved by requiring additional authorization (such as email verification or a secret question and answer), or by enciphering the passwd file.

3.3. It was observed that using crypt() with "xxxx" as a salt produced the same digest as using crypt() with "xx" as a salt. Thus, we infer that DES only uses the first two letters of the salt. To confirm this, we attempted to run John twice but with two different salt values: "xx" and "xxxx" (see johnTester.sh). When running it with the latter salt value, John reported completing 736.3 c/s (that is, 736 username/password combinations were tried per second). When running it with the former, John reported completing 1012 c/s. However, we ran it a few more times and noticed that there were times when the computation speed was higher with the longer salt value than the shorter value. To explore this seeming contradiction, we ran task5 twice to input two different username with two different passwords into passwd, but with the different salt values of 'xx' and 'xxxx'. When we ran John The Ripper, the output read: Loaded 2 password hashes with no different salts. Thus, it does not seem that John The Ripper differentiates between 'xx' and 'xxxx' when the crypt() function is utilized. I infer from this that increasing the size of the salt will not result in better security.

3.4. A typical time period will depend upon a variety of factors. First, the usability and desired security of the platform has to be taken into account: A very short time-frame may be a nuisance if we are concerned with the lock-screen of a cell-phone, but it would be a welcome parameter for something like accessing nuclear launch codes. Furthermore, the difficulty of inputting the password or authentication has to be taken into account: Many services require email verification, so that if a user attempts to log into the service from an unrecognized IP they must access their email. If the time-frame is very short, a user may have to access their email multiple times, which could be very annoying for the user. Generally, there seems to be a trade-off between security and usability or convenience. An individual may disable security features if they find them to be bothersome so it could paradoxically result in better security to implement shorter time-frames.

References

1. M. Krawcyzyk, M. Bellare, R. Canetti, 2017: HMAC: Keyed-Hashing for Message Authentication. https://tools.ietf.org/html/rfc2104#section-3

Department of Computer Science, Ryerson University, Toronto, ON, Canada
E-mail address: matthew.francis@ryerson.ca / 500462062

Department of Mathematics, Ryerson University, Toronto, ON, Canada
E-mail address: reaz.huq@ryerson.ca / 500392795