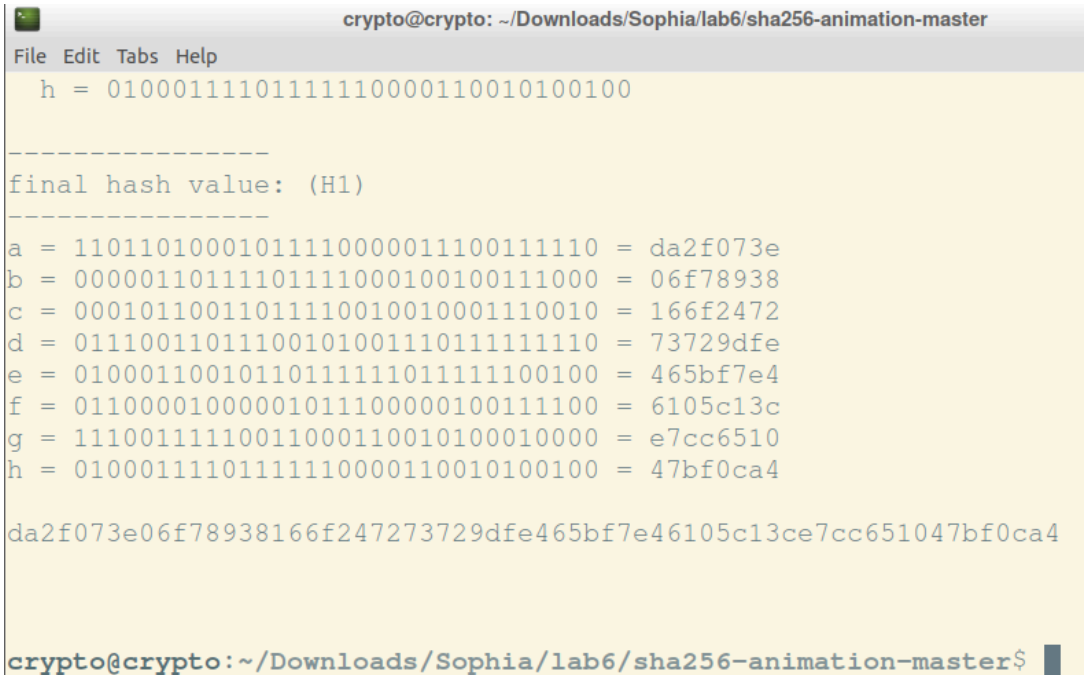
One way hash functions

SHA-512 hash function

Task 1 - Details of SHA-256/512 algorithm

- SHA-256 = produces 256 bit (32 byte) hash
- SHA 512 = produces 512 bit (64 byte) hash

- Steps:
    1. Preprocessing
        - Padding msg - add 1 bit, add 0s for specified msg length
            - SHA-256 = 448 mod 512
            - SHA-512 = 896 mod 1024
        - Parsing msg - breaking down msg
            - SHA 256 = 512 bit blocks
            - SHa 512 = 1024 bit blocks
    2. Set initial hash values
    3. Generate msg schedule - block
    4. Initialise 8 values (a-h) with current hash value
        - SHA 256 = 64 rounds
        - SHA 512 = 80 rounds




```
crypto@crypto:~/Downloads/Sophia/lab6/sha256-animation-master$ echo -n crypto |
openssl dgst -sha256
(stdin)= da2f073e06f78938166f247273729dfe465bf7e46105c13ce7cc651047bf0ca4
```

Task 2 - SHA-512 on a small file

```
crypto@crypto:~/Downloads/Sophia/lab6$ echo 0 > sfile
crypto@crypto:~/Downloads/Sophia/lab6$ cat sfile
0
crypto@crypto:~/Downloads/Sophia/lab6$ openssl dgst -sha512 sfile
SHA512(sfile)= a546d1300f49037a465ecec8bc1ebd07d57015a5ff1abfa1c94da9b30576933fb68e38
98ff764d4de6e6741da822a7c93adc6e845806a266a63aa14c8bb09ebb
crypto@crypto:~/Downloads/Sophia/lab6$
```

Q: How long is the SHA-512 hash value?

SHA 512 = produces 512 bit (64 byte) hash

Task 3 - SHA-512 on a large file

```
crypto@crypto:~/Downloads/Sophia/lab6$ head -c 10M /dev/urandom > lfile
crypto@crypto:~/Downloads/Sophia/lab6$ ll
total 17496
drwxrwxr-x 3 crypto crypto     4096 Apr 27 14:52 ./
drwxrwxr-x 4 crypto crypto     4096 Apr 27 14:17 ../
-rw-rw-r-- 1 crypto crypto 10485760 Apr 27 14:52 lfile
-rwxrw-rw- 1 crypto crypto   540287 Apr 27 14:18 MD5.pdf*
-rwxrw-rw- 1 crypto crypto    16164 Apr 27 14:18 password.lst*
-rwxrw-rw- 1 crypto crypto    11434 Apr 27 14:18 python-md5-collision-master.zip*
-rwxrw-rw- 1 crypto crypto   223901 Apr 27 14:18 Salt_1.pdf*
-rwxrw-rw- 1 crypto crypto  1146348 Apr 27 14:18 Salt_2.pdf*
-rw-rw-r-- 1 crypto crypto        2 Apr 27 14:46 sfile
drwxrwxr-x 3 crypto crypto     4096 Apr 27 14:45 sha256-animation-master/
-rwxrw-rw- 1 crypto crypto  5464529 Apr 27 14:18 sha256-animation-master.zip*
-rwxrw-rw- 1 crypto crypto      265 Apr 27 14:18 sha512crack.sh*
crypto@crypto:~/Downloads/Sophia/lab6$ openssl dgst -sha512 lfile
SHA512(lfile)= d0c09d0bdd262b82c1ceda69296973d46a4473fdb83d6edd851717d291a801148d86f2
881445a15d666b96c3abff28c70dfe95673b552dbc558c0bbe9f8fbc93
crypto@crypto:~/Downloads/Sophia/lab6$
```

Q: How long is the SHA-512 hash value?

SHA 512 = produces 512 bit (64 byte) hash - regardless of file size

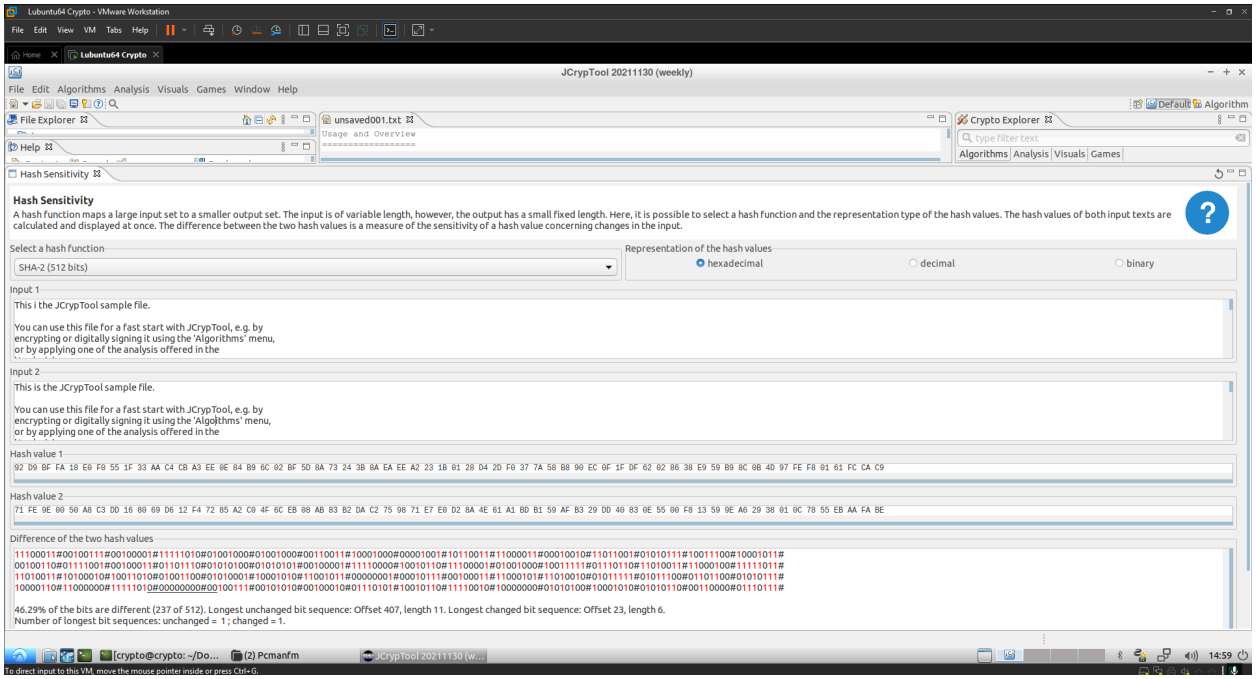
Task 4 - Change the file and redo the SHA-512

```
crypto@crypto:~/Downloads/Sophia/lab6$ echo -n "1" >> lfile
crypto@crypto:~/Downloads/Sophia/lab6$ ll
total 17500
drwxrwxr-x 3 crypto crypto     4096 Apr 27 14:52 ./
drwxrwxr-x 4 crypto crypto     4096 Apr 27 14:17 ../
-rw-rw-r-- 1 crypto crypto 10485761 Apr 27 14:56 lfile
-rwxrw-rw- 1 crypto crypto   540287 Apr 27 14:18 MD5.pdf*
-rwxrw-rw- 1 crypto crypto    16164 Apr 27 14:18 password.lst*
-rwxrw-rw- 1 crypto crypto    11434 Apr 27 14:18 python-md5-collision-master.zip*
-rwxrw-rw- 1 crypto crypto   223901 Apr 27 14:18 Salt_1.pdf*
-rwxrw-rw- 1 crypto crypto  1146348 Apr 27 14:18 Salt_2.pdf*
-rw-rw-r-- 1 crypto crypto        2 Apr 27 14:46 sfile
drwxrwxr-x 3 crypto crypto     4096 Apr 27 14:45 sha256-animation-master/
-rwxrw-rw- 1 crypto crypto  5464529 Apr 27 14:18 sha256-animation-master.zip*
-rwxrw-rw- 1 crypto crypto      265 Apr 27 14:18 sha512crack.sh*
crypto@crypto:~/Downloads/Sophia/lab6$ openssl dgst -sha512 lfile
SHA512(lfile)= 4f32340461e75e74d2b7ed175fee0dd729f4bd10a08145a284d0aa9cd4cd87cad657a2
a8d56e60ab698ac66c335e43107aef24579d2c262e1bd2ea103a6e57b9
crypto@crypto:~/Downloads/Sophia/lab6$
```

Q: How long is the SHA-512 hash value? Is this hash value significantly different from the last hash value?

SHA 512 = produces 512 bit (64 byte) hash - regardless of file size

Task 5 - Hash sensitivity visualisation using JCrypTools



Q: Comment the sensitivity of hash functions to the input. Why can hash functions verify data integrity?

- Very sensitive to input changes - avalanche effect
- Hash functions can verify data integrity as it easily reveals tampering - hash is original if the data remains unchanged

Hash crack

Task 1 - Brute-force attack against the preimage resistant

- Preimage resistance - hash function property hard to invert (can't find input)

```
crypto@crypto:~/Downloads/Sophia/lab6$ chmod +x sha512crack.sh
crypto@crypto:~/Downloads/Sophia/lab6$ ./sha512crack.sh ab1c2b83a2d9b0304541de63f302a
9580c0b4252dcd70d5a6ce0bd5b8829d2b3bc1f727c70140113071bcc7f16a737a74a6f0f5166168185dd
43bbba690b5041
```

Q: What is the SHA-512 preimage of "ab1c2b83a2d9b0304541de63f302a9580c0b4252dcd70d5a6ce0bd5b8829d2b3bc1f727c70140113071bcc7f16a737a74a6f0f5166168185dd43bbba690b5041"?

```
try diction ab1c2b83a2d9b0304541de63f302a9580c0b4252dcd70d5a6ce0bd5b882
0140113071bcc7f16a737a74a6f0f5166168185dd43bbba690b5041
plaintext is diction
search end
crypto@crypto:~/Downloads/Sophia/lab6$
```

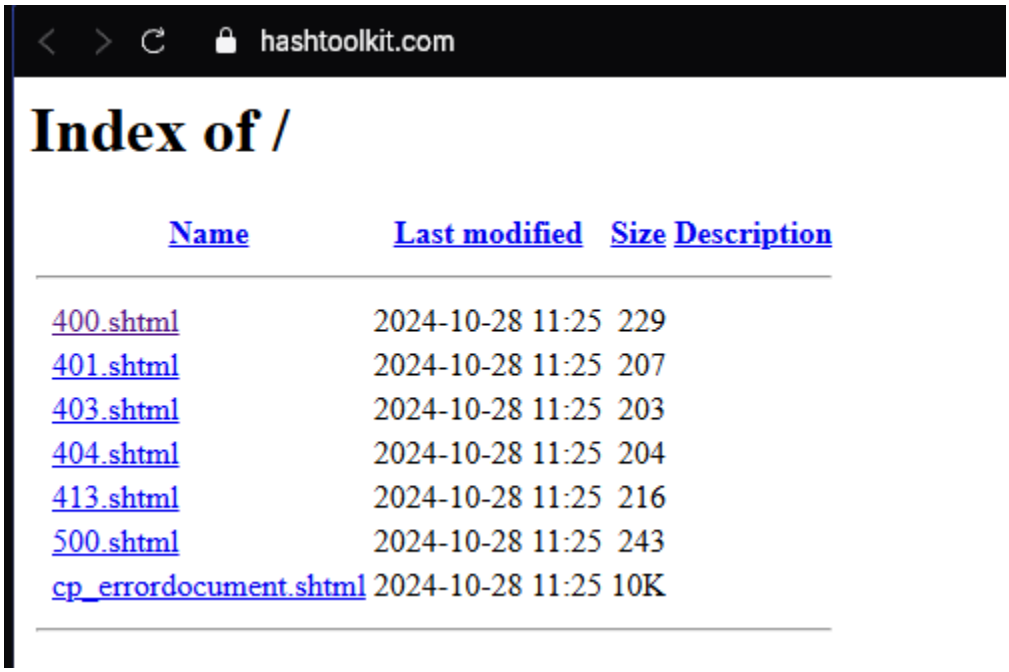
Q: How does the brute-force crack work?

- Goes through every input
- Hashing the input using SHA-512
- Comparing the result with the targeted hash
- Repeating the process until match found

Task 2 - SHA-512 crack with rainbow table against the preimage resistant

- Rainbow table - precomputed table for caching the output of cryptographic hash functions - cracks password hashes
- Used in recovering key derivation function - uses more storage but less processing time

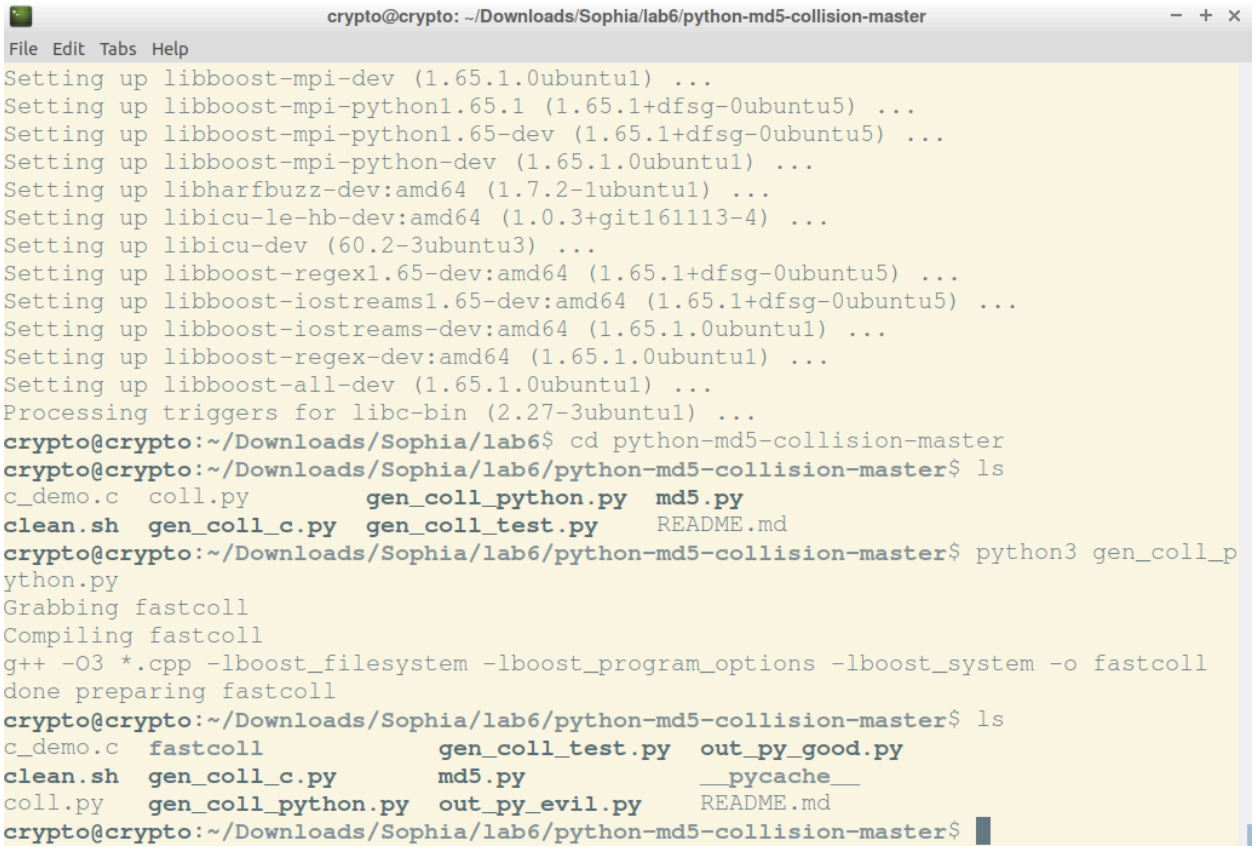
Q: What is the SHA-512 Preimage of "21fc9c65b173371a479de565a9a529c2aedefea54db72b7b7718e786dbb1265c4517d1c71237f566e98ab961d0b49e5ae4462099735122c61c8f7ab37ac5c389"?

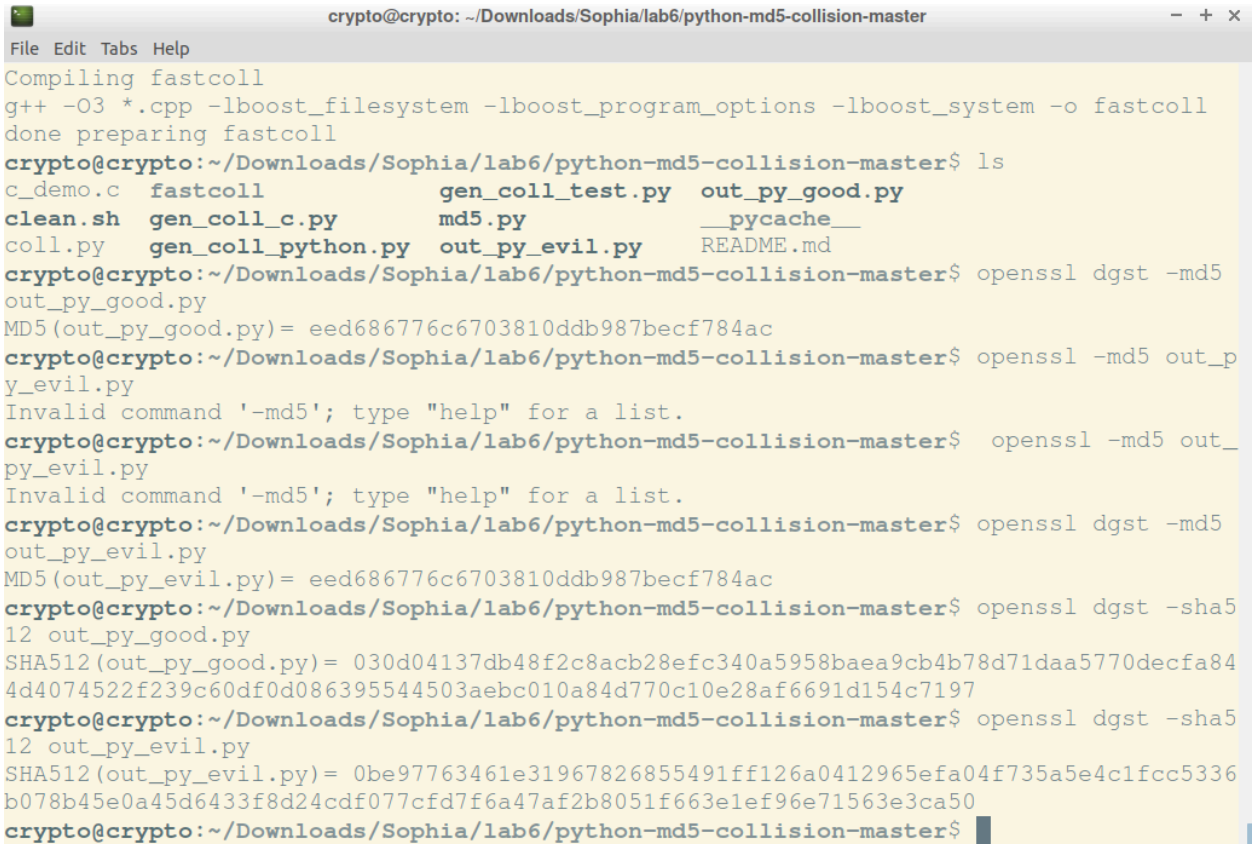


- Can't reverse hash of SHA-512 unless listed in rainbow table^

Task 3 - MD5 collision attack against the strong collision resistant

- MD5 message-digest algorithm - produces 128 bit hash value
- Suffers from vulnerabilities

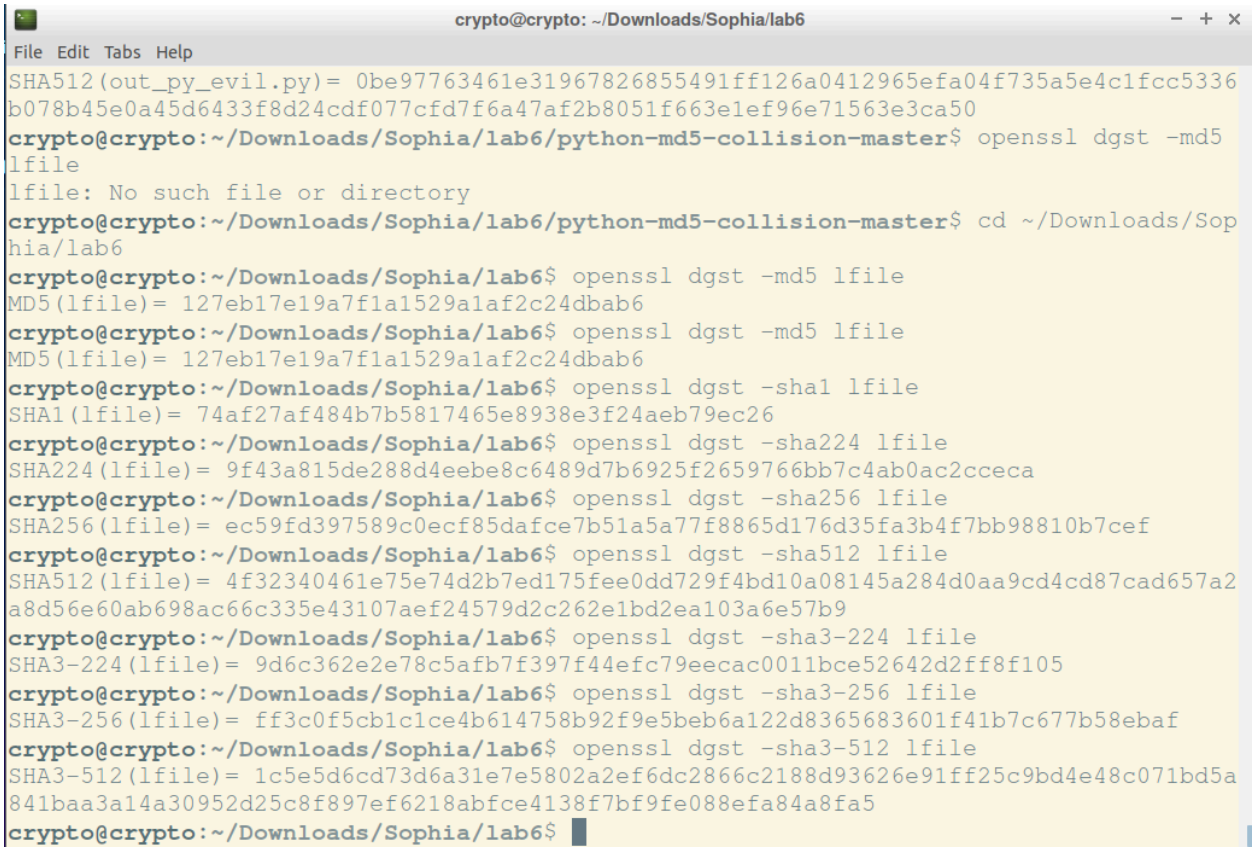




Q: Suppose an adversary uses brute-force to attack hash functions. For a general hash function with m-bit hash value, how many times should an adversary try to find the preimage of a given hash value? How many times should an adversary try to find two preimages that generate the same hash value?

- How many times adversary should try to find preimage = 2^m on avg
    - M = bits
- How many times adversary should try to find 2 preimages that generates the same hash value = 2^m/2
    - Attempts lowered - bday paradox

Other hash functions



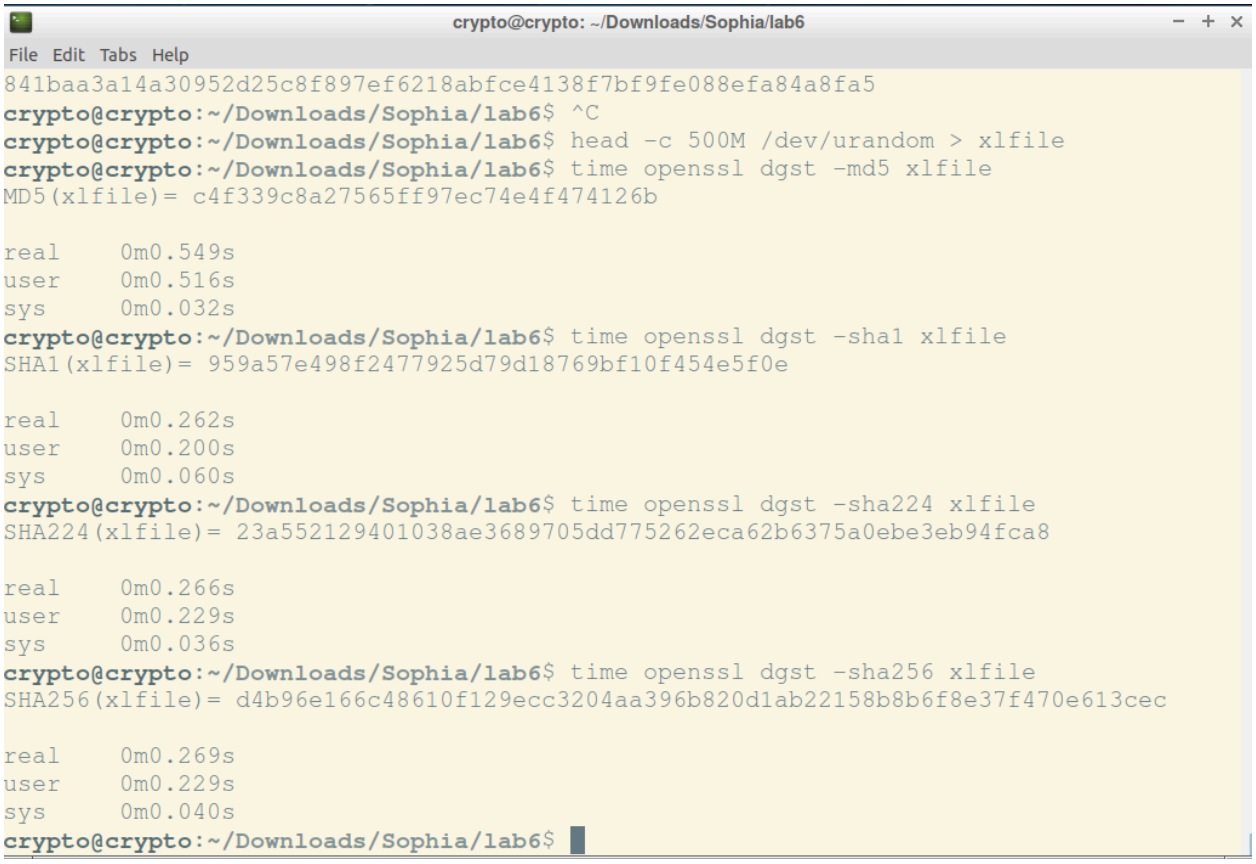
Q: What are the lengths of the hash values?

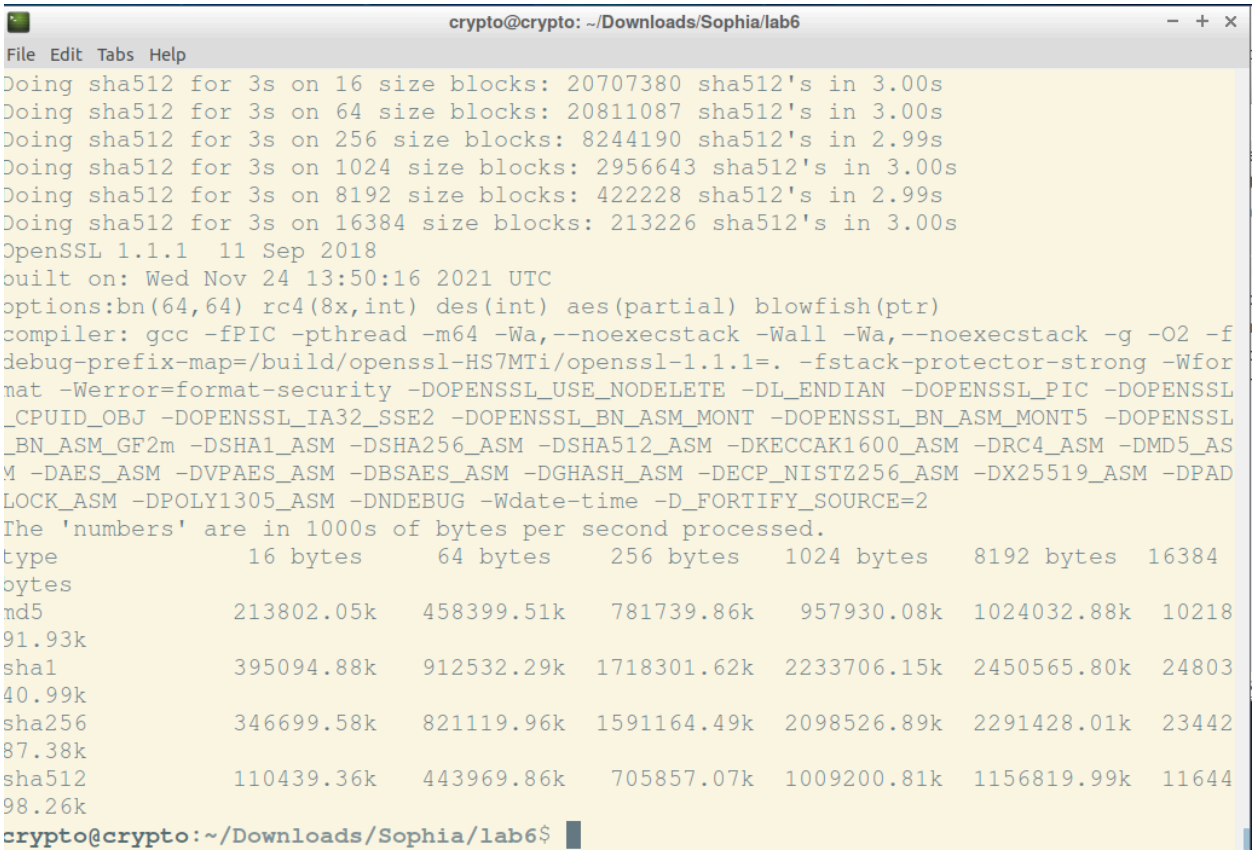
1. MD5 = 12 bits
2. SHA-1 = 160 bits
3. SHA-224 = 224 bits
4. SHA-256 = 256 bits
5. SHA-512 = 512 bits
6. SHA3-224 = 224 bits
7. SHA3-256 = 256 bits
8. SHA3-512 = 512 bits

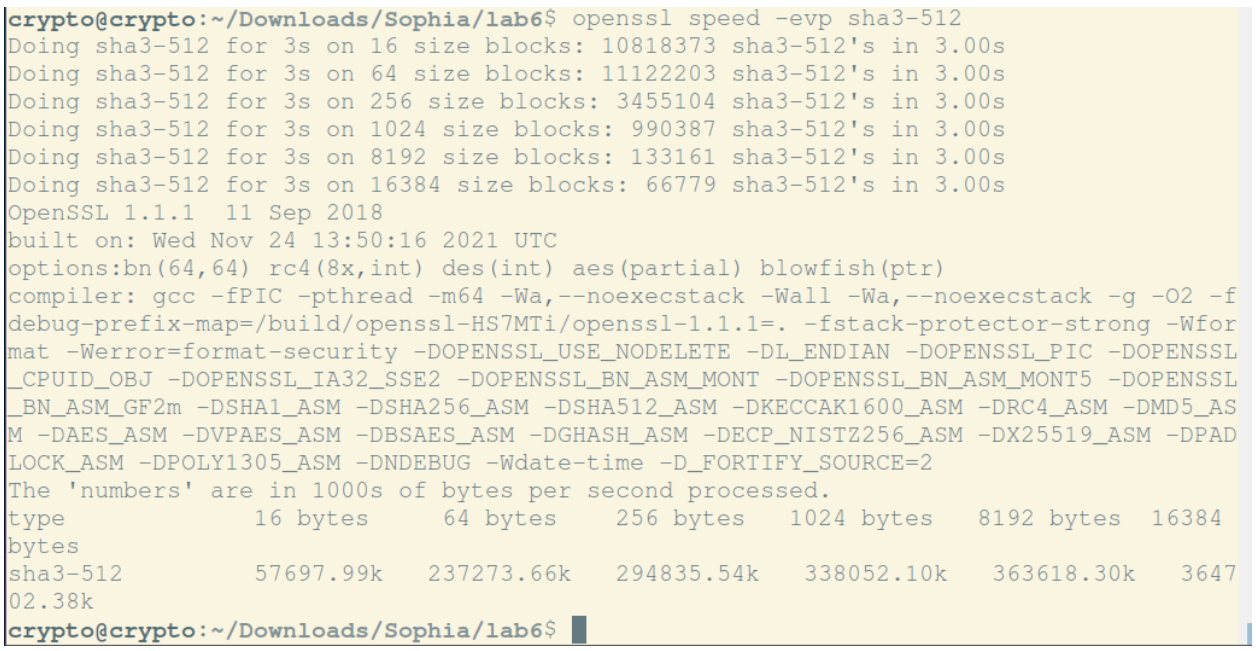
Q: Compare the running time and rank the speed of hash functions, i.e., md5, sha1, SHA-2(sha224, sha256 and sha512) and SHA-3(sha3-224, sha3-256 and sha3-512)? Which one is the fastest?

1. MD5 = fast, insecure
2. SHA-1 = slightly slow, insecure
3. SHA-224 and SHA-256 = similar speed, SHA-256 slightly slower
4. SHA-512 = uses 64-bit block processing, slower
5. SHA3-224 and SHA3-256 = slower, secure and future-proof

6. SHA3-512 = slowest, Keccak algorithm and larger block size




crypto@crypto: ~/Downloads/Sophia/lab6
File Edit Tabs Help
Doing sha512 for 3s on 16 size blocks: 20707380 sha512's in 3.00s
Doing sha512 for 3s on 64 size blocks: 20811087 sha512's in 3.00s
Doing sha512 for 3s on 256 size blocks: 8244190 sha512's in 2.99s
Doing sha512 for 3s on 1024 size blocks: 2956643 sha512's in 3.00s
Doing sha512 for 3s on 8192 size blocks: 422228 sha512's in 2.99s
Doing sha512 for 3s on 16384 size blocks: 213226 sha512's in 3.00s
OpenSSL 1.1.1 11 Sep 2018
built on: Wed Nov 24 13:50:16 2021 UTC
options:bn(64,64) rc4(8x,int) des(int) aes(partial) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack -g -O2 -f
debug-prefix-map=/build/openssl-HS7MTi/openssl-1.1.1=. -fstack-protector-strong -Wfor
mat -Werror=format-security -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL
_CPUID_OBJ -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL
_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_AS
M -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPAD
LOCK_ASM -DPOLY1305_ASM -DNDEBUG -Wdate-time -D_FORTIFY_SOURCE=2
The 'numbers' are in 1000s of bytes per second processed.
type 16 bytes 64 bytes 256 bytes 1024 bytes 8192 bytes 16384
bytes
md5 213802.05k 458399.51k 781739.86k 957930.08k 1024032.88k 10218
91.93k
sha1 395094.88k 912532.29k 1718301.62k 2233706.15k 2450565.80k 24803
40.99k
sha256 346699.58k 821119.96k 1591164.49k 2098526.89k 2291428.01k 23442
87.38k
sha512 110439.36k 443969.86k 705857.07k 1009200.81k 1156819.99k 11644
98.26k
crypto@crypto:~/Downloads/Sophia/lab6$


crypto@crypto:~/Downloads/Sophia/lab6$ openssl speed -evp sha3-512
Doing sha3-512 for 3s on 16 size blocks: 10818373 sha3-512's in 3.00s
Doing sha3-512 for 3s on 64 size blocks: 11122203 sha3-512's in 3.00s
Doing sha3-512 for 3s on 256 size blocks: 3455104 sha3-512's in 3.00s
Doing sha3-512 for 3s on 1024 size blocks: 990387 sha3-512's in 3.00s
Doing sha3-512 for 3s on 8192 size blocks: 133161 sha3-512's in 3.00s
Doing sha3-512 for 3s on 16384 size blocks: 66779 sha3-512's in 3.00s
OpenSSL 1.1.1 11 Sep 2018
built on: Wed Nov 24 13:50:16 2021 UTC
options:bn(64,64) rc4(8x,int) des(int) aes(partial) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack -g -O2 -f
debug-prefix-map=/build/openssl-HS7MTi/openssl-1.1.1=. -fstack-protector-strong -Wfor
mat -Werror=format-security -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL
_CPUID_OBJ -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL
_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_AS
M -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPAD
LOCK_ASM -DPOLY1305_ASM -DNDEBUG -Wdate-time -D_FORTIFY_SOURCE=2
The 'numbers' are in 1000s of bytes per second processed.
type 16 bytes 64 bytes 256 bytes 1024 bytes 8192 bytes 16384
bytes
sha3-512 57697.99k 237273.66k 294835.54k 338052.10k 363618.30k 3647
02.38k
crypto@crypto:~/Downloads/Sophia/lab6$

Securing hash functions
Task 1 - Salt

- Salt - random data (password) used as an additional input before hashing, makes each one different
- Safeguards passwords in storage

Q: How does the salt scheme improve the security of hash algorithms?

- Prevents rainbow table attacks - hash values for common passwords
- Prevents dictionary attacks - uses common words lists for password guessing
- Prevents parallel processing attacks - can't check multiple hashes in parallel (expensive for brute force attacks

Q: How to use the salt scheme?

- Salt randomly generated for each password (no reusing)
- salt + password = hash
- Store the salt and hash
- Retrieve salt and hash for verification: match both with input password that has been combined with salt and hashed

Task 2 - Strong Hash Functions

Q: What is the vulnerability of MD5 hash function?

- Collision attacks - generating 2 different inputs that produces same hash, affecting system integrity
- Preimage attacks - finding input through exploiting hash output
- Brute force attacks - MD5 is fast, but insecure

Q: Which hash functions are recommended now?

- SHA-2 and SHA-3
    - Resistant due to high computational cost