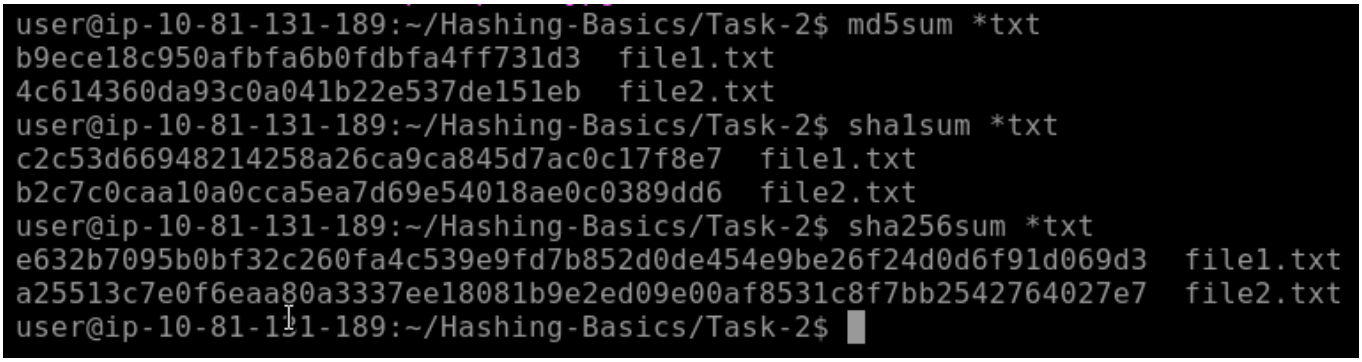
Hashing Basics

A **hash value** is a fixed-size string or characters that is computed by a hash function. A **hash function** takes an input of an arbitrary size and returns an output of fixed length, i.e., a hash value

Hash Functions

The output has a fixed size. It's hard to predict the output for any input and vice versa



different hashing algorithms provide different hashes for the the same words/files

The output of a hash function is typically raw bytes, which are then encoded. Common encodings are base64 or hexadecimal. `md5sum`, `sha1sum`, `sha256sum`, and `sha512sum` produce their outputs in hexadecimal format

Why is Hashing Important?

Hashing plays a vital role in our daily use of the Internet. Like other cryptographic functions, hashing remains hidden from the user. Hashing helps protect data's integrity and ensure password confidentiality.

What's a Hash Collision?

A hash collision is when two different inputs give the same output. Hash functions are designed to avoid collisions as best as possible.
a good hash function ensures that the probability of a collision is negligible.

Insecure Password Storage for Authentication

Stories of Insecure Password Storage for Authentication

Most web applications need to verify a user's password at some point. Storing these passwords in plaintext is a very insecure security practice

three insecure practices when it comes to passwords:

- Storing passwords in plaintext
- Storing passwords using a deprecated encryption
- Storing passwords using an insecure hashing algorithm

Storing Passwords in Plaintext

“rockyou.txt” password list on Kali Linux, among many other offensive security distributions. This password list came from RockYou, a company that developed social media applications and widgets. They stored their passwords in **plaintext**, and the company had a data breach

The text file contains over 14 million passwords

```
strategos@g5000 /usr/share/wordlists> wc -l rockyou.txt
14344392 rockyou.txt
strategos@g5000 /usr/share/wordlists> head rockyou.txt
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
abc123
```

Using an Insecure Encryption Algorithm

Adobe’s notable data breach was slightly different. Instead of using a secure hashing function to store the hash values of the passwords, the company used a deprecated encryption format. furthermore, password hints were stored in plain text, sometimes containing the password itself.

Using an Insecure Hash Function

LinkedIn also suffered a data breach in 2012. LinkedIn used an insecure hashing algorithm, the SHA-1, to store user passwords. Furthermore, no password salting was used. **Password salting** refers to adding a **salt**, i.e., a random value, to the password before it is hashed.

this displays the first 20 passwords in rockyou.txt

Using Hashing for Secure Password Storage

instead of storing the password, you just stored its hash value using a secure hashing function if your database is leaked, an attacker will have to crack each password to find out what the password was.

if two users have the same password? As a hash function will always turn the same input into the same output. if someone cracks that hash, they gain access to more than one account.

A **Rainbow Table** is a lookup table of hashes to plaintexts, so you can quickly find out what password a user had just from the hash

Hash	Password
02c75fb22c75b23dc963c7eb91a062cc	zxcvbnm
b0baee9d279d34fa1dfd71aadb908c3f	11111
c44a471bd78cc6c2fea32b9fe028d30a	asdfghjkl
d0199f51d2728db6011945145a1b607a	basketball
dcddb75469b4b4875094e14561e573d8	000000
e10adc3949ba59abbe56e057f20f883e	123456
e19d5cd5af0378da05f63f891c7467af	abcd1234

Hash	Password
e99a18c428cb38d5f260853678922e03	abc123
fcea920f7412b5da7be0cf42b8c93759	1234567
4c5923b6a6fac7b7355f53bfe2b8f8c1	inS3CyourP4$$

Websites like CrackStation and Hashes.com internally use massive rainbow tables to provide fast password cracking for **hashes without salts**

Protecting Against Rainbow Tables

we add a salt to the passwords. The salt is a randomly generated value stored in the database and should be unique to each user.

The salt is added to either the start or the end of the password before it's hashed, and this means that every user will have a different password hash even if they have the same password

Example of Securely Storing Passwords

following good security practices when storing user passwords:

1. We select a secure hashing function, such as Argon2, Scrypt, Bcrypt, or PBKDF2.
2. We add a unique salt to the password, such as `Y4UV*^(=go_!`
3. Concatenate the password with the unique salt. For example, if the password is `AL4RMc10k`, the result string would be `AL4RMc10kY4UV*^(=go_!`
4. Calculate the hash value of the combined password and salt. In this example, using the chosen algorithm, you need to calculate the hash value of `AL4RMc10kY4UV*^(=go_!`.
5. Store the hash value and the unique salt used (`Y4UV*^(=go_!`).

Using Encryption to Store Passwords

if we select a secure hashing algorithm to encrypt the passwords before storing them, we still need to store the used key. if someone gets the key, they can easily decrypt all the passwords.

Recognising Password Hashes

Automated hash recognition tools such as hashID exist but are unreliable for many formats. For hashes that have a prefix, the tools are reliable. Automated hash recognition tools often get these hash types mixed up, highlighting the importance of learning yourself.

Linux Passwords

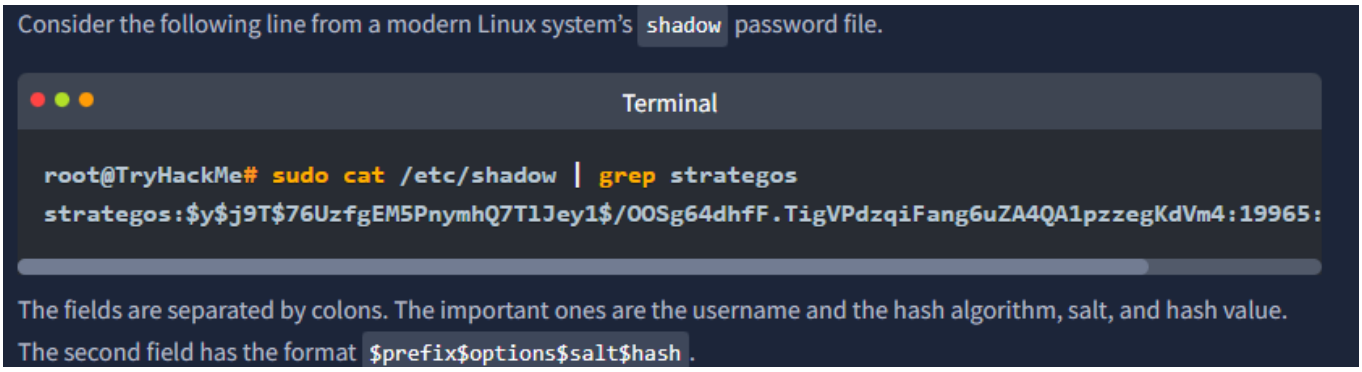
On Linux, password hashes are stored in `/etc/shadow`, which is normally only readable by root.

More information about the other fields can be found by executing `man 5 shadow`

table of some of the most common Unix-style password prefixes you might encounter. They are listed in the order of decreasing strength. man 5 crypt

Prefix	Algorithm
`$y$`	yescrypt is a scalable hashing scheme and is the default and recommended choice in new systems
`$gy$`	gost-yescrypt uses the GOST R 34.11-2012 hash function and the yescrypt hashing method
`$7$`	scrypt is a password-based key derivation function
`$2b$`, `$2y$`, `$2a$`, `$2x$`	bcrypt is a hash based on the Blowfish block cipher originally developed for OpenBSD but supported on a recent version of FreeBSD, NetBSD, Solaris 10 and newer, and several Linux distributions
`$6$`	sha512crypt is a hash based on SHA-2 with 512-bit output originally developed for GNU libc and commonly used on (older) Linux systems
`$md5`	SunMD5 is a hash based on the MD5 algorithm originally developed for Solaris
`$1$`	md5crypt is a hash based on the MD5 algorithm originally developed for FreeBSD

Modern Linux Example



In the example above, we have four parts separated by `$`:

- `y` indicates the hash algorithm used, **yescrypt**
- `j9T` is a parameter passed to the algorithm

- `76UzfgEM5PnymhQ7TlJey1` is the salt used
- `/OOSg64dhfF.TigVPdzqiFang6uZA4QA1pzzegKdVm4` is the hash value

MS Windows Passwords

MS Windows passwords are hashed using NTLM, a variant of MD4. They're visually identical to MD4 and MD5 hashes, so it's very important to use context to determine the hash type.

On MS Windows, password hashes are stored in the SAM (Security Accounts Manager)

place to find more hash formats and password prefixes is the Hashcat Example Hashes page

Password Cracking

Tools like Hashcat and John the Ripper are commonly used for these purposes if a salt is involved

Cracking Passwords with GPUs

they are very good at some mathematical calculations involved in hash functions. You can use a graphics card to crack many hash types quickly.

Cracking on VMs?

performance degradation occurs as you use the CPU from a virtualised OS, and when your purpose is to crack a hash, you need every extra CPU cycle.

Hashcat, it's best to run it on your host to make the most of your GPU, if available

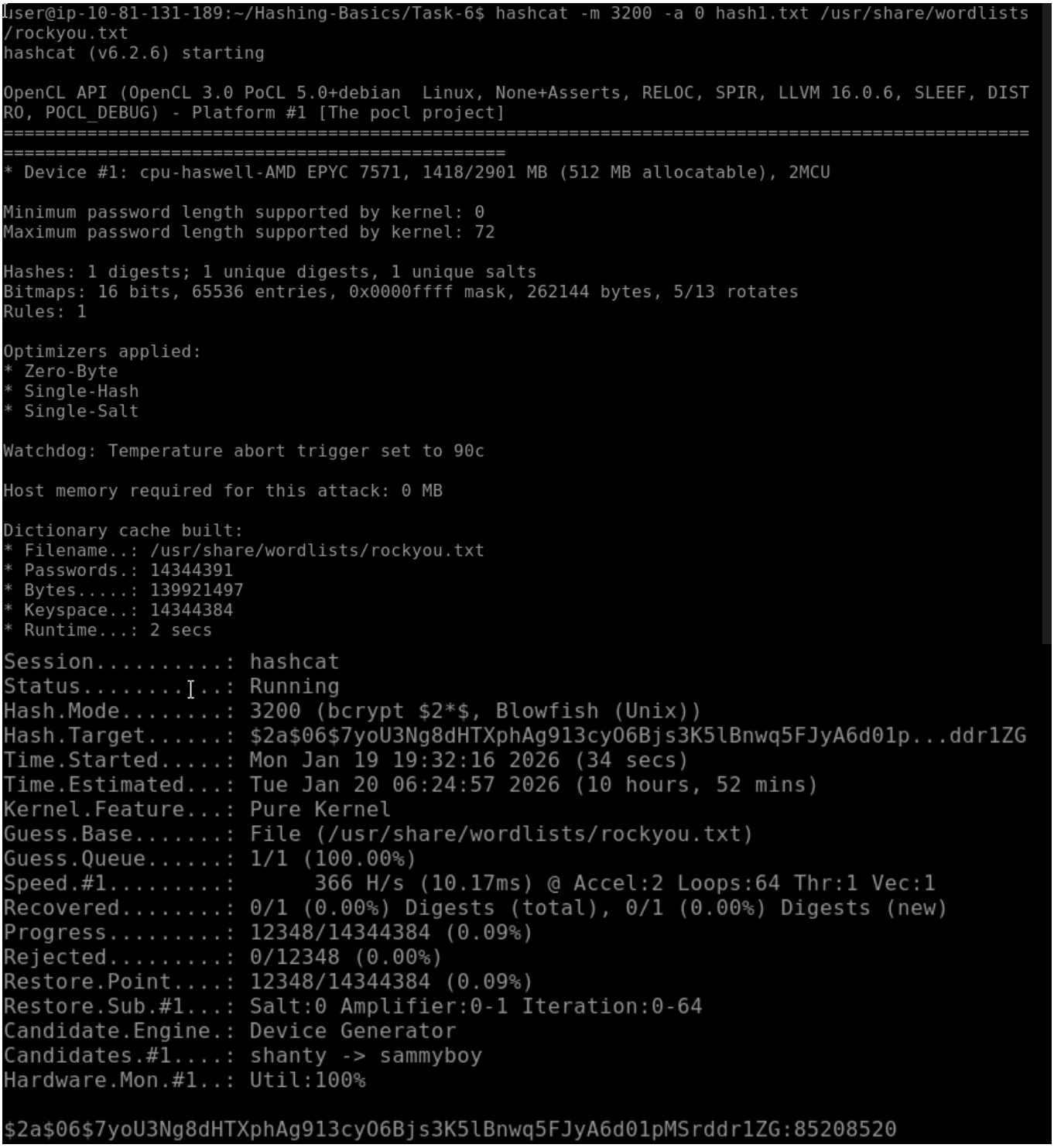
John the Ripper uses CPU by default and works in a VM out of the box, although you may get better speeds running it on the host OS

Time to Crack Some Hashes

Hashcat uses the following basic syntax: `hashcat -m <hash_type> -a <attack_mode> hashfile wordlist`, where:

- `-m <hash_type>` specifies the hash-type in numeric format. For example, `-m 1000` is for NTLM. Check the official documentation (`man hashcat`) and example page to find the hash type code to use.
- `-a <attack_mode>` specifies the attack-mode. For example, `-a 0` is for straight, i.e., trying one password from the wordlist after the other.
- `hashfile` is the file containing the hash you want to crack.
- `wordlist` is the security word list you want to use in your attack.

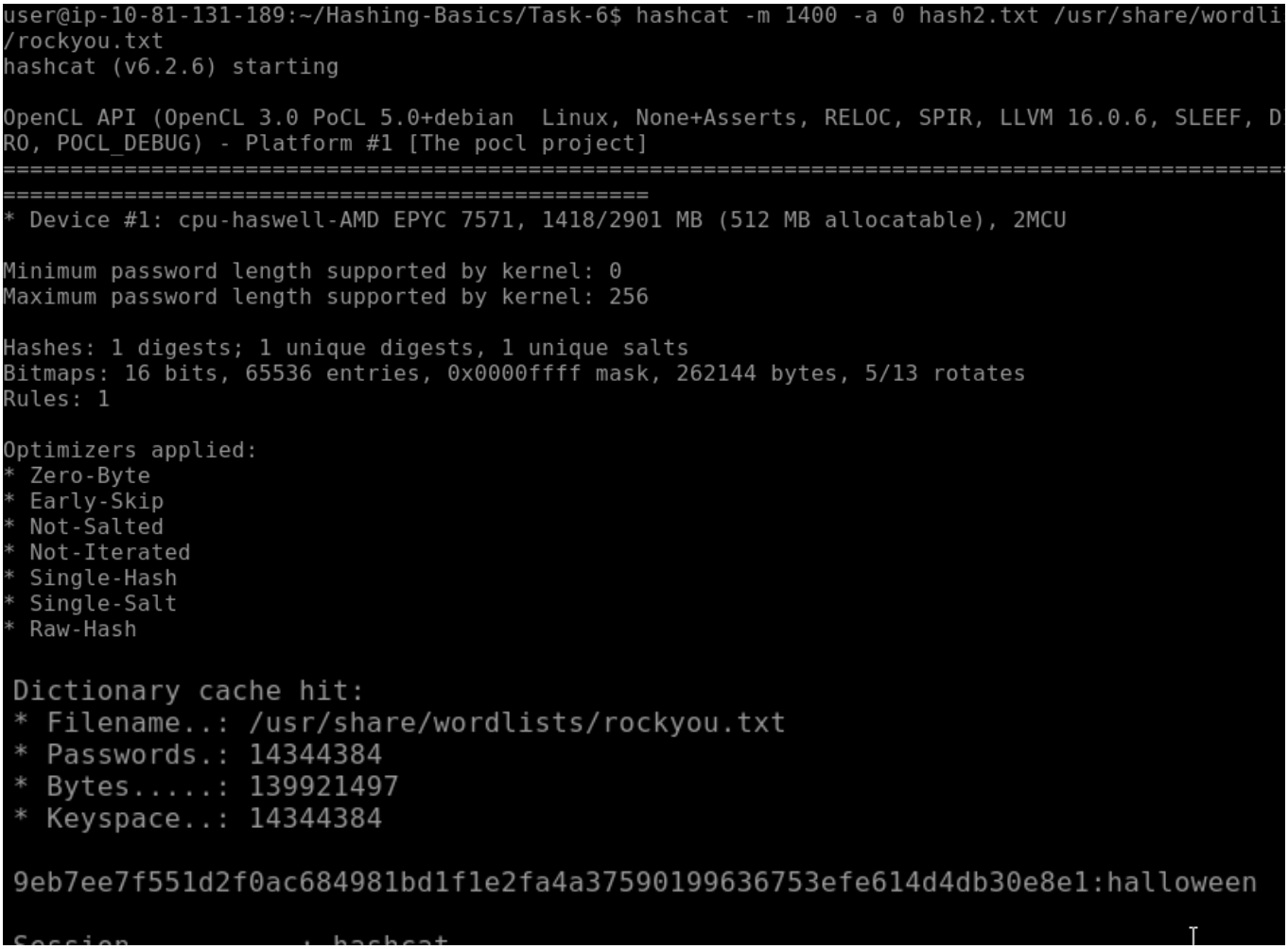
For example, `hashcat -m 3200 -a 0 hash.txt /usr/share/wordlists/rockyou.txt` will treat the hash as Bcrypt and try the passwords in the `rockyou.txt` file.



i ran the hashcat command with the rockyou.txt

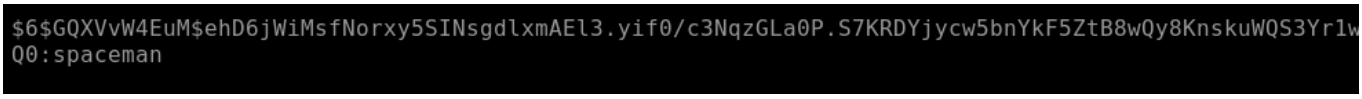
`hashcat -m 3200 -a 0 hash1.txt /usr/share/wordlists/rockyou.txt`

did the same with another text file but it was using the SHA2-256 hashing function



which gave me the password halloween

task 3 the same thing but was using sha512crypt



Hashing for Integrity Checking

Integrity Checking

Hashing can be used to check that files haven't been changed. If you put the same data in, you always get the same data out

the text file listed below shows the SHA256 hash of two Fedora Workstation ISO files. If running `sha256sum` on the file you downloaded returned the same hash listed in this signed file, you can be confident that your file is identical to the official one.

```
root@AttackBox# head Fedora-Workstation-40-1.14-x86_64-CHECKSUM
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

# Fedora-Workstation-Live-osb-40-1.14.x86_64.iso: 2623733760 bytes
SHA256 (Fedora-Workstation-Live-osb-40-1.14.x86_64.iso) = 8d3cb4d99f27eb932064915bc9ad34a7
# Fedora-Workstation-Live-x86_64-40-1.14.iso: 2295853056 bytes
SHA256 (Fedora-Workstation-Live-x86_64-40-1.14.iso) = dd1faca950d1a8c3d169adf2df4c3644ebb6
[...]
```

HMACs

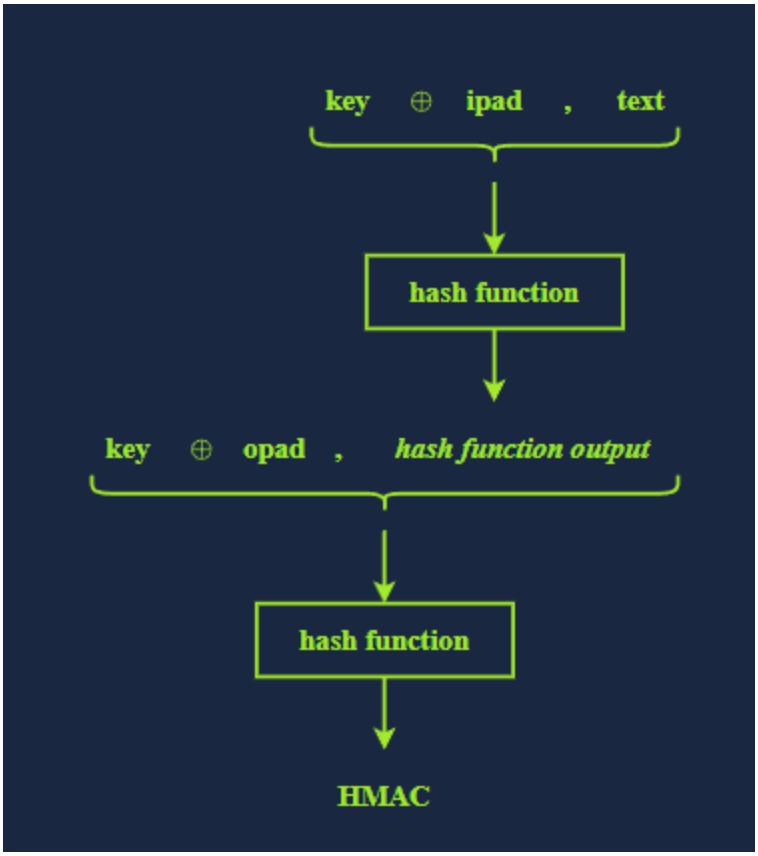
HMAC (Keyed-Hash Message Authentication Code) is a type of message authentication code (MAC) that uses a cryptographic hash function in combination with a secret key to verify the authenticity and integrity of data.

HMAC can be used to ensure that the person who created the HMAC is who they say they are

The following steps give you a fair idea of how HMAC works.

1. The secret key is padded to the block size of the hash function.
2. The padded key is XORed with a constant (usually a block of zeros or ones).
3. The message is hashed using the hash function with the XORed key.
4. The result from Step 3 is then hashed again with the same hash function but using the padded key XORed with another constant.

5. The final output is the HMAC value, typically a fixed-size string.



refresher on how to look at a hashsum of a file

```
user@ip-10-81-131-189:~/Hashing-Basics/Task-7$ sha256sum libgcrypt-1.11.0.tar.bz2
09120c9867ce7f2081d6aaa1775386b98c2f2f246135761aae47d81f58685b9c  libgcrypt-1.11.0.tar.bz2
```