Intro to Cybersecurity - Lab 3
Cryptography: Hashes, Stashes, and Rainbows

Aymane Sghier, Jun Wang, Zhipeng Yang

September 28, 2025

1 Introduction

This lab explores dictionary attacks and password cracking techniques using various cryptographic hash functions. Our team investigated methods to crack Unix passwords, MD5 hashes, SHA-1 hashes, and encrypted archive files using tools like John the Ripper, Hashcat, and online hash databases.

2 Task 4.1: Finding Unix passwords

We attempted to crack 5 different hash entries using John the Ripper:

- Hash type: descrypt
- Warnings about password length (DES limited to 8 chars).
- Progress indicators, cracked words appearing on screen.

3 Task 4.2: Finding a preimage to an MD5 hash

A small dictionary quickly cracked one hash (12345), showing unsalted MD5 is vulnerable to offline dictionary attacks. The other hashes did not appear in password.lst; larger lists (e.g., rockyou) or rule-based mutations would likely improve coverage. MD5 is a fast, obsolete hash; secure password storage should use a slow, salted KDF

- upyterLab on Linux; Python 3.
- hashlib for MD5; wordlist: password.lst
- System john lacked the Raw-MD5 format, so I used Python

4 Task 4.3: Finding SHA-1 Hash Preimages

4.1 Methodology

We attempted to crack 5 SHA-1 hashes using multiple approaches:

- Online hash database (CrackStation)
- Dictionary attack with rockyou.txt wordlist (14.3M passwords)
- Brute-force attacks with various character sets
- Hybrid attacks (dictionary + numeric suffixes)

4.2 Results

We successfully cracked 3 out of 5 hashes:

Hash 1: `41a6619fdbaebba7b498075d40277dbaaf060b1a`
Password: hunter2025
Method: CrackStation online database

Hash 2: `7b7a8d8e9435d1064967f8ba2a43eee1f7804f5e`
Password: Cybersecurity
Method: CrackStation online database

Hash 3: `1b5cbfcc0f2b490f358227f218a450db1b317f6e`
Password: hansolo
Method: CrackStation online database

4.3 Unsuccessful Attempts

Two hashes remained uncracked despite our attempts:

Hash 4: `5d9318d5969e82e93a6c20bbc44b273dee8594b4`
Attempts: CrackStation, rockyou.txt (14.3M passwords), brute-force (5-7 lowercase letters), hybrid attack (dictionary + 2-digit suffix)

Hash 5: `dc92e34e0b330083c87282980aec2478e11cda87`
Attempts: CrackStation, rockyou.txt (14.3M passwords), brute-force (5-7 lowercase letters), hybrid attack (dictionary + 2-digit suffix)

4.4 Analysis

Our 60% success rate shows that common passwords are still vulnerable to dictionary attacks and online rainbow tables. The two hashes we couldn't crack likely use stronger, more unique passwords that aren't in standard wordlists.

5 Task 4.4.1: Unlabeled Hashes

5.1 Hash Type Identification

We identified the hash types by analyzing their length and format:

Hash 1: `c8cfc421f2a881bbd835bfb6afeea5abd25c81eb`
Length: 40 hexadecimal characters
Identified as: RIPEMD160 (160 bits / 4 = 40 hex chars)

Hash 2: `0a4628c5ae0771b1d27ac71e2d2574dec4174f35fa0a6c5e7e0f98871fbcc657`
Length: 64 hexadecimal characters
Identified as: SHA-256 (256 bits / 4 = 64 hex chars)

Hash 3: bcrypt hash (begins with `\$2b\$`)
Format: bcrypt identifier prefix
Identified as: bcrypt (the `$2b$` prefix is the standard identifier)

Hash 4: `e61985475a6b6cab...486ae639` (128 hexadecimal characters total)
Length: 128 hexadecimal characters
Identified as: SHA-512 (512 bits / 4 = 128 hex chars)

5.2 Cracking Results

5.2.1 Successfully Cracked (2/4)

Hash 1 (RIPEMD160): `c8cfc421f2a881bbd835bfb6afeea5abd25c81eb`
Password: `escaped`
Method: CrackStation online database

Hash 2 (SHA-256): `0a4628c5ae0771b1d27ac71e2d2574dec4174f35fa0a6c5e7e0f98871fbcc657`
Password: `Boom!`
Method: CrackStation online database

5.2.2 Unsuccessful (2/4)

Hash 3 (bcrypt):
We ran a Hashcat dictionary attack using bcrypt mode 3200 with the rockyou.txt wordlist. After over 2 hours of computation, the hash remained uncracked. This is expected since bcrypt is specifically designed to be computationally expensive—it uses a slow key derivation function that requires significant processing time for each password attempt, making it resistant to both dictionary and brute-force attacks even with modern hardware.

Hash 4 (SHA-512):
We tried CrackStation, the full rockyou.txt dictionary, and multiple brute-force patterns (4-5 characters with various character sets). The password wasn't found in any of our attempts, suggesting it's either a strong unique password or uses an uncommon pattern not covered by standard wordlists.

5.3 Analysis

We correctly identified all four hash types and managed to crack half of them. The bcrypt result really demonstrates why it's considered a strong hashing algorithm—even with dedicated cracking tools, it takes hours just to test a wordlist because of its intentional computational cost. The SHA-512 hash appears to use a password that's genuinely strong and not in common dictionaries.

6 Task 4.4.2: Keyed and nested hashes

Use python brute force with two wordlist files to crack the hash.
hmac_md5_bruteforce_pairs.py

wordlist files:
author_names.txt female_names.txt

Running command: python3 hmac_md5_bruteforce_pairs.py 8b5a3e95656026f9ce2f405e279adf06 female_names.txt
author_names.txt

Running result:
Loaded 168 female keys and 55 author texts. total combos = 9240
FOUND!
Key: Adele
Text: Miller
HMAC: 8b5a3e95656026f9ce2f405e279adf06
Elapsed: 0.07s

Check by python:print(hmac.new(b"Adele", b"Miller",hashlib.md5).hexdigest())
it shows HMAC is 8b5a3e95656026f9ce2f405e279adf06 that matches the given hash

for crack a nested SHA2-256 hash, we need to decode SHA2-256(test) first , and then decode SHA2-256(SHA2-256(test)).

7 4.4.3 Something much harder: unusual hashes

Use python brute force with wordlist files to crack
code:
find_newline_hashes
wordlist: rockyou.txt
rockyou.txt is more than 50MB, nad it can not be uploaded

command: python3 find_newline_hashes.py rockyou.txt
it found 2 of 4

db9a2da27f2ed3e04823fd17160e7c94 radical
482b2d5c05d5b8cb9ae886cf8642fd5f quotient
I tried to use other different large wordlist to crack SH3-384,but there are no results.

8 4.5 Encrypted archive files

8.1 4.5.1 ZIP

I use ZIP Password Finder and Checker tool to crack the zip file

https://kwebpia.herokuapp.com/zippassword/

It found the password is screenshot,and the results for information are 3 images:1,2,3 which are uplaoded in zip file

8.2 4.5.2 7ZIP

I use VSPL 7z Password Recovery Software from Microsoft to crack
It takes a long time to crack by broute force attack and dictionary attack

9 Section 5: Word Problems

9.1 5.1 Attack Techniques Summary

The tools we used employ several different attack strategies. Dictionary attacks work by testing millions of common passwords from wordlists against the target hashes. Rule-based attacks take those dictionary words and automatically apply modifications like capitalizing letters, substituting numbers for letters (like "3" for "e"), and adding number patterns at the end. Brute-force attacks try every possible combination of characters within certain parameters, though this gets

exponentially more expensive as passwords get longer. Hybrid attacks combine dictionary words with systematic modifications. Online services like CrackStation use rainbow tables, which are basically huge precomputed databases that map hashes to passwords, allowing instant lookups for any hash that's already been computed.

9.2 5.2 Rainbow Table Viability

Rainbow tables work well in specific situations but have major limitations. They're great for quickly cracking unsalted hashes since everything's precomputed—you just look it up. The problem is they require massive amounts of storage (we're talking hundreds of gigabytes to terabytes), only work for specific hash algorithms, and completely fail against salted hashes. Since each different salt would need its own rainbow table, it becomes totally impractical for modern systems that use salting. So while rainbow tables are still useful for old systems that don't use salts, any properly implemented modern password system makes them basically useless.

9.3 5.3 JTR Improvements Over Simple Dictionary Attacks

John the Ripper improves on basic dictionary attacks in several ways. It has a built-in rule engine that automatically generates variations of each dictionary word—trying different capitalizations, character substitutions, adding suffixes, etc. It also has an "incremental mode" that uses statistical patterns to intelligently prioritize which password combinations to try first, rather than just randomly guessing. JTR can automatically detect what hash format you're working with, so you don't have to manually convert files. It can run across multiple CPU cores simultaneously and optionally use GPU acceleration for faster cracking. You can also pause and resume sessions, and it can chain together different attack modes to maximize your chances of success.

9.4 5.4 GPU Acceleration Impact

GPUs make a huge difference for password cracking because they have thousands of cores that can work in parallel, compared to CPUs that typically have less than 20 cores. Since hashing is easily parallelizable (each password attempt is independent), you can test way more passwords per second with a GPU. For fast hash algorithms like MD5, SHA-1, and SHA-256, GPU acceleration can speed things up 50-100 times or more compared to using just a CPU. However, this advantage mostly disappears with deliberately slow hash functions like bcrypt, scrypt, and Argon2. These algorithms are specifically designed to use a lot of computation and memory in ways that limit how much you can parallelize them, which is exactly why they're preferred for modern password storage.

9.5 5.5 Technique Improvement Strategies

There are several ways to improve password cracking effectiveness. Using targeted wordlists based on the specific target (like company names, local slang, demographic-specific passwords) works better than generic dictionaries. Incorporating leaked password databases from real breaches gives you actual passwords that people use in practice. Machine learning could analyze which passwords get cracked successfully and use that to better predict what to try next. Running the attack across multiple machines multiplies your computing power. For really specific applications, you could even build custom ASIC hardware optimized for particular hash algorithms, which would be faster than general-purpose GPUs. You can also analyze the target's password policy (minimum length, required character types) to skip testing passwords that wouldn't even be valid. But ultimately, the best defense is still implementing strong hash functions with proper salting and appropriate computational costs.

10 Conclusion

This lab gave us hands-on experience with real password cracking techniques and showed us where they succeed and where they fail. We successfully cracked 60% of the SHA-1 hashes and 50% of the unlabeled hashes using industry-standard tools. The hashes we couldn't crack demonstrated why strong passwords and properly implemented cryptographic systems (especially computationally expensive hash functions like bcrypt) actually work to protect user credentials in practice.

11 Appendix: Commands and Screenshots

All Hashcat commands executed and their outputs have been included as separate text files in the submission:

- hashcat_commands_4.3.txt - Commands for SHA-1 hash attacks
- hashcat_commands_4.4.1.txt - Commands for unlabeled hash attacks
- hashcat_output_sample.txt - Sample terminal output showing exhaustive search

Screenshots of CrackStation results are included in the screenshots folder.