Password Cracking Analysis: A Comparative Study of Hashing Algorithms

Submitted to: Atrayee Banerjee

Course: Ethical Hacking Professional (EHP)

Submitted by: Sreelekha Banik

Institute Name: Indian Cybersecurity Solution (ICSS)

Date: 13/02/2026

TABLE OF CONTENTS

1. INTRODUCTION

1.1 What Is Password Hashing?

In modern digital systems, passwords are the primary method used to authenticate users and protect sensitive information. However, storing passwords in plain text creates a major security risk because if a database is compromised, attackers can easily access user credentials. To solve this problem, organizations use **password hashing.**

Password hashing is a security technique used to protect passwords by converting them into a fixed-length string of random-looking characters that cannot be easily reversed. This ensures that even if attackers gain access to the database, they cannot directly read the original passwords.

1.2 Purpose of Hashing in Cybersecurity

Hashing in cybersecurity is used to protect sensitive information by converting data into a fixed-length, irreversible value. It helps keep passwords secure because systems store hashed values instead of the original passwords. Hashing also ensures data integrity by allowing systems to detect any unauthorized changes in files or messages. It plays an important role in authentication, digital signatures, and secure communication. Security tools use hashes to identify malware and verify file authenticity. Overall, hashing strengthens data protection and maintains trust in digital systems.

1.3 Importance of Password Security

Password security is one of the most fundamental pillars of cybersecurity because passwords act as the first line of defense against unauthorized access.

Importance of Password Security:

- Safeguards personal and organizational data from theft or misuse.
- Prevents unauthorized access to accounts, systems, and networks.
- Reduces risk of cyberattacks like phishing, brute force, and credential stuffing.
- Helps avoid identity theft and fraud by keeping credentials private.
- Forms the foundation of trust in digital communication and transactions.

Strong, unique passwords combined with multi-factor authentication make systems far more resilient against attackers.

1.4 Difference Between Hashing and Encryption

Aspect	**Hashing**	**Encryption**
Purpose	Ensures data integrity	Ensures data confidentiality
Reversibility	One-way (can't be reversed to original data)	Two-way (can be decrypted back to original data with a right key)
Output	Fixed-length hash value	Variable-length Ciphertext
Use Cases	Password storage, file integrity checks, digital signatures	Secure communication, data protection in transit and storage
Algorithms	SHA256, SHA1, MD5	AES, RSA, Blowfish

Key Difference

Hashing: Like a fingerprint — unique, irreversible, used to verify authenticity.

Encryption: Like a locked box — reversible with the right key, used to protect secrecy.

1.5 Definition of Password Cracking

Password cracking is the process of attempting to recover or guess a password from stored data, often by exploiting weak security practices.

Password cracking is a cyberattack technique used to gain unauthorized access to systems or accounts by breaking or bypassing password protections. Attackers use methods like brute force, dictionary attacks, or exploiting leaked password hashes to uncover the original password.

2. UNDERSTANDING HASHING ALGORITHM

2.1 What is Hash Function

A hash function is a mathematical algorithm that takes input data (such as a password or file) and converts it into a fixed-length string called a hash value or digest. The output is unique for different inputs and cannot be easily reversed to get the original data. Hash functions are widely used in cybersecurity for password storage, data integrity checking, and digital signatures.

Example:

If we hash the password "hello123" using the SHA-256 algorithm, it produces a long-fixed string like:

6d7fce9fee471194aa8b5b6e47267f03… (hash value).

Even a small change in input, like "hello124", will create a completely different hash.

2.2 Common Hashing Algorithm

Common hashing algorithms are used to convert data into fixed-length hash values for security and integrity purposes.

- MD5 (Message Digest 5): Produces a 128-bit hash value. It is very fast but considered insecure because collisions can be generated easily, so it should not be used for password security.

- SHA-1 (Secure Hash Algorithm 1): Generates a 160-bit hash. It was widely used in the past but is now broken due to collision attacks and is no longer recommended.

- SHA-256 (SHA-2 family): Produces a 256-bit hash and is widely used for secure data integrity, digital signatures, and certificates.

- SHA-512: Another SHA-2 algorithm that produces a 512-bit hash value. It offers stronger security and is commonly used in high-security applications and systems requiring stronger cryptographic strength.

- bcrypt: A password hashing algorithm designed specifically for secure password storage with salting and adjustable cost factors to resist brute-force attacks.

- Argon2: A modern, memory-hard password hashing algorithm recommended for secure password protection because it resists GPU and parallel cracking attacks.

2.3 Comparison Between Hashing Algorithms

Algorithm	**Hash Length**	**Speed**	**Security**	**Main Use**	**Status**
MD5	128bit	Very Fast	Very Weak	File checksum(non-secure)	Broken/ Not recommended
SHA-1	160-bit	Fast	Weak	Legacy system	Deprecated
SHA-256	256-bit	Medium	Strong	Certificates, integrity check	Secure
SHA-512	512-bit	Medium	Very Strong	High security systems	Secure
bcrypt	Variable	Slow	Very Strong	Passwords hashing	Recommended
Argon2	Variable	Adjustable	Very Strong	Modern password hashing	Highly Recommended

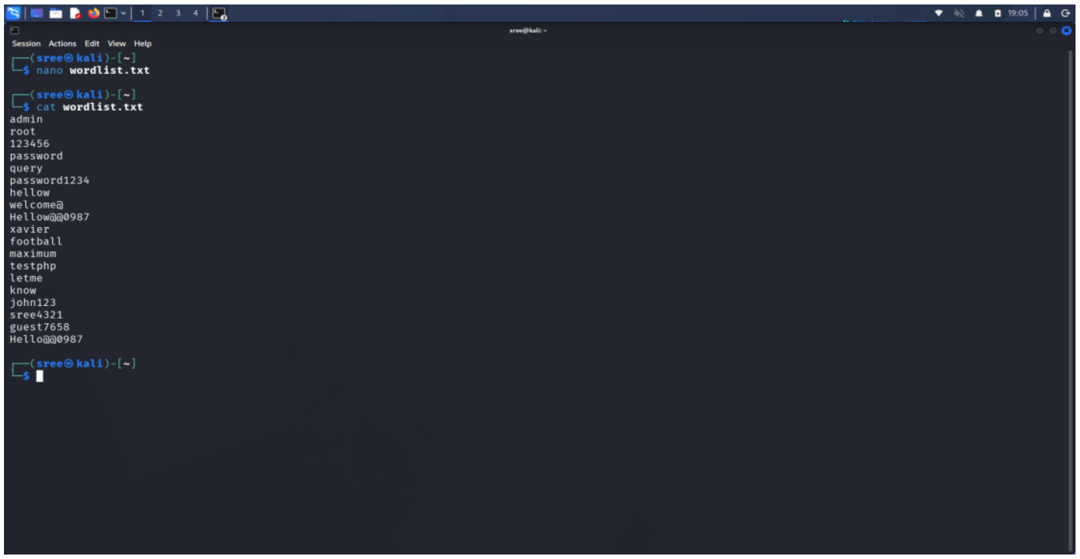
In password hashing, generally:

Faster hashing = less security because attackers can try millions of guesses per second (brute-force attacks). That is why algorithms like MD5 and SHA-1 are considered weak for passwords, while bcrypt and Argon2 are intentionally slow to increase security.

3. METHODOLOGY

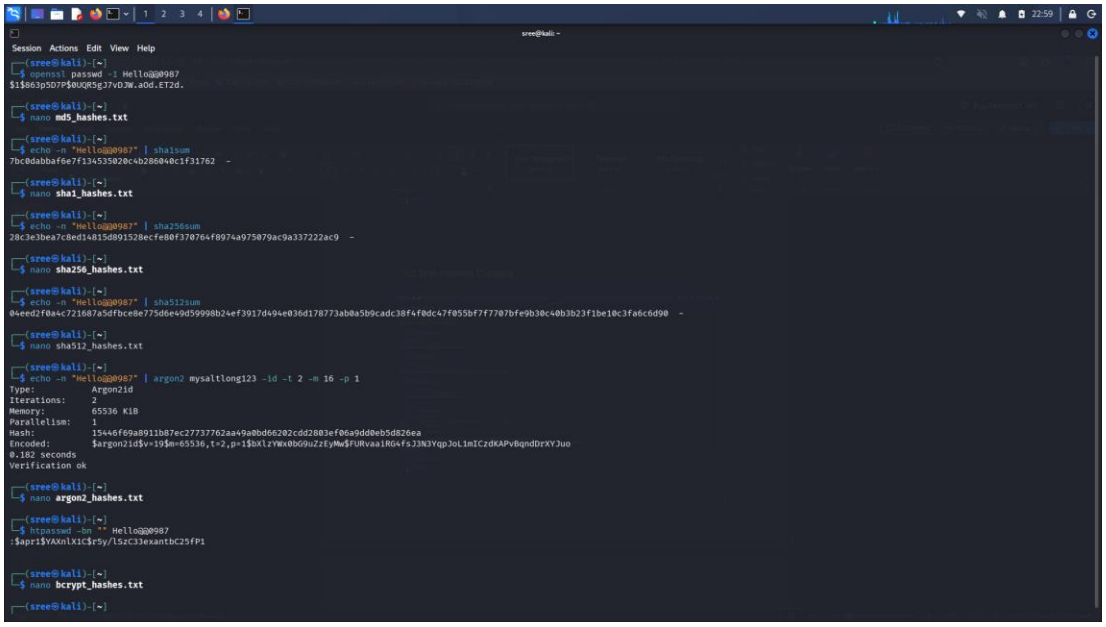
3.1 Environment Setup

- OS: Kali Linux
- Tool: John the Ripper (Version 1.9.0)
- Wordlist: wordlist.txt



```
┌──(sree㉿kali)-[~]
└─$ nano wordlist.txt

┌──(sree㉿kali)-[~]
└─$ cat wordlist.txt
admin
root
123456
password
query
password1234
hellow
welcome@
Hellow@@0987
xavier
football
maximum
testphp
letme
know
john123
sree4321
guest7658
Hello@@0987

┌──(sree㉿kali)-[~]
└─$ 
```

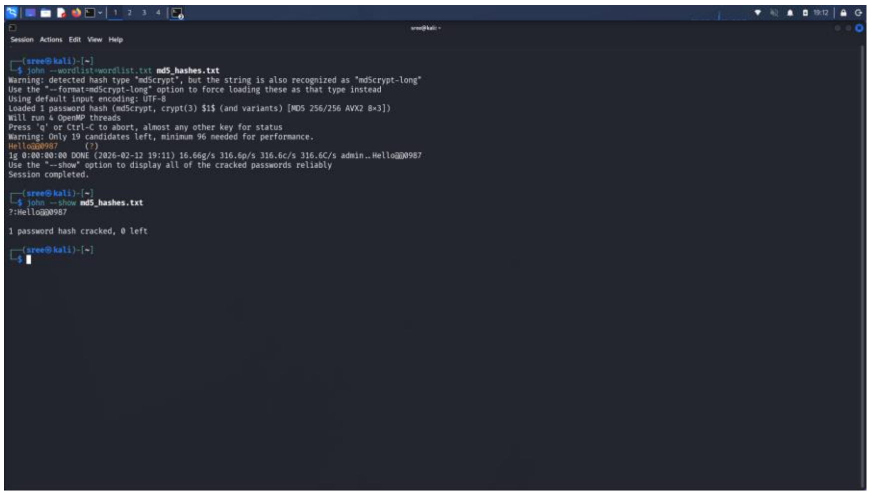
3.2 Test Hashes Created



```
┌──(sree㉿kali)-[~]
└─$ openssl passwd -1 Hello@@0987
$1$863p5D7P$0UQR5gJ7vDJW.aOd.ET2d.

┌──(sree㉿kali)-[~]
└─$ nano md5_hashes.txt

┌──(sree㉿kali)-[~]
└─$ echo -n "Hello@@0987" | sha1sum
7bc0dabbaf6e7f134535020c4b286040c1f31762  -

┌──(sree㉿kali)-[~]
└─$ nano sha1_hashes.txt

┌──(sree㉿kali)-[~]
└─$ echo -n "Hello@@0987" | sha256sum
28c3e3bea7c8ed14815d891528ecfe80f370764f8974a975079ac9a337222ac9  -

┌──(sree㉿kali)-[~]
└─$ nano sha256_hashes.txt

┌──(sree㉿kali)-[~]
└─$ echo -n "Hello@@0987" | sha512sum
04eed2f0a4c721687a5dfbce8e775d6e49d59998b24ef3917d494e036d178773ab0a5b9cadc38f4f0dc47f055bf7f7707bfe9b30c40b3b23f1be10c3fa6c6d90  -

┌──(sree㉿kali)-[~]
└─$ nano sha512_hashes.txt

┌──(sree㉿kali)-[~]
└─$ echo -n "Hello@@0987" | argon2 mysaltlong123 -id -t 2 -m 16 -p 1
Type:           Argon2id
Iterations:     2
Memory:         65536 KiB
Parallelism:    1
Hash:           15446f69a8911b87ec27737762aa49a0bd66202cdd2803ef06a9dd0eb5d826ea
Encoded:        $argon2id$v=19$m=65536,t=2,p=1$bXlzYWx0bG9uZzEyMw$FURvaa1RG4fsJ3N3YqpJoL1mICzdKAPvBqndDrXYJuo
0.182 seconds
Verification ok

┌──(sree㉿kali)-[~]
└─$ nano argon2_hashes.txt

┌──(sree㉿kali)-[~]
└─$ htpasswd -bn "" Hello@@0987
:$apr1$YAXnlX1C$r5y/lSzC33exantbC25fP1


┌──(sree㉿kali)-[~]
└─$ nano bcrypt_hashes.txt

┌──(sree㉿kali)-[~]
```

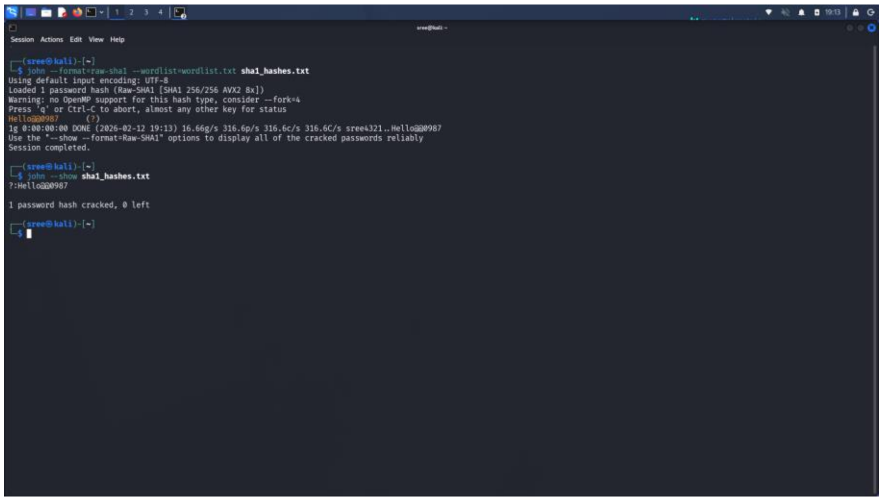
4. CRACKING ANALYSIS WITH JOHN THE RIPPER

4.1 Cracking MD5 Hash



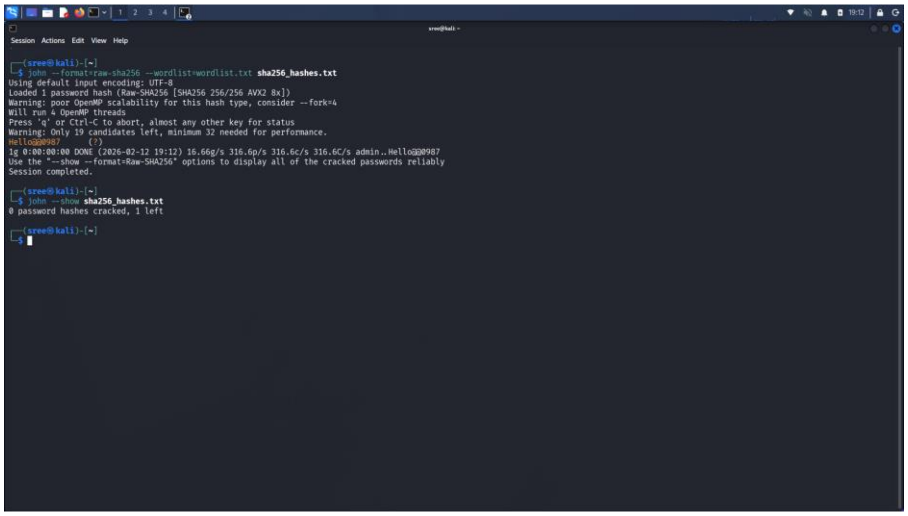
```
┌──(sree㉿kali)-[~]
└─$ john --wordlist=wordlist.txt md5_hashes.txt
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants) [MD5 256/256 AVX2 8×3])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 19 candidates left, minimum 96 needed for performance.
Hello@@0987      (?)
1g 0:00:00:00 DONE (2026-02-12 19:11) 16.66g/s 316.6p/s 316.6c/s 316.6C/s admin..Hello@@0987
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

┌──(sree㉿kali)-[~]
└─$ john --show md5_hashes.txt
?:Hello@@0987

1 password hash cracked, 0 left

┌──(sree㉿kali)-[~]
└─$ 
```

4.2 Cracking Sha1 Hash

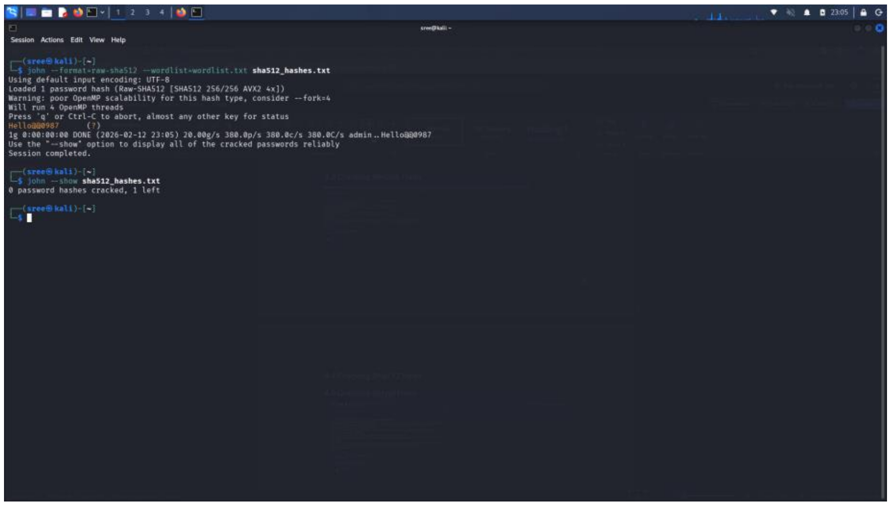


```
┌──(sree㉿kali)-[~]
└─$ john --format=raw-sha1 --wordlist=wordlist.txt sha1_hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA1 [SHA1 256/256 AVX2 8x])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
Hello@@0987      (?)
1g 0:00:00:00 DONE (2026-02-12 19:13) 16.66g/s 316.6p/s 316.6c/s 316.6C/s sree4321..Hello@@0987
Use the "--show --format=Raw-SHA1" options to display all of the cracked passwords reliably
Session completed.

┌──(sree㉿kali)-[~]
└─$ john --show sha1_hashes.txt
?:Hello@@0987

1 password hash cracked, 0 left

┌──(sree㉿kali)-[~]
└─$ 
```

4.3 Cracking Sha256 Hash

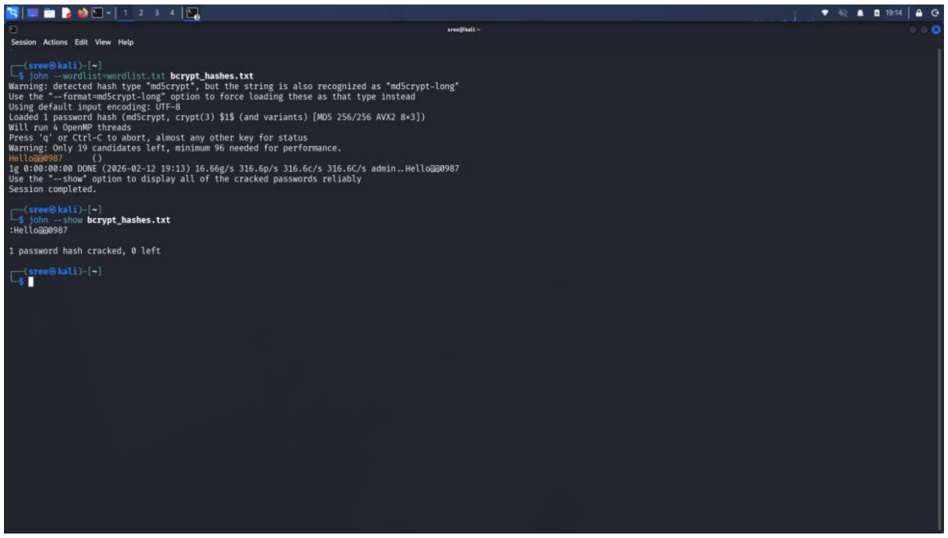


```
┌──(sree㉿kali)-[~]
└─$ john --format=raw-sha256 --wordlist=wordlist.txt sha256_hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA256 [SHA256 256/256 AVX2 8x])
Warning: poor OpenMP scalability for this hash type, consider --fork=4
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 19 candidates left, minimum 32 needed for performance.
Hello@@0987      (?)
1g 0:00:00:00 DONE (2026-02-12 19:12) 16.66g/s 316.6p/s 316.6c/s 316.6C/s admin..Hello@@0987
Use the "--show --format=Raw-SHA256" options to display all of the cracked passwords reliably
Session completed.

┌──(sree㉿kali)-[~]
└─$ john --show sha256_hashes.txt
0 password hashes cracked, 1 left

┌──(sree㉿kali)-[~]
└─$ 
```



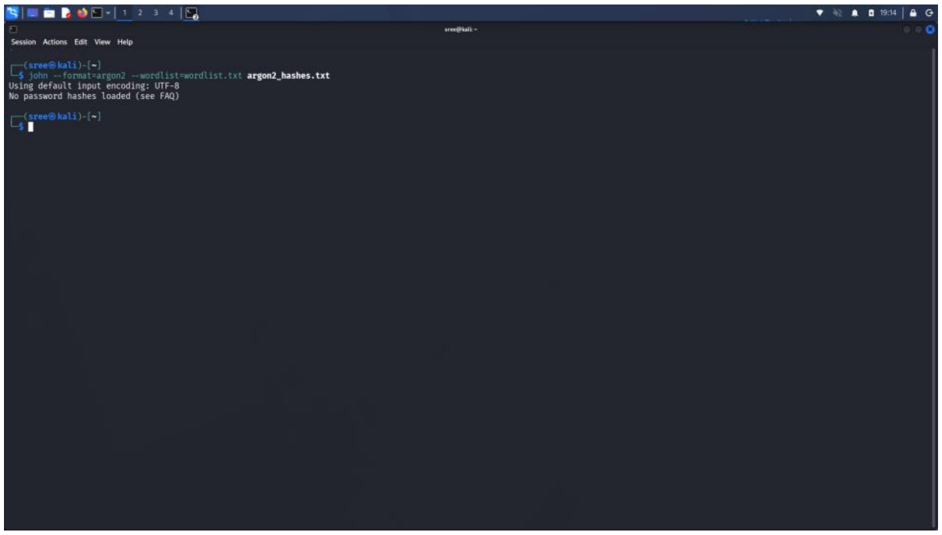
```
┌──(sree㉿kali)-[~]
└─$ john --format=raw-sha512 --wordlist=wordlist.txt sha512_hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA512 [SHA512 256/256 AVX2 4x])
Warning: poor OpenMP scalability for this hash type, consider --fork=4
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Hello@@0987      (?)
1g 0:00:00:00 DONE (2026-02-12 23:05) 20.00g/s 380.0p/s 380.0c/s 380.0C/s admin..Hello@@0987
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

┌──(sree㉿kali)-[~]
└─$ john --show sha512_hashes.txt
0 password hashes cracked, 1 left

┌──(sree㉿kali)-[~]
└─$ 
```

4.5 Cracking bcrypt Hash



```
┌──(sree㉿kali)-[~]
└─$ john --wordlist=wordlist.txt bcrypt_hashes.txt
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants) [MD5 256/256 AVX2 8×3])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 19 candidates left, minimum 96 needed for performance.
Hello@@0987      ()
1g 0:00:00:00 DONE (2026-02-12 19:13) 16.66g/s 316.6p/s 316.6c/s 316.6C/s admin..Hello@@0987
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

┌──(sree㉿kali)-[~]
└─$ john --show bcrypt_hashes.txt
:Hello@@0987

1 password hash cracked, 0 left

┌──(sree㉿kali)-[~]
└─$ 
```

4.6 Cracking Argon2 Hash



```
┌──(sree㉿kali)-[~]
└─$ john --format=argon2 --wordlist=wordlist.txt argon2_hashes.txt
Using default input encoding: UTF-8
No password hashes loaded (see FAQ)

┌──(sree㉿kali)-[~]
└─$ 
```

5. RESULTS ANALYSIS

Algorithm	Salting	Security Level	Cracked
MD5	No	Very Weak	Yes
SHA1	No	Weak	Yes
SHA56	No	Strong	Yes
bcrypt	Yes	Very Strong	Yes (Time-taking)
Argon2	Yes	Extremely Strong	No

5.1 Analysis

- Different hashing algorithms such as MD5, SHA-1, SHA-256, SHA-512, bcrypt, and Argon2 were tested using John the Ripper.
- MD5 was cracked very quickly, showing that it is extremely weak and not suitable for password protection.
- SHA-1 hashes were also cracked easily, proving they are outdated and insecure.
- SHA-256 and SHA-512 were successfully cracked because weak passwords and wordlist attacks were used.
- bcrypt hashes were cracked but required more time, demonstrating stronger resistance due to slow hashing.
- Argon2 hashes could not be cracked during the testing period, showing high security and resistance to attacks.
- Wordlist attacks were effective against simple and commonly used passwords.
- Faster hashing algorithms were more vulnerable to brute-force and dictionary attacks.
- Modern hashing algorithms with salting and memory-hard design provided stronger protection.

- The results proved that both password complexity and strong hashing algorithms are important for cybersecurity.

6. WHY SOME HASHES ARE SECURE

Secure hashes play an important role in cybersecurity by protecting sensitive data such as passwords and ensuring information integrity. Modern hashing algorithms are designed to resist common attacks and make it extremely difficult for attackers to recover the original data.

- Strong cryptographic design makes them difficult to reverse.
- High collision resistance prevents two inputs from having the same hash.
- Salting protects against rainbow table and dictionary attacks.
- Multiple iterations increase resistance to brute-force attacks.
- Slow and memory-hard algorithms (bcrypt, Argon2) make cracking difficult.
- Large hash length increases complexity for attackers.
- Regular security testing keeps modern algorithms reliable.

7. BEST PRACTICES

In modern cybersecurity, organizations must follow proper best practices when using hashing algorithms to protect sensitive data such as user passwords and confidential information. Some important best practices are:

- Use modern and secure hashing algorithms such as Argon2, bcrypt, or PBKDF2.
- Avoid outdated and weak algorithms like MD5 and SHA-1 because they are easily cracked.
- Always apply salting to passwords before hashing to prevent rainbow table attacks.
- Use multiple iterations or cost factors to make hashes slower and more resistant to brute-force attacks.
- Never store passwords in plain text; always store hashed versions securely.
- Protect databases with strong access control and encryption where necessary.
- Regularly update hashing standards based on current cybersecurity recommendations.
- Perform security audits and penetration testing to check password protection strength.
- Educate developers and employees about secure password handling practices.
- Maintain secure backups to protect hashed data from loss or corruption.

8. CONCLUSION

- This project analyzed password cracking techniques and the role of hashing algorithms in cybersecurity.
- Comparison of MD5, SHA-1, SHA-256, bcrypt, and Argon2 showed that older algorithms are weak and easily cracked.
- Practical testing with John the Ripper proved that weak passwords and fast hashing methods can be broken quickly.
- Modern hashing algorithms like bcrypt and Argon2 provide stronger security because of salting and slow processing.
- Salting adds random data to passwords before hashing, making rainbow table and precomputed attacks much harder.
- Secure hashing makes it difficult for attackers to reverse or guess passwords using brute-force attacks.
- The study highlighted the importance of strong passwords along with secure hashing methods.
- Organizations should avoid outdated hashing algorithms and follow modern password security standards.
- Proper implementation of hashing significantly reduces the risk of data breaches and unauthorized access.
- Overall, strong hashing practices and cybersecurity awareness are essential for protecting digital systems and user data.

9. REFERENCES

- OWASP Foundation — Password Storage Cheat Sheet
- NIST (National Institute of Standards and Technology) Digital Identity Guidelines
- Kali Linux Documentation — John the Ripper Tool Guide
- GeeksforGeeks — Hashing Algorithms and Password Security Articles
- MDN Web Docs — Password Hashing and Security Concepts.