Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 2, 2307-2317
2025
Publisher: Learning Gate
DOI: 10.55214/25768484.v9i2.5072

Vulnerabilities in cryptographic hash functions: A practical study of hash cracking techniques and security implications

Jasmin Redžepagić[1*], Damir Regvart[2], Hrvoje Rudeš[3], Robert Petrunić[4]
[1,2,3,4]Department of System Engineering and Cybersecurity Algebra University, Zagreb, Croatia; jasmin.redzepagic@algebra.hr (J.R.) damir.regvart@algebra.hr (D.R.) hrudes@algebra.hr (H.R.) robert.petrunic@algebra.hr (R.P.).

Abstract: This paper explores hash functions and their security challenges, which ensure data integrity and confidentiality in digital systems. With advancing computing power, particularly GPUs and distributed networks, hash functions face increasing threats from brute-force, dictionary, and rainbow table attacks. Tools like Hashcat, Cain & Abel, and John the Ripper were analyzed for their efficiency against MD5, SHA-1, and SHA-512. Results highlight vulnerabilities in standard algorithms, emphasizing the need for more robust hash functions, especially in high-security and resource-constrained environments.

Keywords: *Brute-force attacks, Cybersecurity, Data integrity, Dictionary attacks, Hashes, Rainbow tables.*

1. Introduction

Hash functions are an essential component of cryptographic systems, providing critical functions in data integrity, authentication, and error detection. With the increasing prevalence of digital data and online services, the security of hash functions has become paramount. Yet, hash algorithms are vulnerable to attacks that seek to reveal the plaintext behind a hash or exploit weaknesses in the hashing process. This paper examines these vulnerabilities in-depth, exploring the characteristics, weaknesses, and strengths of different types of hash functions through practical experiments. The primary tools analyzed include Hashcat, Cain and Abel, and John the Ripper, which employ various methods such as brute-force attacks, dictionary attacks, and rainbow tables to crack hash values.

In conducting these experiments, we used a laptop with limited resources to provide a baseline understanding of each method's computational demands. While the hardware constraints may not reflect high-performance environments, they illustrate the practical limitations and potential improvements achievable with more advanced configurations.

This paper offers several contributions to the study of hash function security. First, it analyzes popular hash-cracking tools and their capabilities, focusing on speed, compatibility, and processing requirements. Second, it compares different hash-value attack techniques, including brute force, dictionary, and rainbow table attacks, to assess their effectiveness against various hash functions. Third, it examines hardware efficiency, illustrating the impact of CPU and GPU usage on cracking times and demonstrating the advantages of GPU-optimized algorithms.

The rest of this paper is organized as follows. Section II discusses related work to hash functions, followed by section III, which discusses the basics. Section IV details the environment setup and data collection process, followed by various hash-cracking tools and their configurations. Section VI covers specific attack techniques, while section VII analyzes experimental results, discussing their implications for practical security applications. Finally, section VIII suggests areas for future research, and section IX concludes the paper.

History: Received: 20 December 2024; Revised: 24 January 2025; Accepted: 6 February 2025; Published: 27 February 2025
* Correspondence: jasmin.redzepagic@algebra.hr

2. Related Work

Hash functions, as a cornerstone of cryptographic security, have evolved significantly in response to emerging vulnerabilities and attacks. Various research studies over recent years have focused on improving the robustness of hash algorithms, particularly in applications like digital forensics, password protection, and data integrity. For instance, Jangid and Vidushi [1] S-HASH tool emphasizes the ongoing threat of dictionary and brute-force attacks on widely used algorithms such as MD5 and SHA, highlighting their susceptibility to cracking efforts. Similarly, Kundu and Dutta [2] discuss the vulnerability of traditional cryptographic hash functions to a spectrum of attacks, including those targeting collision and preimage resistance.

The advancements in hash function security are underscored by modifications to widely used algorithms, as seen in Bemida, et al. [3] enhanced SHA-512, which introduces changes to increase resistance against brute-force and rainbow table attacks. In another effort, Tihanyi, et al. [4] developed a privacy-preserving cracking protocol to allow third-party password recovery without compromising data confidentiality, employing methods such as predicate encryption. Furthermore, Marchetti and Bodily [5] illustrate the effectiveness of the John the Ripper tool in dictionary attacks, noting the tool's limitations in brute-force scenarios, particularly against well-constructed password hashes.

Research on lightweight cryptographic hash functions tailored for resource-constrained environments, such as IoT, addresses the need to balance performance and security. Windarta, et al. [6] analyze various lightweight hash functions that cater to the specific requirements of IoT devices, which typically cannot accommodate resource-intensive security protocols. In line with these constraints, El Hanouti, et al. [7] present a streamlined hash function designed for speed without severely compromising security, making it suitable for applications needing rapid processing and moderate security.

Enhancements in algorithm architecture also focus on optimizing hash functions for specialized hardware, as demonstrated by Sideris, et al. [8] who propose a modified Keccak-based SHA-3 architecture for FPGA devices, significantly boosting throughput. Similarly, Rudy and Rodwald [9] demonstrate that distributed cracking platforms like Hashtopolis can leverage multiple GPUs, optimizing performance for forensic and security applications requiring high-speed hash computations.

The design and classification of hash functions based on anti-attack resilience reflect ongoing efforts to develop more secure hash mechanisms. Wang and Gu [10] introduce a classification methodology based on anti-attack performance, providing a framework for choosing appropriate hash functions in various security scenarios. The classification aims to address weaknesses in traditional algorithms, a theme echoed by Upadhyay, et al. [11] who emphasize the importance of the avalanche effect in hash security, identifying vulnerabilities in widely used algorithms through rigorous statistical testing.

In password protection, salt hash mechanisms remain a popular countermeasure against dictionary attacks. Nugroho and Mantoro [12] application of salt to MD5 hashing demonstrates increased resilience to dictionary-based cracking attempts. Meanwhile, Nair and Song [13] present a multi-factor credential hashing function, which integrates multi-factor authentication for asymmetric brute-force resistance, a critical feature in securing high-value systems against unauthorized access.

Multiple papers Kpieleh [14] and Anwar, et al. [15] researched the adaptation of cryptographic hashes for new applications, such as digital stamping and certificate verification. These papers underscore the role of hash functions in verifying digital signatures and authenticating certificate data. They advocate for improvements in the cryptographic integrity of data, especially in applications where data authenticity is crucial.

Soni, et al. [16] investigation into novel hash mechanisms like NeuroHash exemplifies the quest for innovative solutions that resist known vulnerabilities in classical hash functions. This neuro-inspired approach, which utilizes XOR operations and neural network models, provides an alternative hashing method with potential resilience to attacks commonly affecting traditional hash functions.

3. Hash Functions

To access nearly any online service or company resource, users must have an account with a username and password. Based on the permissions granted to each user, they have specific rights to

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 2: 2307-2317, 2025
DOI: 10.55214/25768484.v9i2.5072

resources and applications. To enable the system to differentiate between users, each individual is identified by their username, which must generally be unique, and their password. Access to the system is only granted when both pieces of information match the data stored within the user database and are correctly entered in the login form.

Many systems have predefined formats for usernames and passwords. For example, usernames may not allow spaces or special symbols, while passwords must contain specific character combinations to ensure security.

3.1. Defining Hash Functions

Before explaining how hash functions ensure the integrity of a message or document and provide additional protection when storing sensitive data such as passwords, it is essential to define what a hash, or digital digest, is and why some authors refer to it as a "cryptographic Swiss army knife" [17]. It is also noted that nearly all cryptographic functions and protocols use some form of hash, whether for preserving integrity or as a security mechanism [17]. Generating a digest effectively produces a fixed-size digital signature for the input data [18].

A hash algorithm must be collision-resistant, meaning it should not produce the same output for two inputs. If the algorithm is robust, each unique input sequence creates a distinct digest, ensuring that the digital fingerprint is unique and cannot be replicated with a different input value [18]. However, creating an ideal hash function is nearly impossible since the hash output is of fixed length, typically 2^N, where N is a natural number, resulting in outputs of 16, 32, 64, or more bytes. Conversely, input values are unrestricted in size, inevitably leading to the possibility of two different inputs producing the same output.

The integrity of information or a system is one of the three foundational principles of a secure information system, known as the CIA triad: confidentiality, availability, and integrity. Data integrity means that information remains unchanged during communication, preventing unauthorized individuals from altering the message and assuring the recipient that the message received is the original sent by the sender. Cryptographic hash functions are responsible for this task and will be analyzed further in the following sections. Additionally, hash functions verify the immutability of messages amidst various adverse events, from cyberattacks to natural disasters. In an ideal hash function, if a digest is calculated for a given input value, and then a single bit of that input is altered, the resulting output should be entirely different. Furthermore, an ideal hash function is one-way, meaning the initial input value cannot be determined from the output.

3.2. Application of Hash Functions

Due to their properties, hash functions are well-suited for many data protection operations. However, despite their abundance, they are primarily used in two main areas: data protection and error detection.

It is important to note that different algorithms are used in these two areas. In data protection, there is always the assumption that a malicious attacker may attempt to reveal protected data. Therefore, hash algorithms designed for data protection must resist collisions and prevent reverse-engineering of the initial value from the digest, or more – they must be one-way functions.

Algorithms used for error checking do not need to be collision-resistant or immune to reverse-engineering of initial values. These algorithms detect errors during message transmission through a communication channel or when writing to media. The most used algorithm for error detection is CRC—Cyclic Redundancy Check.

There are two main categories of hash functions:

- those that calculate the digest without a key, using only the message as an input parameter (MDC – Modification Detection Codes).
- those that calculate the digest with the help of a key, using both the key and the message as input parameters (MAC – Message Authentication Codes).

The purpose of MDC functions is to verify integrity in various systems. These functions have additional requirements they must meet. For example, the input and output data must not be correlated in any way; it must be tough to compute input values x_1 and x_2 that differ in a few bits (low entropy

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 2: 2307-2317, 2025
DOI: 10.55214/25768484.v9i2.5072

content), and it must not be easy to deduce parts of the message even if other parts of the message are known. On the other hand, MAC functions, which use a key, have different requirements. In addition to computational simplicity, these functions must satisfy the condition of computational resistance. Computational resistance refers to the inability to find a set of input data that yields the same output value as a set of known input data.

Hash functions play a crucial role in maintaining data integrity during digital forensics. All data must remain unaltered throughout an investigation to serve as valid evidence in court. According to Johansen [19] all digital forensics and evidence collection tools use hash functions. Before any action is taken on potential evidence, the hash value of each document is calculated and stored. At the end of the evidentiary process, the hash is recalculated, and if it does not match the initial value, the evidence may be dismissed due to possible manipulation.

Different hash functions have varying "strengths," which are measured by four criteria [20]:

- Collision resistance – two input values should have no identical output value.
- Output distribution – particularly relevant for non-cryptographic functions, where it is crucial that each possible output value can be generated with the same probability, regardless of any correlation among input values.
- Avalanche effect – a minimal change in input should produce a significantly different output.
- Speed – hash calculation must be performed in real-time.

For example, Aura and Roe [21] analyzes hash functions with an output of less than 128 bits. Such functions are beneficial for securing communication channels, where we aim to minimize the bandwidth used for transmitting "non-useful" data. The study focused on CPU usage during brute-force attacks and the increased cost (CPU load) of hash generation and executing brute-force attacks. The principle they achieved this with is similar to the approach used in slow hash functions, described later in this paper, which is to increase the number of iterations in the hash calculation.

Another study Tchórzewski and Jakóbik [22] theoretically analyzes the strength of selected hash functions approved by NIST, primarily the widely used SHA-1, SHA-2, and SHA-3 functions. The likelihood of specific output bit values given the input, collision resistance, and randomness in output calculation is evaluated through mathematical analysis and three different tests. Given its longer output length and higher computational requirements, the SHA3-512 function performed best, as expected.

4. Methodology and Data Collection

The paper's data collection methodology aimed to assess the vulnerabilities of commonly utilized hash functions under regulated conditions. The researchers created a dataset of hash values employing prevalent cryptographic algorithms, including MD5, SHA-1, and SHA-512. The hash values were generated from plaintext inputs, such as basic passwords and strings, facilitating the assessment of the efficacy of attack techniques in retrieving the original data. The study utilized a recognized plaintext-to-hash correlation to guarantee consistency and reliability in assessing different attack methodologies.

The experiments were performed in a standardized testing environment utilizing a consumer-grade laptop. The configuration comprised an Intel Core i5 2450M processor, 6GB of DDR3 RAM, and an ATI Radeon HD7650M graphics card. Despite its modest specifications, this hardware established a baseline for evaluating various hash-cracking tools' computational requirements and efficacy. The tools being assessed comprised well-known applications like Hashcat, John the Ripper, and Cain & Abel, which were set up to utilize CPU and GPU resources when feasible.

The study generated hash values and simulated real-world conditions by examining various attack scenarios, such as brute-force attacks, dictionary attacks, and rainbow table lookups. Each assault was designed with parameters, including utilizing GPU acceleration and incorporating custom dictionaries. These configurations enabled the researchers to thoroughly investigate the performance and limitations of each method, facilitating a comprehensive analysis of hash function vulnerabilities.

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 2: 2307-2317, 2025
DOI: 10.55214/25768484.v9i2.5072

5. Types of Hash Functions

Hash functions come in different types, each designed to address specific security needs and computational requirements. This section explores fast and slow hash functions, analyzing their strengths, vulnerabilities, and ideal use cases.

5.1. Fast Hash Functions

Certain hash functions achieved significantly faster hash space search results than others on the laptop used for these experiments.

The sample processing speed for NTLM on this setup is 1590 MH/s, or 1590 mega hashes per second, where "mega" represents 106106, or one million. This means the computer used in the project, with the help of graphics processors, can analyze 1.59 billion samples per second, even on a laptop over ten years old. We can only imagine the performance of a system with 16,384 CUDA processors.

Throughout history, numerous hash algorithms have been developed, and some make us question how they were ever approved for use. The LM algorithm, for instance, is outdated and no longer in use. It was popular during the era of Windows XP OS and is specific in that it limits the input parameter to a length of 14 characters, which is split into two inputs of up to 7 characters each. This design flaw makes it highly vulnerable because cracking the hash doesn't require finding a unique 14-character value but rather two 7-character values, significantly speeding up the process. Another major flaw of this algorithm is its case insensitivity; for instance, the hashes for "Test" and "TEST" are identical, which reduces the number of combinations to be examined.

MD5, or Message-Digest Algorithm 5, is arguably the most well-known and widely used hash algorithm. It computes a 128-bit digest and is mainly used for file integrity verification. Designed by Ronald Rivest, also the creator of the highly secure AES algorithm, MD5 vulnerabilities were discovered as early as 1996. Yet, the algorithm is still used primarily for verifying data integrity. Using the Hashcat tool, the search speed for MD5 hash space remains relatively high, around 500 MH/s, or half a billion samples per second.

SHA1, or Secure Hash Algorithm 1, is also frequently used, though no longer considered secure enough for storing user data in databases. SHA1 accepts an input of up to 264 bits and outputs a 160-bit digest. Despite being broken, SHA1 is still deemed relatively secure, as it requires advanced analytical techniques inaccessible to typical internet users. The Hashcat performance for SHA1 in comparable settings on the laptop was about 280 MH/s or 280 million samples per second.

5.2. Slow Hash Functions

The SHA-512 function generates a 128-character hexadecimal digest. The first 256 bits represent the message, while the other 256 bits serve as a security addition (salt) to prevent collisions and improve function security. The search speed for this hash function was only 11 million samples per second, balancing speed and security.

One might wonder why SHA1, a relatively insecure algorithm, is still widely used despite the existence of the NIST-approved SHA-512, which offers a higher level of security. The answer lies primarily in execution speed. Although not a scholarly source, the example on Stack Overflow [23] is illustrative and informative. One user explained why SHA-512 is not universally used in place of SHA1, using the example of establishing a secure VPN connection. It is estimated that if we want to hash all transmitted data on a typical consumer modem with a single 200 MHz CPU core, the data transmission speed would be 6 Mb/s with SHA1. However, using SHA-512 instead would reduce the speed to 1.5 Mb/s.

Another algorithm that is gaining popularity is scrypt. Scrypt became widely known with the rise of cryptocurrencies, as Bitcoin uses SHA-256, while Litecoin uses scrypt. Scrypt is intentionally slower than SHA-256 to enhance security and resistance against brute-force attacks. Scrypt performs many iterations during computation, meaning the processor must compute the hash a specified number of times. Each time it calculates the digest, it becomes the new input value, repeating the process as often as configured. Scrypt's variable iteration count further complicates hash cracking. In addition to requiring substantial processing time, the script also demands considerable memory. A similar function is bcrypt, which is also classified as a slow hash function.

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 2: 2307-2317, 2025
DOI: 10.55214/25768484.v9i2.5072

6. Tools for Cracking Hash Values

Numerous publicly available tools exist for cracking hash values. Most can be run on any operating system and often differ in execution speed—some support parallel processing across multiple CPU cores, while others utilize GPU processing. Below is an overview of the most popular tools, all free to download and use.

6.1. Hashcat

One of the most well-known tools for password cracking and discovering initial hash values is Hashcat. It is a free, open-source program, meaning anyone can download and modify it according to their needs and preferences. The official website describes it as the world's fastest tool, supporting over 160 hash functions with various configurations. It is available on all operating systems and compatible with a wide range of hardware components on the market. Additionally, Hashcat supports cracking multiple hashes simultaneously, parallel processing across various devices, and even distributed processing over a network.

A key advantage of Hashcat is that it supports CPU (central processing unit) and GPU (graphics processing unit) operations. While the CPU can process specific data types much faster, typical consumer CPUs have between 4-8 cores (whether physical or virtual, generated through Intel's hyper-threading technology). High-performance server processors, such as the Intel Xeon Gold series, can have 36 physical or 72 virtual cores [24] and even more in the AMD Ryzen Threadripper series [25]. On the other hand, GPUs have a vastly more significant number of cores. For example, the latest-generation Nvidia RTX 4090 GPU boasts 16,384 CUDA cores [26] which enable parallel processing for specific tasks, such as hash cracking. Although these GPU cores do not have the same capabilities as CPU cores, GPUs are colloquially referred to as rudimentary because they work with a limited instruction set. However, they are particularly effective at cracking hash algorithms.

Suppose we can harness the potential of all GPU cores, which are often underutilized unless a game or graphics-intensive task is running (as modern CPUs usually have built-in graphics). In that case, we can speed up the hash cracking process hundreds of times over, using a single GPU that is far more cost- and energy-efficient than a CPU with a large number of cores.

In addition to GPU power, Hashcat can employ advanced mathematical models, such as Markov chains. A Markov chain is a mathematical model that moves from one state to another according to specific probability rules. Its key feature is that the set of possible future states remains the same regardless of the path to reach the current state. In other words, the likelihood of moving to any given state depends only on the current state and the time elapsed [27]. This system utilizes information theory or the statistical probability that a particular character in a text/password follows another specific character. This significantly narrows down the list of possible passwords, speeding up the discovery process. However, while it accelerates the search, it can also hinder it.

In Fig. 1, ten NTLM hashes were loaded into the application and attacked. For speed, each hash in plaintext contained 4-6 characters, including uppercase and lowercase letters, numbers, and special symbols. The search concluded exceptionally quickly using the GPU, but despite the high speed, the Markov chain search failed to find any initial values. In lay terms, it was a statistical error.

The second scenario, as shown in Fig. 2 I repeated the same test setup and sample but with Markov chains disabled.

GPUs were again used, but the search took over 10 minutes. Although more time-consuming, this method successfully revealed all initial hash values.

6.2. Cain and Abel

Another top-rated, older-generation tool is Cain and Abel. Initially developed for Windows XP, this tool offers a range of functions beyond password cracking. It combines several basic features of the Wireshark program. In addition to capturing and decoding packets, Cain and Abel can analyze Wi-Fi signal quality, assess network security levels, and perform traceroutes to a server. The downside of this tool is that it only supports CPU processing, which significantly slows down its performance.

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 2: 2307-2317, 2025
DOI: 10.55214/25768484.v9i2.5072

In Fig. 1, the cracking of a single NTLM hash composed of 4-6 characters is shown. The method used for the attack is brute force, meaning it tests every possible combination within the 4–6-character range. The computer utilized only a dual-core CPU, using 50% of its total processing power. The estimated time to crack a single hash was 7 hours.

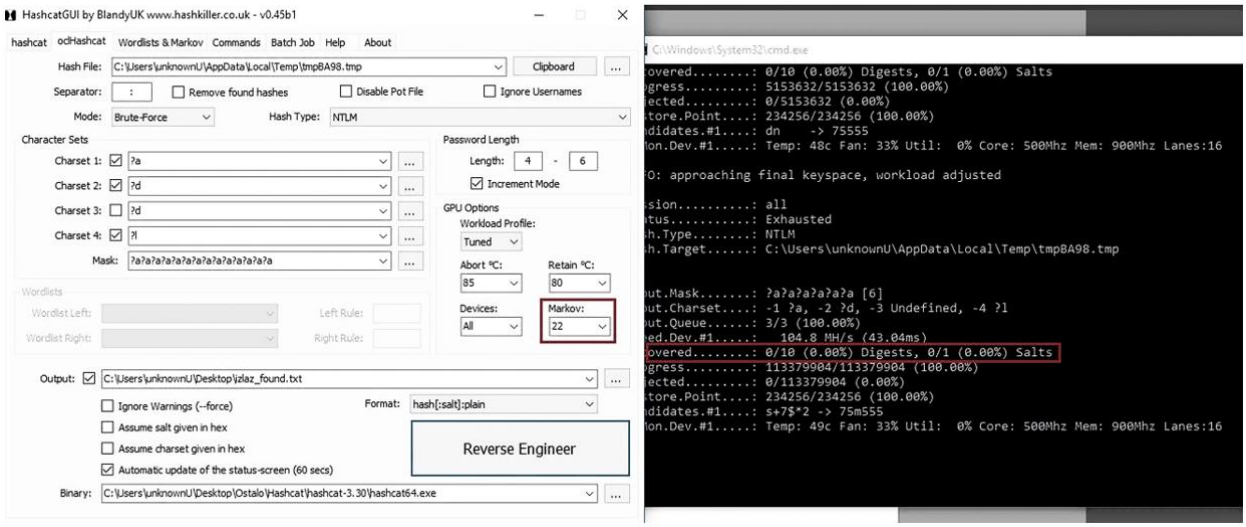

Figure 1.
Cracking passwords using Markov chains.

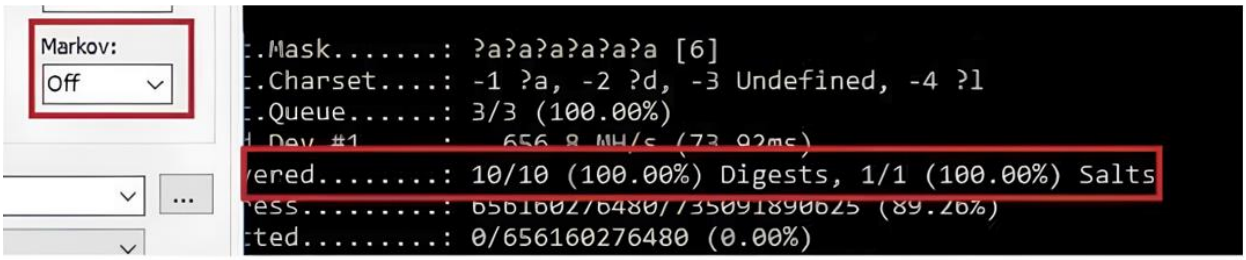

Figure 2.
Cracking passwords without using Markov chains.

In comparison, Hashcat, using a GPU, completed the same task in just over 10 minutes and successfully cracked ten different hashes.

6.3. John the Ripper

John the Ripper, or JtR for short, is another popular tool for cracking hash values. Like Hashcat, JtR can be used on all operating systems and run via the command line or a graphical interface. JtR supports GPU processing but is explicitly optimized for Nvidia CUDA technology. The program supports many different hash algorithms and is known for its speed and reliability. Like Hashcat, JtR allows multiple hashes to be loaded and cracked simultaneously.

7. Techniques for Cracking Hash Values

Multiple approaches are possible when attacking any system. Some are more effective but costly, while others are cheaper (in terms of cost-benefit analysis) but yield poorer results. It is important to note that the goal of any attack remains the same: in this case, finding the plaintext value for a given hash.

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 2: 2307-2317, 2025
DOI: 10.55214/25768484.v9i2.5072

7.1. Brute Force Attacks

Brute force attacks are the oldest and simplest form of attack. They are easy to implement and guarantee accurate results, though execution time can be a concern. Brute force attacks, also called exhaustive searches, systematically test all possible combinations [28]. A traditional brute force attack tests every possible combination. For example, if we have a password that we know is composed only of lowercase English letters and has eight characters, there are precisely 26^8 possible combinations, and the task of a brute force attack is to test them all. There are several improved versions of this approach. One way is to use Markov chains as a statistical model, or backtracking, which employs logic to reduce the number of possible combinations. Execution time can also be shortened if characters are chosen randomly rather than sequentially. If all potential characters are selected sequentially, a password in the form "bcdef" is significantly weaker than one in the form "zzzzz," even though the first has higher entropy with the same number of characters. With random selection, statistically, the password is likely to be discovered after testing just over half of all possible values, assuming the hash algorithm used is known.

7.2. Dictionary Attacks

Another popular attack type is the dictionary attack. As the name intuitively suggests, a dictionary is a predefined set of meaningful words. This attack type is a subset of brute force attacks, which involves testing many combinations. However, the number of combinations tested in this attack is far smaller, as only sensible, meaningful word combinations are considered. The assumption is that users choose logical letter sequences for easier recall. According to one source, the most common passwords are meaningful words like "password," "let me in," or "star wars". Numerous dictionaries containing the most common passwords and their variations are available online. To guess such words through a brute-force attack would require many combinations. Yet, experience shows that most users do not use random character combinations, opting for meaningful strings, making it logical to focus on sensible character sequences.

Despite being faster at finding passwords, dictionaries have a significant limitation: a word meaningful in one language may not exist in another. Additionally, the character set for one language's alphabet may differ significantly from another's. For example, Croatian contains diacritical marks. If a downloaded dictionary is used to try and crack the password "ćiro," there is a high chance it will remain uncracked in most dictionaries due to the letter "ć." Similarly, imagine an attack on a password containing German characters such as "ß," "ü," or "ë," using an English dictionary. Alongside the need for appropriate dictionary localization and its suitability for attacks, disk space usage is another factor to consider. For practical use, the dictionary should ideally be stored locally. While exceptions exist, such as attacks where words are downloaded from the internet and immediately used, then deleted from temporary files, dictionaries can occupy tens to hundreds of gigabytes. This may not seem excessive, but it's important to remember that these dictionaries only contain the most common words of a particular language or region.

7.3. Rainbow Tables

Rainbow tables represent a combination of brute force and dictionary attacks. This concept is best explained using the theory of databases. In databases, one primary key is usually unique for each table row and serves as a unique identifier. Similarly, in rainbow tables, each hash is unique for each row, with a corresponding plaintext value. This attack is by far the fastest. In previous attacks, the computer had to fetch a value, calculate its hash, and compare it with the target values. This step is unnecessary with this attack type, as the hash is precomputed for all table entries. The computer retrieves the precomputed hash and compares it to the target hash. If they match, it only needs to recover the plaintext version from the adjacent column.

Figure 3 Shows an attack on an NTLM hash using the Ophcrack tool, which leverages rainbow tables.

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 2: 2307-2317, 2025
DOI: 10.55214/25768484.v9i2.5072

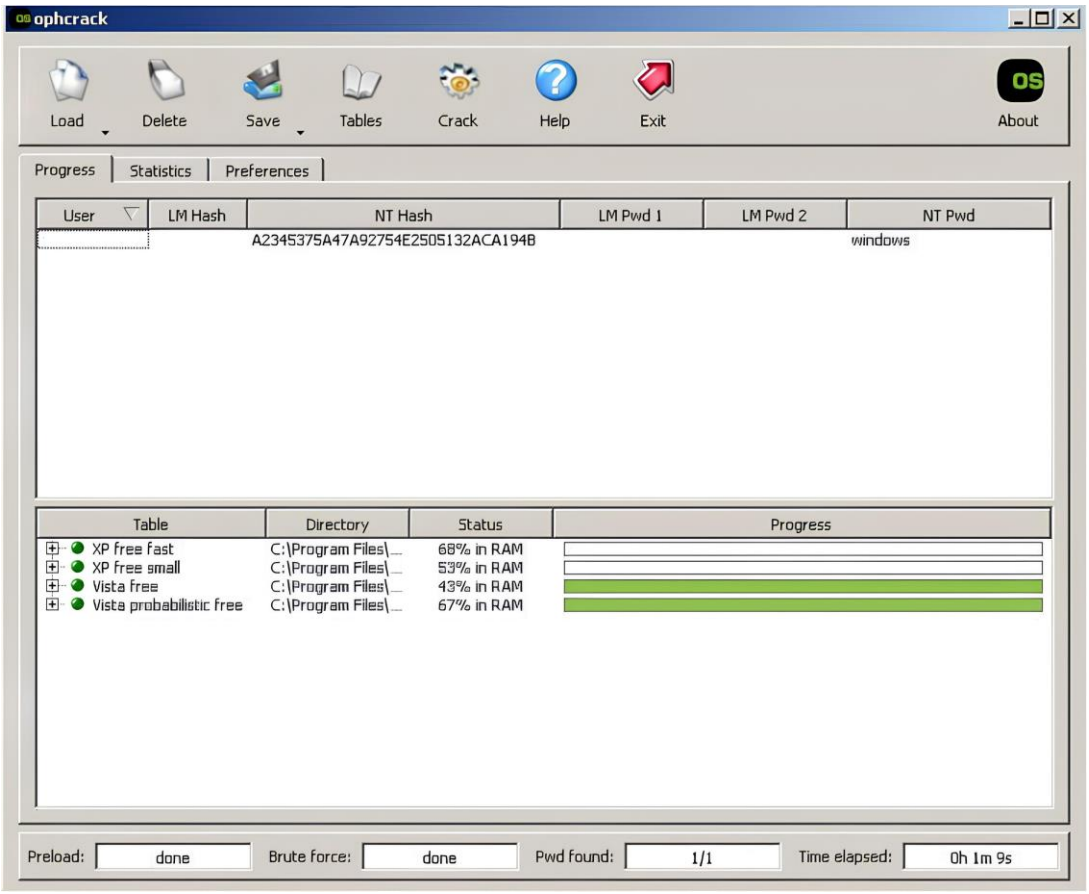

Figure 3.
Rainbow tables in Ophcrack.

Although the password was relatively simple, the tables quickly found the corresponding value. However, this technique has its drawbacks, similar to dictionary attacks. Rainbow tables require significant storage space, and data retrieval from a hard drive creates a bottleneck that drastically slows down the process.

8. Future Research

Future research in hash functions should focus on developing new algorithms that address the vulnerabilities identified in current cryptographic functions. As advancements in computing power make brute-force and dictionary attacks increasingly feasible, especially with GPU and distributed computing systems, exploring hash functions with higher resistance to these attacks is essential. Additionally, there is a need for lightweight cryptographic hash functions that are optimized for resource-constrained devices such as those in IoT environments, which have limited processing power yet require robust security mechanisms. Enhancing the efficiency and scalability of privacy-preserving cracking protocols could also be a valuable study area, especially for third-party services applications. Research may further investigate integrating neural network models into hash functions, potentially leveraging machine learning for adaptive security responses to evolving attack patterns. Lastly, novel approaches to multi-factor credential hashing functions that balance security and latency requirements could strengthen authentication mechanisms without significantly impacting user experience.

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 2: 2307-2317, 2025
DOI: 10.55214/25768484.v9i2.5072

9. Conclusion

This paper's findings highlight the vulnerabilities in commonly utilized hash functions when subjected to contemporary attack methods, including brute-force, dictionary, and rainbow table attacks. Using tools like Hashcat, John the Ripper, and Cain & Abel, our assessment revealed substantial vulnerabilities in hash algorithms, including MD5 and SHA-1. While historically significant, these algorithms exhibit vulnerabilities owing to their comparatively low computational complexity, rendering them susceptible to attacks utilizing modern hardware capabilities. Conversely, although SHA-512 demonstrated superior resistance, it compromised processing speed, highlighting the trade-off between security and efficiency.

The study demonstrated that advancements in GPU and distributed network computing significantly improve the efficiency of hash-cracking techniques in a resource-constrained environment. Our findings indicate that tools utilizing GPU acceleration substantially surpass those limited to CPU operations, demonstrating the influence of hardware optimization on the efficacy of these attacks. Furthermore, examining rainbow tables demonstrated their effectiveness in expedited hash recovery, although this comes with significant storage demands.

These findings have significant ramifications for both scholarly and practical applications. They emphasize the necessity of transitioning from obsolete hash functions to more secure alternatives, such as SHA-3 or slower hash functions like bcrypt and scrypt, particularly in high-security environments. The study underscores the importance of incorporating advanced cryptographic methods like salt and multi-factor authentication to reduce risks linked to predictable or brute-force attack scenarios.

This study enhances the discourse on cryptographic security by elucidating the performance attributes of diverse attack methodologies and emphasizing the imperative for ongoing advancement in hash function design. Future research should investigate the creation of hash algorithms that optimize computational efficiency while improving resistance to both conventional and novel attack vectors. Such endeavors are crucial to protect the integrity and confidentiality of digital systems from the relentless advancement of computational power.

Transparency:

The authors confirm that the manuscript is an honest, accurate, and transparent account of the study; that no vital features of the study have been omitted; and that any discrepancies from the study as planned have been explained. This study followed all ethical practices during writing.

Copyright:

References

[1] K. Jangid and Vidushi, "S-HASH: A crack towards cryptographic hash functions," *CSVTU Research Journal on Engineering and Technology*, 2021. https://doi.org/10.30732/csvturj.20211001005

[2] R. Kundu and A. Dutta, "Cryptographic Hash functions and attacks-a detailed study," *International Journal of Advanced Research in Computer Science*, vol. 11, no. 2, pp. 37-44, 2020. https://doi.org/10.26483/ijarcs.v11i2.6508

[3] P. J. F. Bemida, A. M. Sison, and R. P. Medina, "Modified SHA-512 algorithm for secured password hashing," presented at the 2021 Innovations in Power and Advanced Computing Technologies (i-PACT), 2021.

[4] N. Tihanyi, T. Bisztray, B. Borsos, and S. Raveau, "Privacy-preserving password crackin: How a third party can crack our password Hash without learning the Hash value or the cleartext," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 2981-2996, 2024. https://doi.org/10.48550/arXiv.2306.08740

[5] K. Marchetti and P. Bodily, "John the Ripper: An examination and analysis of the popular hash cracking algorithm," presented at the 2022 Intermountain Engineering, Technology and Computing (IETC), 2022.

[6] S. Windarta, S. Suryadi, K. Ramli, B. Pranggono, and T. S. Gunawan, "Lightweight cryptographic hash functions: Design trends, comparative study, and future directions," *Ieee Access*, vol. 10, pp. 82272-82294, 2022. https://doi.org/10.1109/ACCESS.2022.3195572

[7] I. El Hanouti, H. El Fadili, S. Hraoui, and A. Jarjar, "A lightweight hash function for cryptographic and pseudo-cryptographic applications," in *WITS 2020: Proceedings of the 6th International Conference on Wireless Technologies, Embedded, and Intelligent Systems, Springer Singapore*, 2022, pp. 495-505.

[8] A. Sideris, T. Sanida, and M. Dasygenis, "A novel hardware architecture for enhancing the keccak hash function in fpga devices," *Information*, vol. 14, no. 9, p. 475, 2023. https://doi.org/10.3390/info14090475
[9] J. Rudy and P. Rodwald, "Job scheduling with machine speeds for password cracking using hashtopolis," presented at the International Conference on Dependability and Complex Systems, Cham: Springer International Publishing, 2020.
[10] Y. Wang and N. Gu, "Classification of hash functions based on anti-attack ability," presented at the 2023 International Conference on Networking and Network Applications (NaNA), 2023.
[11] D. Upadhyay, N. Gaikwad, M. Zaman, and S. Sampalli, "Investigating the avalanche effect of various cryptographically secure hash functions and hash-based applications," *IEEE Access*, vol. 10, pp. 112472-112486, 2022. https://doi.org/10.1109/ACCESS.2022.3215778
[12] A. Nugroho and T. Mantoro, "Salt hash password using MD5 combination for dictionary attack protection," *2023 6th International Conference of Computer and Informatics Engineering (IC2IE)*, pp. 292-296, 2023. https://doi.org/10.1109/IC2IE60547.2023.10331606
[13] V. Nair and D. Song, "Multi-factor credential hashing for asymmetric brute-force attack resistance," presented at the 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P), 2023.
[14] F. Kpieleh, "Cryptographic hash functions for digital stamping," *Advances in Multidisciplinary and Scientific Research Journal Publication*, 2022. https://doi.org/10.22624/aims/digital/v10n4p9
[15] M. R. Anwar, D. Apriani, and I. R. Adianita, "Hash algorithm in verification of certificate data integrity and security," *Aptisi Transactions on Technopreneurship*, vol. 3, no. 2, pp. 181-188, 2021. https://doi.org/10.34306/att.v3i2.212
[16] V. Soni, D. P. Bhatt, and N. Yadav, "An efficient approach of NeuroHash and its comparison with cryptographic hash," presented at the 2020 8th International Conference on Reliability, Infocom Technologies and Optimization, 2020.
[17] A. Mittelbach and M. Fischlin, *The theory of hash functions and Random oracles*. Springer. https://doi.org/10.1007/978-3-030-63287-8, 2021.
[18] Algorithms for Calculating the Summary, "CCERT-PUBDOC-2006-08-166," Retrieved: https://www.cert.hr/wp-content/uploads/2006/08/CCERT-PUBDOC-2006-08-166.pdf. [Accessed Nov. 3, 2024], 2024.
[19] G. Johansen, *Digital forensics and incident response*. United Kingdom: Packt Publishing Ltd, 2017.
[20] C. Estébanez, Y. Saez, G. Recio, and P. Isasi, "Performance of the most common non-cryptographic hash functions," *Software: Practice and Experience*, vol. 44, no. 6, pp. 681-698, 2014. https://doi.org/10.1002/spe.2179
[21] T. Aura and M. Roe, "Strengthening short hash values," Retrieved: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=47d90a3ceffe2b6d353201e29b6ee8f442c2dc52. [Accessed Nov. 3, 2024], 2024.
[22] J. Tchórzewski and A. Jakóbik, "Theoretical and experimental analysis of cryptographic hash functions," *Journal of Telecommunications and Information Technology*, no. 1, pp. 125-133, 2019. https://doi.org/10.26636/jtit.2019.128018
[23] Why use SHA1, "Why use SHA1 for hashing secrets when SHA-512 is more secure," Stack Overflow," Retrieved: https://stackoverflow.com/questions/2640566/why-use-sha1-for-hashing-secrets-when-sha-512-is-more-secure, 2010.
[24] Intel Xeon Gold, "Intel Xeon gold," Retrieved: https://www.intel.com/content/www/us/en/products/sku/237263/intel-xeon-gold-6554s-processor-180m-cache-2-20-ghz/specifications.html. [Accessed Nov. 3, 2024], 2024.
[25] AMD Ryzen Threadripper, "AMD Ryzen Threadripper," Retrieved: https://www.amd.com/en/products/processors/workstations/ryzen-threadripper.html#specs. [Accessed Nov. 3, 2024], 2024.
[26] Nvidia RTX4090, "Nvidia RTX4090," Retrieved: https://www.nvidia.com/en-eu/geforce/graphics-cards/40-series/rtx-4090/. [Accessed Nov. 3, 2024], 2024.
[27] M. Predovan, "Markov chains," Final Thesis, University of Rijeka, Faculty of Engineering, 2024.
[28] Brute Force Napadi, "CCERT-PUBDOC-2007-08-201," Retrieved: https://www.cert.hr/wp-content/uploads/2019/04/CCERT-PUBDOC-2007-08-201.pdf. [Accessed Nov. 3, 2024], 2024.