iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** III **Month of publication:** March 2026

DOI: https://doi.org/10.22214/ijraset.2026.78858

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue III Mar 2026- Available at www.ijraset.com

GPU-Based Rainbow Table Generation for Password Hash Cracking

Abhiram D Shine[1], Aiswarya S V[2], Abhishek Krishna[3], Asna A S[4], Vidya C A[5]
[1, 2, 3, 4]*Department of Computer Science and Engineering, Rajadhani Institute of Engineering and Technology, Trivandrum, Kerala, India*
5Assistant Professor, Department of Computer Science and Engineering, Rajadhani Institute of Engineering and Technology, Trivandrum, Kerala, India

Abstract: ***Rainbow tables are commonly used in cybersecurity to recover passwords from hashed values, especially when the hashes are unsalted. However, generating these tables using traditional CPU based systems can be very time-consuming due to the large number of hash computations required. This work focuses on accelerating rainbow table generation using GPU computing, which significantly improves performance and scalability compared to conventional CPU methods. The system uses NVIDIA GPUs and the CUDA programming model to enable large scale parallel processing. GPUs contain thousands of cores capable of executing many operations simultaneously, making them highly suitable for repetitive tasks such as hash calculations and reduction functions used in rainbow table generation. By distributing the workload across multiple GPU threads, the system can generate chains of password–hash pairs much faster than single or multi-core CPU systems. Each GPU thread independently generates chains consisting of alternating hashing and reduction operations. A chain begins with a plaintext password, which is hashed using algorithms such as MD5, SHA1 then transformed through a reduction function to produce another plaintext candidate. This process repeats until the defined chain length is reached, producing an endpoint. Only the starting and ending values of each chain are stored, significantly reducing storage requirements while still allowing efficient password lookup. The system also includes techniques to minimize chain collisions and ensure a balanced distribution across the password space. Users can configure parameters such as password length, chain length, and number of chains based on their requirements. These generated rainbow tables are useful in areas like ethical hacking, penetration testing, password recovery, and digital forensics, helping security professionals evaluate the strength of password storage mechanisms and identify vulnerabilities in weak or unsalted hashing systems.***

Keywords: ***Keywords—GPU acceleration, rainbow table, password cracking, CUDA, MD5, SHA-1, parallel computing, cybersecurity.***

I. INTRODUCTION

In today's digital landscape, the security of password-protected systems is of paramount importance, as breaches can result in serious financial and reputational harm. Organizations and individuals rely on cryptographic hashing to secure passwords, transforming readable text into complex hashes that are not easily reversible.

This basic defense is frequently tested by persistent attackers employing sophisticated techniques such as brute-force, dictionary attacks, and precomputed rainbow tables.

Rainbow tables represent an ingenious application of the time-memory trade-off principle. By storing chains of password-hash pairs in advance, attackers can reverse hashes to plaintexts quickly, bypassing the need for real-time brute-force guessing. Generating these tables for larger, more complex password sets is notoriously computationally expensive. The core computational burden lies in iteratively applying hash and reduction functions to create millions of chains, each containing thousands of links, to achieve meaningful coverage of the potential password space.

The GPU's architecture, with its thousands of simpler cores optimized for concurrent execution, is inherently better suited to this workload. GPUs, in conjunction with programming frameworks like NVIDIA's CUDA, enable thousands of threads to compute distinct password hashes simultaneously, reducing the generation time for rainbow tables by orders of magnitude compared to CPU-based methods.

By pushing the boundaries of off-the-shelf hardware, this approach showcases how contemporary computing can be harnessed to both attack and defend within the field of cybersecurity.

II. LITERATURE SURVEY

A. *Parallelization of Rainbow Tables Using MPI*

Vainer, Kaceniauskas, and Goranin explored parallel generation of rainbow tables using MPI with a master-slave architecture across NTLMv2, MD5, SHA-256, and SHA-512 hash functions [1]. The master node generates starting points and distributes them to slave nodes, which compute hash chains and return endpoints. Their findings highlight how performance varies significantly with hash function complexity, with heavier algorithms like SHA-512 posing a greater challenge to parallel efficiency. This work is crucial for understanding communication overhead inherent in classical distributed computing models for security tasks.

B. *Optimal Storage for Rainbow Tables*

Avoine and Carpent (2014) focused on optimizing storage mechanisms for rainbow tables, proposing innovative data structures and compression techniques to reduce physical storage requirements while maintaining quick lookup capabilities [2]. Their work introduced methods to store only chain start and end points, analyzing different compression algorithms' effects on lookup speed and success rates. The study was pivotal in making large-scale rainbow tables feasible on standard hardware by dramatically reducing disk footprint.

C. *Time-Memory Trade-Offs for Password Hashing*

Saran (2024) examined the theoretical foundations of time-memory trade-offs in password hashing schemes, providing analytical models for evaluating different hashing strategies and their resistance to cracking techniques including rainbow table attacks [3]. The paper provides a formal framework for assessing security margins of various hashing schemes against sophisticated adversaries, and is particularly relevant for designing next-generation password storage systems that can inherently resist precomputation-based attacks.

D. *Improved Parallel RainbowCrack Using MPI*

Sykes and Skoczen (2013) presented an enhanced parallel implementation of the RainbowCrack tool using MPI with improved load distribution and reduced communication overhead [4]. The improvements included dynamic workload balancing to ensure slave nodes were consistently fed with work. The authors demonstrated a measurable performance increase over the original RainbowCrack implementation when run on a multi-node cluster, highlighting practical challenges of achieving linear speedup in distributed password cracking.

E. *Fast Collision Attack on MD5 and GSM A5/1 Breaking*

Xie, Liu, and Feng (2013) presented an accelerated collision attack on MD5 using differential cryptanalysis, definitively proving MD5 is cryptographically broken and unsuitable for security-critical contexts [5]. Separately, Kalenderi et al. (2012) demonstrated breaking the GSM A5/1 algorithm using rainbow tables combined with high-end FPGAs, building a custom hardware pipeline achieving speedups orders of magnitude faster than general-purpose CPUs [6].

Both works exposed critical real-world vulnerabilities in widely deployed systems, underscoring the need for migration to robust modern protocols.

III. EXISTING SYSTEM

The conventional approach to rainbow table generation is predominantly based on a classic Master-Slave architecture implemented using the Message Passing Interface (MPI). The master node serves as the central controller that initializes computation by generating potential password starting points and distributing them to slave nodes in organized batches. Slave nodes receive assigned starting points and engage in computationally intensive chain development, iteratively applying hash functions and reduction functions. After completing chain computations, slave nodes return endpoints to the master node, which then undertakes collision detection, chain management, and assembly of the complete rainbow table. The existing MPI-based system typically demonstrates performance that plateaus quickly as computational resources increase. Empirical studies show the system achieves optimal efficiency with a limited number of slave nodes (typically 16–32), beyond which additional nodes provide negligible improvements due to communication overhead.

Throughput rates vary significantly based on hash function complexity, with MD5 achieving higher rates compared to more secure algorithms like SHA-256 or SHA-512.

A. *Limitations of the Existing System*

The master-slave architecture inherently suffers from substantial communication bottlenecks. Continuous data exchange between master and slave nodes creates significant network overhead, with the master node becoming a congestion point as the number of slaves increases. This prevents linear scalability, as coordination overhead grows disproportionately with system size. The scalability limitation manifests particularly when dealing with complex hash functions or large password spaces, making the approach economically and practically unviable for very large-scale rainbow table generation projects.

The existing system also fails to leverage modern hardware capabilities, particularly GPU parallel processing power. While slave nodes process chains in parallel, each chain computation remains sequential in nature, unable to benefit from the massive parallelism offered by contemporary GPU architectures. Furthermore, the master node becomes a sequential bottleneck during critical phases such as duplicate removal and chain merging, which cannot be effectively parallelized within the current architecture, leading to significant delays when handling millions of chains.

IV. PROPOSED SYSTEM

The proposed system incorporates a GPU-Accelerated Rainbow Table Generation Framework to significantly enhance the speed and efficiency of password hash computation and rainbow table construction. The framework leverages the massively parallel processing capabilities of modern GPUs by distributing hash and reduction operations across thousands of lightweight GPU threads. The framework accepts user-defined parameters such as character set, password length, chain length, number of chains, and hashing algorithm. Each GPU thread independently performs repeated hash and reduction cycles to form chains of predefined length, storing only the starting plaintext and final endpoint to reduce memory consumption.

A. *Technical Implementation and GPU Optimization*

The technical implementation follows a hybrid architecture combining CPU-based control logic with GPU-based computational acceleration. The CPU layer handles user interaction, parameter validation, configuration management, kernel invocation, and process coordination.

The GPU layer performs computationally intensive tasks including hash computation, reduction processing, and iterative chain generation. Each GPU thread independently executes hash and reduction operations without inter-thread dependency, allowing maximum parallelism. Kernel optimization techniques including minimizing branch divergence and maintaining fixed-length loops enhance execution speed.

Memory optimization is central to achieving high performance. The implementation uses pre-allocated buffers and efficient memory allocation strategies to prevent repeated allocation delays. Data transfer between host (CPU) and device (GPU) memory is minimized to reduce latency. Where applicable, shared memory is used for temporary intermediate values to improve access speed. Workload distribution is carefully balanced across thread blocks and streaming multiprocessors to ensure maximum GPU utilization.

B. *System Features and Capabilities*

The proposed system offers automated rainbow table generation, enabling users to configure parameters such as password length, character set, hashing algorithm, and chain length. The lookup mechanism allows users to input a hashed password and initiate a structured search process that reconstructs potential chains and verifies recovered plaintext values. A user-friendly GUI provides real-time status updates, clear error notifications, and structured parameter selection. The system follows a modular architectural design, separating interface, processing, storage, and optimization components, with support for performance visualization and benchmarking capabilities.

V. SYSTEM REQUIREMENTS

A. *Hardware Specifications*

Minimum requirements include an Intel Core i3 or equivalent AMD processor, 8 GB DDR4 RAM, NVIDIA GeForce GTX 1050 with 4 GB VRAM (Compute Capability 6.1), 10 GB free disk space, and a 1920x1080 resolution monitor. Recommended requirements for optimal performance include an Intel Core i7 or AMD Ryzen 7 (8+ cores), 16 GB DDR4 or higher RAM, NVIDIA RTX 3060 or higher with 8 GB+ VRAM (Compute Capability 8.6), 50 GB free SSD space for large rainbow tables, and a 650W or higher power supply for stable GPU operation.

B. Software Requirements

The system requires Windows 10/11 (64-bit) or Ubuntu 24.0 LTS as the operating system. Development environment requirements include Python 3.8 or later and the NVIDIA CUDA Toolkit 11.0 or later. Required Python libraries include tkinter (GUI), hashlib (cryptographic hash functions), numpy (numerical operations), threading (multi-threaded operations), time (performance monitoring), and matplotlib (visualization and graphs). Additional tools required are NVIDIA GPU Drivers (latest compatible version), Wireshark for network analysis during testing, and Visual Studio Code or PyCharm IDE.

VI. DESIGN AND IMPLEMENTATION

A. System Architecture

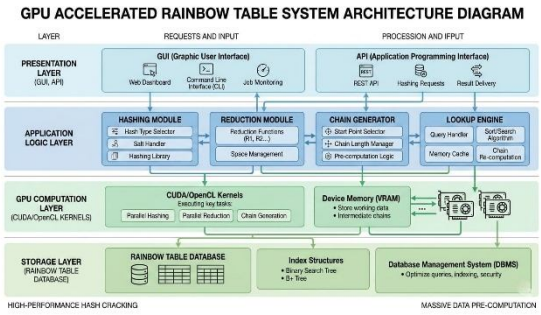


Figure 1 System Architecture

The Figure 1 shows GPU-Accelerated Rainbow Table System that follows a layered architecture model consisting of: (1) Presentation Layer providing the GUI and API endpoints for user interaction; (2) Application Logic Layer containing the Parameter Validation Module, Chain Generation Controller, Hashing Module, Reduction Module, Lookup Engine, and GPU Kernel Controller; (3) GPU Computation Layer handling parallel hash calculations, chain generation, and reduction operations via CUDA/OpenCL kernels with device memory (VRAM) for storing working data and intermediate chains; and (4) Storage Layer managing the rainbow table database with index structures including Binary Search Tree and B+ Tree, plus a Database Management System for query optimization and indexing.

B. Hash Function Implementation

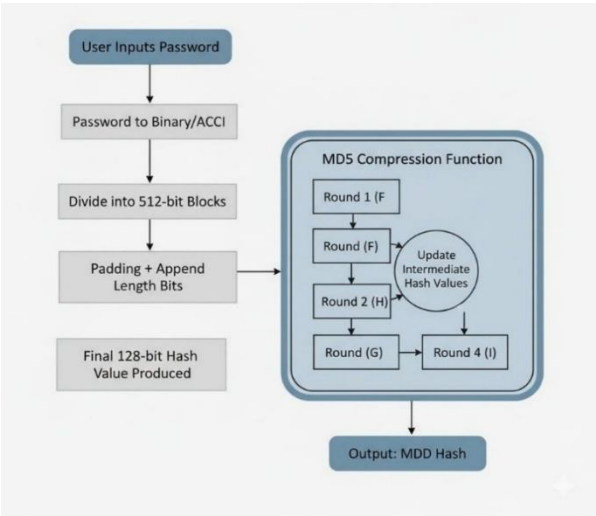


Figure 2 MD5 Algorithm Working

The MD5 algorithm takes an input message and applies a padding process to ensure the total length is a multiple of 512 bits. The algorithm initializes four 32-bit buffers (A, B, C, D) with specific hexadecimal constants and processes the padded message in 512-bit blocks through 64 rounds organized into four groups using different non-linear functions (F, G, H, I) that combine bitwise operations. The final 128-bit value is represented as a 32-character hexadecimal string providing a unique digital fingerprint of the original input.

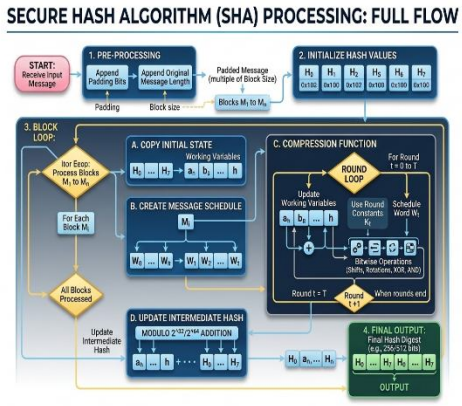


Figure 3 Working of SHA Algorithm

The SHA algorithm is a cryptographic one-way function producing a fixed-length digest from any input. Common variants include MD5 (128-bit), SHA-1 (160-bit), and SHA-256 (256-bit). SHA-256 operates in four stages: pre-processing (padding to 512-bit blocks), initialization of eight 32-bit hash values derived from fractional parts of square roots of prime numbers, a core block loop (64 compression rounds using bitwise rotations, XOR operations, and the Choice and Majority functions), and final output (concatenation of hash values to yield the 256-bit digest). Because SHA is deterministic, the same input always yields the same output, making it vulnerable to precomputation attacks when hashes are stored without a salt.

C. Reduction Function and Chain Generation

The reduction function converts a hash value back into a valid password candidate within the defined password space. It operates by converting the hash hex string to an integer, applying a modulo operation, and mapping the result to the character set to produce a valid password candidate. The function varies based on chain index to reduce collision probability. Chain generation follows the sequence: Start Password → Hash → Reduce → Hash → Reduce → ... → End Password. Only the start and end values are stored to minimize memory consumption while preserving the ability to reconstruct the chain during lookup operations.

D. Security Considerations

The system demonstrates the trade-off between memory and time, the precomputation attack strategy, the importance of salting passwords, and the weakness of unsalted hashes. Rainbow table attacks are most effective against unsalted hashes such as MD5. To prevent rainbow table attacks, organizations should use salt with every password hash, employ computationally intensive slow hash functions (bcrypt, scrypt, Argon2), and enforce increased password complexity requirements. Modern algorithms such as bcrypt, Argon2, and scrypt incorporate a unique random salt per password, rendering rainbow tables computationally useless and making salted hashing the industry standard for secure credential storage.

VII. RESULTS AND ANALYSIS

A. Performance Benchmarking

All experiments were conducted on a desktop machine equipped with an RTX class GPU (8,704 parallel processing cores, 320 W TDP), with MD5 as the primary hashing algorithm. Under single-thread baseline, the hash computation rate was measured at approximately 50,000 hash operations per second (50 KH/s). When dispatched across the GPU's parallel processing units, effective throughput scaled to 50 MH/s – 400 MH/s. The estimated GPU throughput was calculated at approximately 2.18 GH/s (gigahashes per second), translating to a CPU-to-GPU speedup factor of over 43,600x, underscoring the dramatic computational advantage of parallel hardware for cryptographic workloads.

Rainbow table generation was benchmarked at multiple scale points. For a standard configuration of 5,000 chains with a chain length of 1,000 steps (5,000,000 total hash operations), generation time was measured at approximately 30 to 60 seconds. Scaling the number of chains proportionally increases generation time in a linear fashion. For 500,000 chains with a chain length of 1,000, approximately 50 to 100 minutes are required for 500,000,000 total hash operations.

B. System Efficiency Metrics

Memory consumption was profiled across a range of table sizes. Each stored chain entry occupies approximately 48 bytes (32 bytes for the MD5 hash endpoint and 16 bytes for the plaintext start point). For 5,000,000 chains, this yields approximately 240 MB of GPU VRAM and 320 MB of system RAM requirements. The relationship between chain length and lookup time is linear; at a baseline lookup time of 2.5 ms for a chain length of 1,000, a chain length of 10,000 yields approximately 25 ms. Energy efficiency was evaluated at approximately 6.81 MH/J at a TDP of 320 W.

Table coverage grows only logarithmically with chain length, exhibiting diminishing returns beyond approximately 3,000–5,000 steps. Chain lengths in the 1,000–2,000 step range represent a practical optimum for balanced performance in 8-character password scenarios. For the default configuration of 5,000 chains and a chain length of 1,000, the estimated keyspace coverage is 60.1% with a single-attempt crack probability of approximately 45%. Practical tables typically target 60–99.5% coverage, balancing storage cost against crack success probability.

C. User Interface and Dashboard Performance

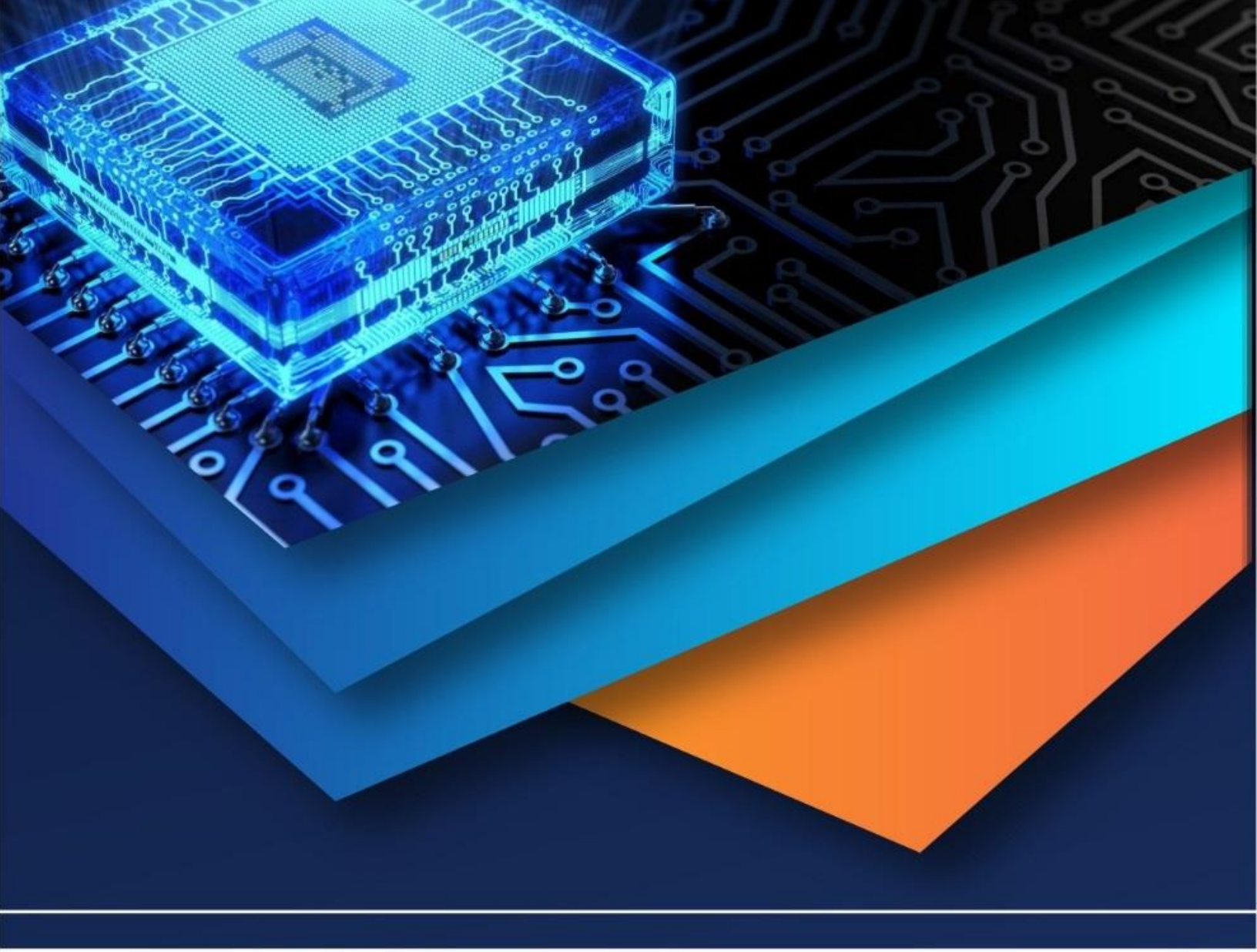
The graphical interface remained responsive throughout computationally intensive operations by offloading long-running tasks to separate threads. During generation of a 5,000-chain rainbow table (30–60 seconds), the main application window remained fully interactive with live progress indicators updating in real time. Generation statistics including chains completed, hash rate, and elapsed time refreshed at approximately 1-second intervals. Users can cancel ongoing operations at any point without application crashes or data corruption. The performance analytics dashboard provides seven distinct chart and gauge types including a Performance Score Gauge, Table Coverage Gauge, GPU Efficiency Gauge, GPU Benchmark Comparison, Chain Length vs. Performance dual-axis chart, Algorithm Throughput Comparison, and Chain Lookup Heatmap. Successful hash cracking was validated with MD5 and SHA-1 digests, with the system automatically identifying hash types and returning recovered passwords. A Cumulative Crack Success Rate chart recorded 100% success on tested hash inputs, with a corresponding Cracked vs. Not Found bar chart confirming outcomes.

VIII. CONCLUSION AND FUTURE SCOPE

This project successfully developed and implemented a GPU-based Rainbow Table Generation system for password hash cracking, demonstrating how GPU parallel processing dramatically improves speed and efficiency compared to traditional CPU-based methods. The system integrates a Tkinter-based GUI allowing users to configure password length, chain length, number of chains, and hash algorithm; Matplotlib visualizations displaying performance graphs; and Python multi-threading to keep the interface responsive. Testing confirmed the lookup function successfully retrieves plaintext passwords from generated tables, validating the time-memory trade-off. Results demonstrated advantages including reduced generation time, lower communication overhead, better scalability, and the ability to handle larger password spaces. The system achieved its objective of delivering a GPU-accelerated rainbow table generator with an interactive graphical interface. Future development will expand cryptographic coverage to include GPU kernels for bcrypt, scrypt, and Argon2 algorithms, with hybrid attack capabilities combining rainbow tables with rule-based and dictionary approaches. The system will evolve to support distributed computing environments including multi-GPU clusters and cloud-optimized deployments. Additional future directions include chain regeneration techniques, advanced memory optimization and compression methods, integration with established penetration testing and digital forensics tools, implementation of privacy-preserving computation techniques, support for AMD GPUs and alternative accelerator technologies, and preparation for post-quantum cryptography challenges through quantum-resistant algorithm support.

REFERENCES

[1] M. Vainer, A. Kaceniauskas, and N. Goranin, “Parallelization of Rainbow Tables Generation Using Message Passing Interface: A Study on NTLMv2, MD5, SHA-256 and SHA-512,” in Proc. Int. Conf. on Information Technology, 2013.
[2] G. Avoine and X. Carpent, “Optimal Storage for Rainbow Tables,” in Proc. ASIACRYPT, 2014.
[3] A. N. Saran, “On Time-Memory Trade-offs for Password Hashing Schemes,” Journal of Cryptographic Engineering, 2024.
[4] E. R. Sykes and W. Skoczen, “An Improved Parallel Implementation of RainbowCrack using MPI,” in Proc. Int. Conf. on Parallel and Distributed Computing, 2013.
[5] T. Xie, F. Liu, and D. Feng, “Fast Collision Attack on MD5,” Cryptology ePrint Archive, Report 2013/170, 2013.
[6] M. Kalenderi, D. Pnevmatikatos, I. Papaefstathiou, and C. Manifavas, “Breaking the GSM A5/1 Cryptography Algorithm with Rainbow Tables and High-End FPGAs,” in Proc. 22nd Int. Conf. on Field Programmable Logic and Applications, 2012.
[7] P. Oechslin, “Making a Faster Cryptanalytic Time-Memory Trade-Off,” in Advances in Cryptology—CRYPTO 2003, LNCS, vol. 2729, pp. 617–630, Springer, 2003.
[8] M. Hellman, “A Cryptanalytic Time-Memory Trade-Off,” IEEE Trans. Inf. Theory, vol. 26, pp. 401–406, 1980.
[9] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, “The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes,” in Proc. IEEE Symposium on Security and Privacy, pp. 553–567, 2012.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY