Hashing Algorithms and HMAC

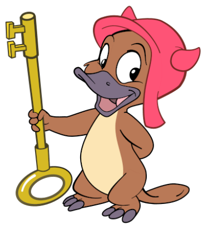
Lab Goal: Exploring hashing algorithms and HMAC.

Lab Requirements:

- md5sum
- OpenSSL package
- Unix or Linux operating system

Lab is tested on Darwin OS, Debian 12 and Kali.

HASH BASED
MESSAGE
AUTHENTICATION
CODE



Cryptographic hash function (Hashing)

“Cryptographic hash functions have many information-security applications, notably in digital signatures, message authentication codes (MACs), and other forms of authentication. They can also be used as ordinary hash functions, to index data in hash tables, for fingerprinting, to detect duplicate data or uniquely identify files, and as checksums to detect accidental data corruption. Indeed, in information-security contexts, cryptographic hash values are sometimes called (digital) fingerprints, checksums, or just hash values, even though all these terms stand for more general functions with rather different properties and purposes.”[i]

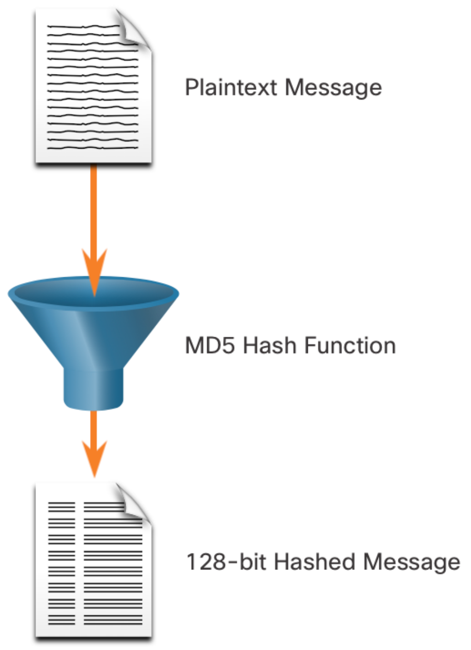
Why we use Hashing

Hashing is utilized in various aspects of IT for its efficiency and security properties. Here are some common uses:

1. **Data Integrity Verification**: When downloading or transferring files, a hash value of the file can be compared with a known hash value to ensure that the file has not been altered or corrupted. This is crucial in verifying the integrity of data.
2. **Digital Signatures**: Hash functions are used in creating digital signatures, which help verify the authenticity of a digital message or document. The hash value of the original message is encrypted with a private key, and this encrypted hash is the digital signature.
3. **Password Storage**: Passwords are stored as hash values rather than plain text. When a user logs in, the entered password is hashed and compared to the stored hash value. This enhances security by ensuring that even if the database is compromised, the actual passwords are not exposed.
4. **Data Structures**: Hashing is used in data structures like hash tables to enable fast data retrieval. Hash tables use hash functions to compute an index into an array of buckets or slots, from which the desired value can be found.

MD5 Hashing algorithm.

- The MD5 (Message Digest Algorithm 5) is developed by Ron Rivest and used in a variety of internet applications,
- It is a one-way function that produces a 128-bit hashed message.is a widely used cryptographic hash function that produces a 128-bit (16-byte) hash value. Despite its historical popularity,
- MD5 is now considered cryptographically broken and unsuitable for further use due to vulnerabilities that allow for collision attacks.
- MD5 processes an input message in 512-bit blocks and produces a 128-bit hash value.



Limitations of MD5

While MD5 was widely used in the past for various applications, it has significant security weaknesses:

1. **Collision Vulnerability**: It is possible to generate two different inputs that produce the same hash value (a collision), which undermines the integrity of the hash function.
2. **Pre-image and Second Pre-image Attacks**: Although less common than collision attacks, MD5 is also susceptible to pre-image and second pre-image attacks.

Due to these vulnerabilities, MD5 is no longer considered secure for cryptographic purposes. It has been largely replaced by more secure hashing algorithms such as SHA-256 and SHA-3. However, MD5 is still used in non-cryptographic contexts, such as checksums for file integrity verification, where collision resistance is less critical.

SHA hashing algorithms family

The SHA (Secure Hash Algorithm) family is a set of cryptographic hash functions designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST). These algorithms are widely used for securing data and ensuring data integrity. The SHA family includes several algorithms with varying levels of security and hash lengths.

Overview of SHA Algorithms

SHA-0:

- Hash Length: 160 bits
- Status: Withdrawn due to significant vulnerabilities
- Details: SHA-0 was the original version published in 1993 but was quickly replaced by SHA-1 due to security weaknesses.

SHA-1:

- Hash Length: 160 bits
- Status: Considered insecure due to vulnerability to collision attacks
- Details: SHA-1 was widely used for many years but is now deprecated for most cryptographic applications.

SHA-2 Family:

- Hash Lengths: Varies (224, 256, 384, 512 bits)
- Variants: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256
- Details: The SHA-2 family was introduced to address vulnerabilities in SHA-1 and provides significantly improved security. These algorithms are widely used in security protocols, including SSL/TLS, and in applications like digital signatures and certificates.

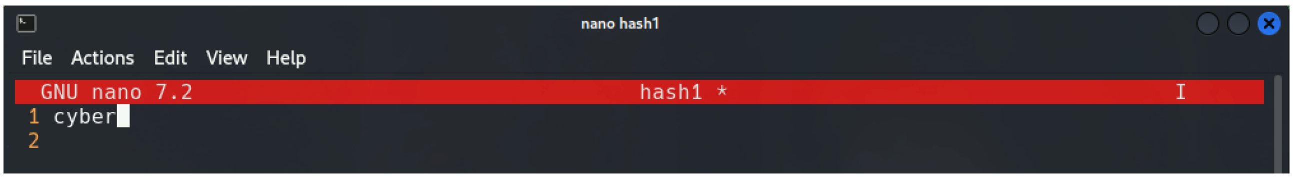
SHA-3 Family:

- Hash Lengths: Varies (224, 256, 384, 512 bits)
- Variants: SHA3-224, SHA3-256, SHA3-384, SHA3-512
- Details: SHA-3 is a completely different algorithm from SHA-2, based on the Keccak algorithm. It was standardized in 2015 and offers a different structural approach to hashing, providing resilience against different types of cryptographic attacks.

Generating HASH from file:

Task 1 - Create file using nano called hash1:

1. that will contain word cyber in it
2. without extra space at end or pressing enter after word
3. Save and exit nano.

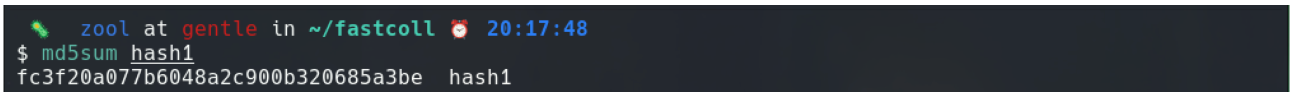


```
 GNU nano 7.2                    hash1 *
cyber
```

Task 2 - Generating MD5 hash from file using md5sum:

To generate the MD5 hash of a file named hash1, you would use the following command:

```
md5sum hash1
```

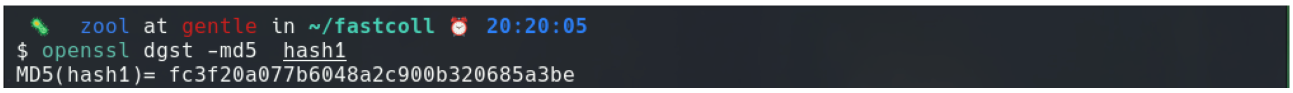


```
  zool at gentle in ~/fastcoll  20:17:48
$ md5sum hash1
fc3f20a077b6048a2c900b320685a3be  hash1
```

Task 3 - Generating MD5 hash from file using OpenSSL.

The openssl dgst command is another way to generate MD5 hashes, as well as other types of message digests, using the OpenSSL toolkit. To generate the MD5 hash of a file named hash1, you can use the following command:

```
openssl dgst -md5 hash1
```

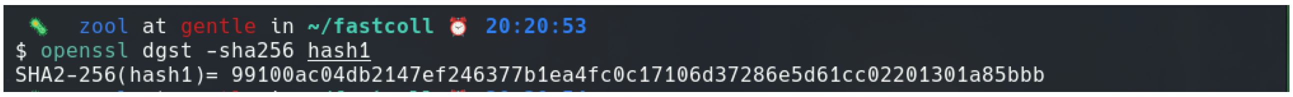


```
  zool at gentle in ~/fastcoll  20:20:05
$ openssl dgst -md5  hash1
MD5(hash1)= fc3f20a077b6048a2c900b320685a3be
```

Task 4 - Generating SHA-256 hash from file using OpenSSL.

To generate the sha256 hash of a file named hash1, you can use the following command:

```
openssl dgst -sha256 hash1
```

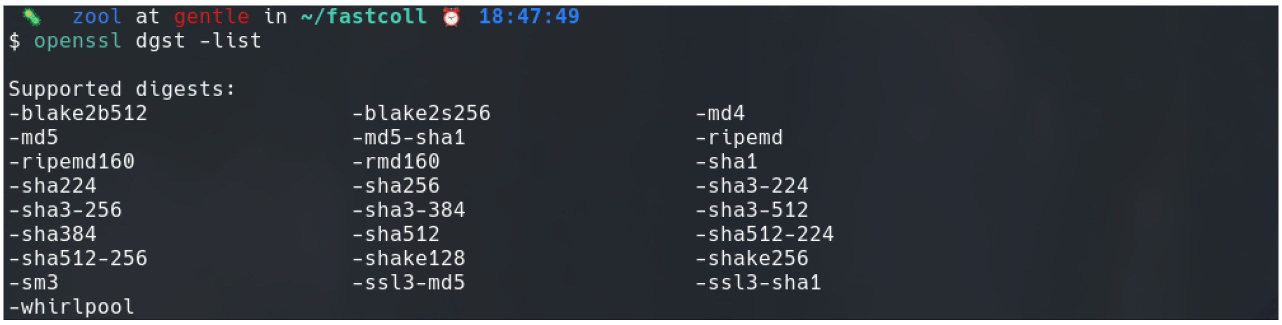


```
  zool at gentle in ~/fastcoll  20:20:53
$ openssl dgst -sha256 hash1
SHA2-256(hash1)= 99100ac04db2147ef246377b1ea4fc0c17106d37286e5d61cc02201301a85bbb
```

Task 5 - Listing all hashing algorithms supported by OpenSSL.

To see list of all **hashing** algorithms supported by OpenSSL issue following command:

```
openssl dgst -list
```



```
  zool at gentle in ~/fastcoll  18:47:49
$ openssl dgst -list

Supported digests:
-blake2b512                -blake2s256                -md4
-md5                       -md5-sha1                  -ripemd
-ripemd160                 -rmd160                    -sha1
-sha224                    -sha256                    -sha3-224
-sha3-256                  -sha3-384                  -sha3-512
-sha384                    -sha512                    -sha512-224
-sha512-256                -shake128                  -shake256
-sm3                       -ssl3-md5                  -ssl3-sha1
-whirlpool
```

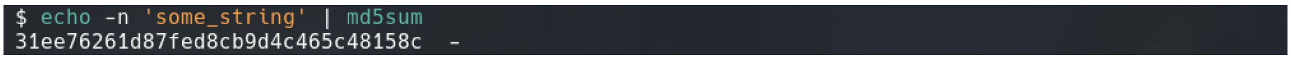
Generating HASH from string:

Task 1 - Generating MD5 hash from string using md5sum and OpenSSL:

NOTE: echo will normally output a newline, which is suppressed with -n

The command echo -n "some_string" | md5sum generates the MD5 hash of the string some_string.

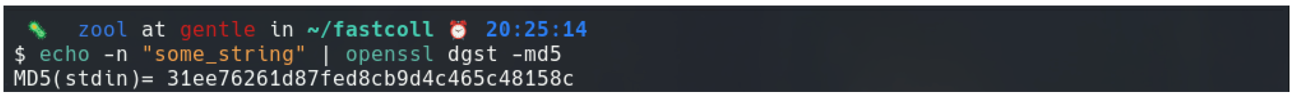
```
echo -n "some_string" | md5sum
```



```
$ echo -n 'some_string' | md5sum
31ee76261d87fed8cb9d4c465c48158c  -
```

The command **echo -n "some_string" | openssl dgst -md5** generates the MD5 hash of the string some_string.

```
echo -n "some_string" | openssl dgst -md5
```

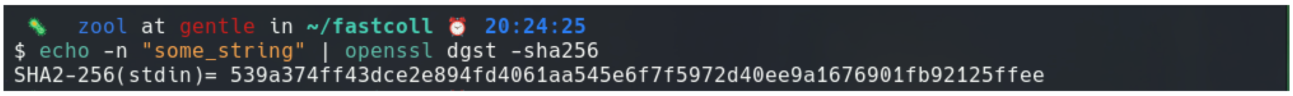


```
   zool at gentle in ~/fastcoll   20:25:14
$ echo -n "some_string" | openssl dgst -md5
MD5(stdin)= 31ee76261d87fed8cb9d4c465c48158c
```

Task 2 - Generating SHA-256 hash from string using OpenSSL:

The command below generates the sha256 hash of the string some_string.

```
echo -n "some_string" | openssl dgst -sha256
```



```
   zool at gentle in ~/fastcoll   20:24:25
$ echo -n "some_string" | openssl dgst -sha256
SHA2-256(stdin)= 539a374ff43dce2e894fd4061aa545e6f7f5972d40ee9a1676901fb92125ffee
```

MD5 collision attack Proof of Concept (POC)

The fastcoll[ii] tool is a program used to generate MD5 hash collisions efficiently. It was developed to demonstrate the vulnerability of the MD5 hashing algorithm to collision attacks.

https://github.com/brimstone/fastcoll

https://www.win.tue.nl/hashclash/

Step 1 – Create file called test.txt

```
echo cyber > test
```

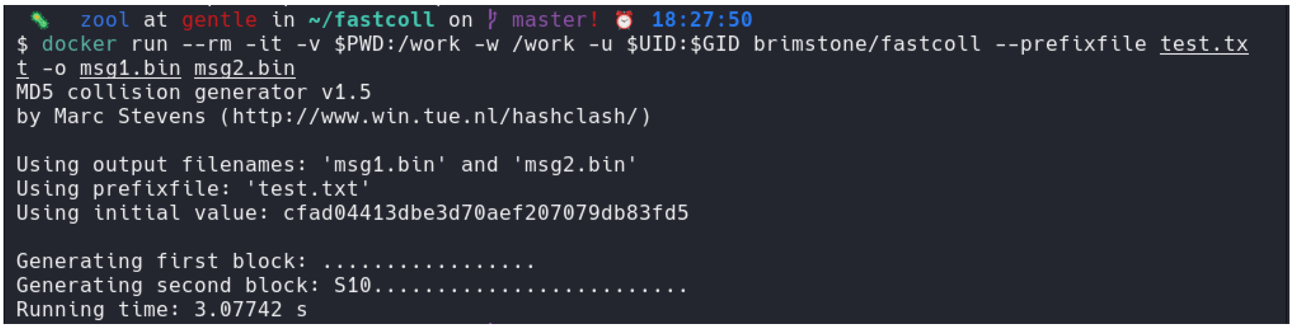
Step 2 – Perform attack using fastcoll docker container

Command below runs the brimstone/fastcoll Docker container to generate two files (msg1.bin and msg2.bin) that have the same MD5 hash, starting from a prefix file (test.txt).

```
docker run --rm -it -v $PWD:/work -w /work -u $UID:$GID brimstone/fastcoll --prefixfile test.txt -o msg1.bin msg2.bin
```

Command Breakdown:

- **docker run**: Runs a command in a new container.
- **--rm**: Automatically removes the container when it exits.
- **-it**: Allocates a pseudo-TTY and keeps the STDIN open, allowing for interactive processes.
- **-v $PWD:/work**: Mounts the current directory ($PWD) on the host to the /work directory in the container. This allows the container to read and write files in the current directory.
- **-w /work**: Sets the working directory inside the container to /work.
- **-u $UID:$GID**: Runs the container as the current user (with the current user's UID and GID), which avoids permission issues when the container creates or modifies files in the mounted directory.
- **brimstone/fastcoll**: The name of the Docker image to run. fastcoll is a tool for generating MD5 collisions.
- **--prefixfile test.txt**: Specifies the prefix file to use as the starting point for the collision generation.
- **-o msg1.bin msg2.bin**: Specifies the output files that will contain the colliding messages.

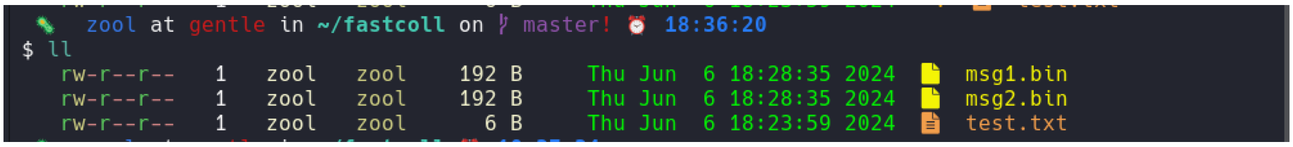


```
   zool at gentle in ~/fastcoll on  master! ⏰ 18:27:50
$ docker run --rm -it -v $PWD:/work -w /work -u $UID:$GID brimstone/fastcoll --prefixfile test.tx
t -o msg1.bin msg2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'msg1.bin' and 'msg2.bin'
Using prefixfile: 'test.txt'
Using initial value: cfad04413dbe3d70aef207079db83fd5

Generating first block: .................
Generating second block: S10........................
Running time: 3.07742 s
```

Fastcoll will use test.txt as input and it will output 2 files, msg1.bin and msg2.bin

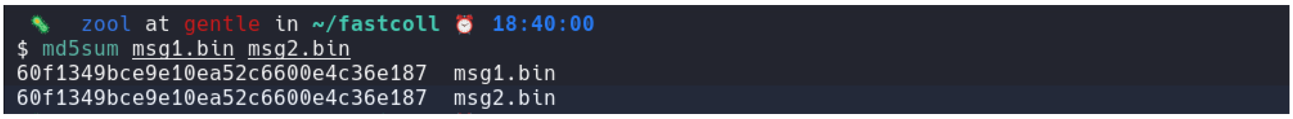


```
   zool at gentle in ~/fastcoll on  master! ⏰ 18:36:20
$ ll
   rw-r--r--   1   zool   zool    192 B     Thu Jun  6 18:28:35 2024    msg1.bin
   rw-r--r--   1   zool   zool    192 B     Thu Jun  6 18:28:35 2024    msg2.bin
   rw-r--r--   1   zool   zool      6 B     Thu Jun  6 18:23:59 2024    test.txt
```

Step 3 – Compare MD5

To verify that the two files (msg1.bin and msg2.bin) generated by the brimstone/fastcoll tool have the same MD5 hash, you can use the md5sum command.

md5sum msg1.bin msg2.bin



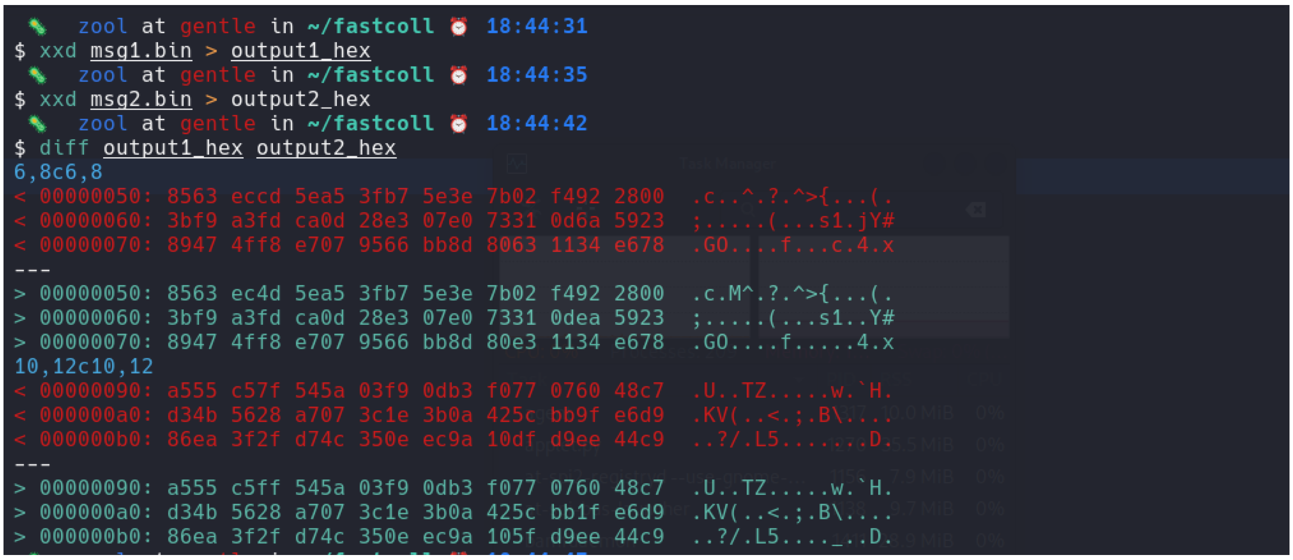
```
   zool at gentle in ~/fastcoll ⏰ 18:40:00
$ md5sum msg1.bin msg2.bin
60f1349bce9e10ea52c6600e4c36e187  msg1.bin
60f1349bce9e10ea52c6600e4c36e187  msg2.bin
```

Step 4 – Examine content of files

The command xxd msg1.bin > output1_hex is used to convert a binary file (out2.bin) into a hexadecimal representation and save the output to a file named output2_hex.

xxd msg1.bin > output1_hex
xxd msg2.bin > output2_hex

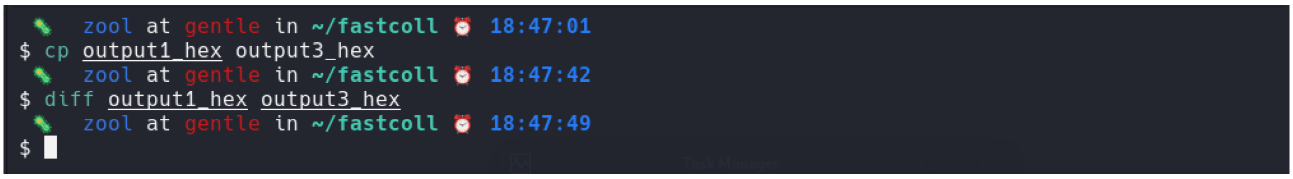
The diff command is used to compare the contents of two files line by line. When applied to the output of two hexadecimal dumps (like output1_hex and output2_hex), it will highlight the differences between the two files.
diff output1_hex output2_hex



```
   zool at gentle in ~/fastcoll ⏰ 18:44:31
$ xxd msg1.bin > output1_hex
   zool at gentle in ~/fastcoll ⏰ 18:44:35
$ xxd msg2.bin > output2_hex
   zool at gentle in ~/fastcoll ⏰ 18:44:42
$ diff output1_hex output2_hex
6,8c6,8
< 00000050: 8563 eccd 5ea5 3fb7 5e3e 7b02 f492 2800  .c..^.?.^>{...(.
< 00000060: 3bf9 a3fd ca0d 28e3 07e0 7331 0d6a 5923  ;.....(...s1.jY#
< 00000070: 8947 4ff8 e707 9566 bb8d 8063 1134 e678  .GO....f...c.4.x
---
> 00000050: 8563 ec4d 5ea5 3fb7 5e3e 7b02 f492 2800  .c.M^.?.^>{...(.
> 00000060: 3bf9 a3fd ca0d 28e3 07e0 7331 0dea 5923  ;.....(...s1..Y#
> 00000070: 8947 4ff8 e707 9566 bb8d 80e3 1134 e678  .GO....f.....4.x
10,12c10,12
< 00000090: a555 c57f 545a 03f9 0db3 f077 0760 48c7  .U..TZ.....w.`H.
< 000000a0: d34b 5628 a707 3c1e 3b0a 425c bb9f e6d9  .KV(..<.;.B\....
< 000000b0: 86ea 3f2f d74c 350e ec9a 10df d9ee 44c9  ..?/.L5.......D.
---
> 00000090: a555 c5ff 545a 03f9 0db3 f077 0760 48c7  .U..TZ.....w.`H.
> 000000a0: d34b 5628 a707 3c1e 3b0a 425c bb1f e6d9  .KV(..<.;.B\....
> 000000b0: 86ea 3f2f d74c 350e ec9a 105f d9ee 44c9  ..?/.L5...._..D.
```

As we can see there are differences.

Let's test is diff working.



```
   zool at gentle in ~/fastcoll ⏰ 18:47:01
$ cp output1_hex output3_hex
   zool at gentle in ~/fastcoll ⏰ 18:47:42
$ diff output1_hex output3_hex
   zool at gentle in ~/fastcoll ⏰ 18:47:49
$ █
```

This part of lab is based on post that can be found on following link. For additional reading please visit it.

Additional collision resources: https://github.com/corkami/collisions

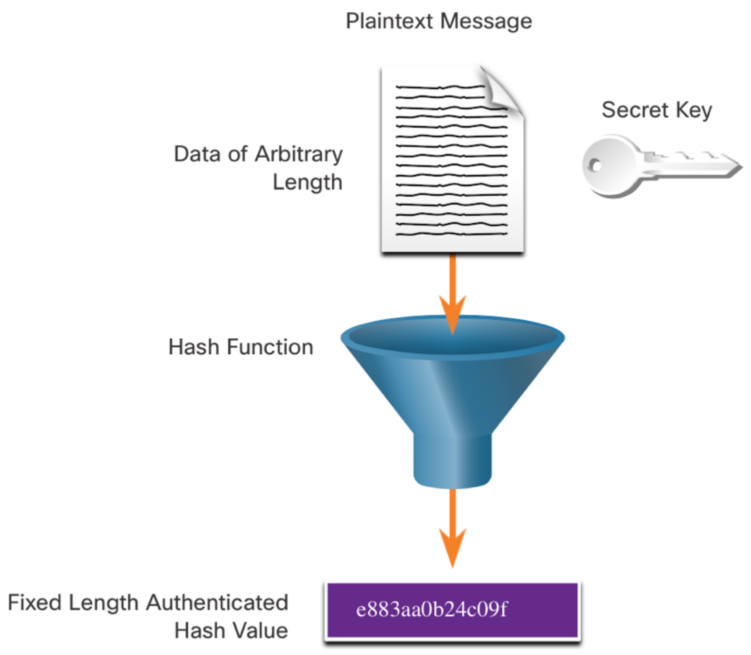
For more advanced Linux users another POC for collision: https://github.com/cr-marcstevens/hashclash#create-you-own-identical-prefix-collision

Hash-based Message Authentication Code - HMAC

HMAC, or Hash-based Message Authentication Code, is a type of message authentication code (MAC) designed to verify both the integrity and authenticity of a message. Unlike standard hashing, which is a one-way function, HMAC incorporates a secret key in the hashing process. This integration ensures the security of the message, as only those who possess the secret key can generate or verify the HMAC.

HMAC can be implemented using any standard hash function, such as MD5 or SHA-1, resulting in variants like HMAC-MD5 and HMAC-SHA-1. The security of HMAC relies on both the strength of the chosen hash function and the size of the hash output. Additionally, the length of the secret key plays a crucial role in the overall security of the HMAC.

The process of HMAC works as follows: the sender generates an HMAC code by hashing the message together with the secret key. This HMAC code is sent along with the message. Upon receiving the message, the receiver uses the same secret key to hash the message again and generate a new HMAC code. If the newly generated HMAC matches the received HMAC, it confirms the authenticity of the sender and the integrity of the message.



Video: Securing Stream Ciphers (HMAC) - Computerphile

MD5 HMAC

The command echo -n "Hello" | openssl dgst -md5 -hmac test is used to generate an HMAC (Hash-based Message Authentication Code) using the MD5 hashing algorithm and a specified key (secretkey).

```
echo -n "Hello" | openssl dgst -md5 -hmac secretkey
```

Command Breakdown:

- **echo -n "Hello":** This part of the command prints "Hello" to standard output without adding a newline at the end (due to the -n flag).
- **|:** The pipe operator takes the output of the command on its left and uses it as the input for the command on its right.
- **openssl dgst**: This invokes the OpenSSL toolkit's digest function.
- **-md5**: Specifies the MD5 hash function.
- **-hmac secretkey**: Specifies that HMAC should be used with "secretkey" as the secret key.

SHA256 HMAC

The command generates an HMAC using the SHA-256 hash function and a secret key for the input string "Hello"

```
echo -n "Hello" | openssl dgst -sha256 -hmac "secretkey"
```

Command Breakdown:

- **echo -n "Hello":** Prints the string "Hello" to standard output without appending a newline character.
- **|:** Pipes the output of the previous command as input to the next command.
- **openssl dgst**: Invokes the openssl command for generating message digests.
- **-sha256**: Specifies that the SHA-256 hash function should be used to generate the HMAC.
- **-hmac "secretkey":** Specifies the secret key used in the HMAC calculation.

Challenge:

Utilize the tools from the previous steps to analyse different MD5 outputs.

1. I issue following command:

   ```
   echo -n "some_string" | md5sum cyber
   ```

2. Record MD5
3. Issue command without –n

```
echo "some_string" | md5sum cyber
```

4. Record MD5
5. Compare hashes
6. Copy file **hash1** and name it **hash2**.

```
cp hash1 hash2
```

7. Compare sha256 hashes of **hash1** and **hash2** files
8. Open **hash2** file and put space at end and save file
9. Compare hash after editing file with previous hash
10. Reedit **hash2** file and remove added spade, save file
11. Generate sh256 and compare.
12. Is hash identical to hash1?

13. If so, why?

14. If not, why?

References

[i] Schneier, Bruce. "Cryptanalysis of MD5 and SHA: Time for a New Standard". Computerworld. Archived from the original on 2016-03-16. Retrieved 2016-04-20. Much more than encryption algorithms, one-way hash functions are the workhorses of modern cryptography.