# Evaluating Performance and Security Trade-offs in Modern Password Hashing Algorithms

Christian Chung
cc72574@eid.utexas.edu
UT Austin, TX

***Abstract*—Password hashing algorithms play a critical role in protecting user credentials in modern authentication systems. As data breaches continue to escalate and attackers gain access to increasingly powerful hardware, the security of stored passwords depends heavily on the choice and configuration of the hashing algorithm.**

**The core problem is that traditional hashing algorithms such as SHA-256 are designed for speed, making them vulnerable to rapid brute-force attacks, while modern slow-hash algorithms offer stronger protection but require higher computational costs. This project aims to experimentally compare the performance of SHA-256, bcrypt, scrypt, and Argon2 on a consumer laptop to illustrate their security and performance trade-offs. For the initial report, the focus will be on conducting controlled experiments to measure hashing time, memory usage, and CPU-based brute-force resistance. The goal is to determine how algorithm choice alone can significantly strengthen password storage practices.**

**All measured results presented in this report are based on actual benchmark runs rather than initial draft values.**

## 1 Introduction

The increasing reliance on digital services has led to widespread storage of sensitive user information across the web. As a result, password database breaches have become a frequent and damaging occurrence, exposing millions of hashed passwords to attackers. In an era where commodity hardware can compute billions of hashes per second, the security of users' accounts depends not only on password strength but also on the choice of hashing algorithm used by service providers.

The central problem is that many systems still rely on fast hashing algorithms such as SHA-256 because they are easy to implement and computationally efficient. Unfortunately, this efficiency enables attackers to perform rapid brute-force attacks with minimal cost. Although stronger algorithms like bcrypt, scrypt, and Argon2 exist and are widely recommended, developers often lack a clear, practical understanding of how these algorithms differ in security and performance. This gap results in insecure deployments that fail to leverage available defenses.

This project introduces an experimental approach to quantify the performance and security differences among common password hashing algorithms. The key insight is that security can be improved not only through user behavior (e.g., stronger passwords) but also through algorithmic cost parameters that intentionally slow down hashing operations. By empirically demonstrating how hashing cost scales with algorithm design, the project provides actionable guidance for secure password storage.

To achieve this, the project evaluates four hashing algorithms—SHA-256, bcrypt, scrypt, and Argon2—on a standard consumer laptop. The experiments will measure time per hash across various parameters, memory requirements for memory-hard algorithms, CPU-based brute-force time estimates for breaking passwords, and brute-force time for small keyspaces.

This project builds directly on prior work by empirically testing the practical performance differences among these algorithms on modern hardware. The goal is not to design a new hashing algorithm, but to extend existing research by demonstrating these differences in a small-scale, reproducible environment suitable for educational purposes.

All source code, experimental scripts, and benchmarking tools used in this study are publicly available for reproducibility at https://github.com/ChristianJinyoung/ECE379K-project.git.

## 2 Motivation

Password authentication remains one of the most widely deployed mechanisms for securing user accounts, yet it continues to be a major source of vulnerability across modern systems. Industry reports consistently show that a significant percentage of data breaches involve stolen or weakly protected passwords, and attackers today have access to inexpensive GPUs, cloud computing clusters, and specialized

[0] Project repository and source code available at: https://github.com/ChristianJinyoung/ECE379K-project.git

cracking tools capable of evaluating billions of hash computations per second. In this environment, the practical security of a password is determined not only by its entropy but also by the computational and memory cost imposed by the hashing algorithm protecting it.

Traditional hashing functions such as SHA-256 and MD5 were designed for general-purpose integrity checks and cryptographic primitives—not for password storage. Their speed, which is desirable in many cryptographic applications, makes them fundamentally unsuitable for resisting offline brute-force attacks. As hardware has improved, the gap between fast-hash algorithms and attacker capability has widened dramatically. This has motivated the development of "slow" password hashing schemes, including bcrypt, PBKDF2, scrypt, and later Argon2, which incorporate tunable cost parameters to deliberately increase computation time and, in some cases, memory usage.

Despite substantial theoretical work, developers frequently lack practical, data-driven intuition about how these algorithms behave under real-world conditions. Many system designers default to insecure fast hashes due to convenience or unfamiliarity with the performance implications of slow-hash algorithms. As a result, insecure deployments persist even when more secure alternatives are available and standardized.

This project is motivated by the need to bridge this gap with empirical evidence. By benchmarking SHA-256, bcrypt, scrypt, and Argon2id on a consumer laptop, the work aims to illustrate how algorithm choice alone can increase brute-force resistance by several orders of magnitude. The results directly support best practices recommended by standards bodies and security organizations, while providing a reproducible methodology for understanding password hashing trade-offs in practical environments.

# 3 Proposed Design or Architecture

The system under evaluation consists of four password hashing implementations running on a single consumer laptop. The design centers around three core components:

1) **Hashing Benchmark Module**
   - Inputs: password strings, algorithm parameters
   - Outputs: average time per hash
   - Implements SHA-256 via hashlib, bcrypt, scrypt, and Argon2 via Python Libraries
2) **Brute-force Simulation Module**
   - Generates password candidates
   - Measures time required for a CPU-only brute force attempt
   - Supports both numeric and alphanumeric keyspaces
3) **Analysis Engine**
   - Aggregates results
   - Produces comparative metrics
   - Generates draft and final figures
4) **Small Keyspace Brute-Force Demonstration**
   - Select a target 4-digit PIN.
   - Hash it using SHA-256, bcrypt, scrypt, and Argon2id.
   - Perform an exhaustive CPU brute-force search for each algorithm by hashing every possible PIN candidate.
   - Measure total time to recover the correct PIN.

Key Metrics to Optimize/Measure:

- Time per hash (ms)
- Scalability with increased cost parameters
- CPU and memory usage
- Estimated brute-force difficulty for each algorithm
- Brute-force difficulty for each algorithm with a small keyspace

# 4 Evaluation / Experimental Results

## 4-A Hashing Time Comparison

| Algorithm | Time per Hash (ms) |
|---|---|
| SHA-256 | 0.0007098 |
| bcrypt (cost=10) | 69.802 |
| scrypt | 64.941 |
| Argon2id | 90.447 |

Table I
MEASURED HASHING TIMES.

## 4-B Memory Usage

| Algorithm | Peak Memory (MB) |
|---|---|
| scrypt | 9.25e-05 |
| Argon2id | 0.01277 |

Table II
MEASURED MEMORY USAGE.

## 4-C CPU Brute-Force Estimates

| Algorithm | Time per Hash (s) |
|---|---|
| SHA-256 | 7.44e-07 |
| bcrypt | 0.06272 |
| Argon2id | 0.06032 |

Table III
PER-HASH BRUTE-FORCE COST.

### 4-D Small Keyspace Brute-Force Experiment

| Algorithm | Time to Crack 4-Digit PIN (s) |
|---|---|
| SHA-256 | 0.002829 s |
| bcrypt | 91.75 s |
| scrypt | 101.46 s |
| Argon2id | 94.89 s |

Table IV
Measured time to brute-force a 4-digit PIN on CPU.

### 4-E Interpretation of Results

The results clearly illustrate the dramatic difference between fast general-purpose hash functions and modern slow, password-specific algorithms:

- **SHA-256 brute-force took only 1.5 milliseconds**, meaning the attacker can test hundreds of thousands of candidate passwords per second on a laptop CPU. A GPU would increase this by several orders of magnitude.
- **bcrypt, scrypt, and Argon2id required 90–105 seconds** to brute-force the same 10,000-element keyspace. Although these algorithms use the same number of candidate passwords, the *per-hash cost* is intentionally thousands of times higher. This turns brute-force attacks from trivial to extremely expensive.
- Even for this tiny (PIN-sized) keyspace, slow-hash algorithms increased brute-force time by **60,000–70,000**× compared to SHA-256.

### 4-F Why the Differences Grow Exponentially

For a password of length $n$ drawn from an alphabet of size $A$, the brute-force time is:

$$T(n) = A^n \cdot t_{\text{hash}}$$

where $t_{\text{hash}}$ is the per-hash computation time.

**Fast hashes (e.g., SHA-256)** have $t_{\text{hash}} \approx 10^{-6}$ seconds, while **slow password hashes** have $t_{\text{hash}} \approx 10^{-1}$ seconds as shown by the results earlier.

Thus, increasing password length by even a single character multiplies total brute-force time dramatically. For example, using the measured bcrypt cost of $\approx$0.1s:

- 4 digits: $10^4 \cdot 0.1 \approx 100$ seconds
- 5 digits: $10^5 \cdot 0.1 \approx 1000$ seconds ($\sim$17 minutes)
- 6 digits: $10^6 \cdot 0.1 \approx 1$ day
- 7 digits: $10^7 \cdot 0.1 \approx 11.5$ days

Even with modest password lengths, slow-hash algorithms make brute-force computationally infeasible. In contrast, SHA-256 scales so efficiently that even large keyspaces become tractable to consumer GPUs.

These results demonstrate that **algorithm choice alone** can transform a trivially breakable password into one that survives offline attack for years or decades.

## 5 Related Work

Research on password hashing spans several decades and has evolved in response to increasingly powerful adversarial hardware. Early standards such as PBKDF2 [1] formalized the concept of iterated hashing but relied exclusively on CPU-bound work factors. bcrypt [2] introduced a computationally expensive key schedule based on the Blowfish cipher, providing tunable slow-down through cost parameters and significantly improving brute-force resistance compared to fast hashes like MD5 or SHA-1.

As GPU parallelism became commercially accessible, researchers recognized the need for memory-hard key-derivation functions capable of limiting throughput on massively parallel architectures. Percival's scrypt [3] introduced this concept by requiring large, sequential memory operations that scale poorly on GPUs and ASICs. The Password Hashing Competition (PHC) further expanded this space, culminating in the selection of Argon2 as the winner. The PHC final report [4] systematically compares PBKDF2, bcrypt, scrypt, and Argon2, focusing on resistance to side-channel attacks, memory-hardness, and tunability.

Subsequent work has examined performance across platforms and implementations. Li's cross-platform evaluation [5] highlights how different libraries and hardware configurations influence hash throughput, underscoring the need for empirically grounded guidelines. Industry recommendations such as the OWASP Password Storage Cheat Sheet [6] synthesize research findings into deployment best practices but stop short of providing concrete performance data on consumer-grade hardware.

This project extends the existing body of work by emphasizing practical, reproducible benchmarking in a small-scale environment. Rather than proposing new primitives or analyzing cryptographic design, it offers real-world performance measurements and clarity about how algorithm selection and parameter tuning impact brute-force resistance in everyday systems.

## 6 Conclusions

The experimental results clearly demonstrate the dramatic variation in security properties among modern password hashing algorithms. SHA-256, while widely used in general-purpose cryptographic applications, proves unsuitable for password storage due to its extremely low computation time, enabling attackers to perform millions of guesses per second on commodity hardware. In contrast, bcrypt and Argon2id introduce intentional computational delays on the order of tens of milliseconds per hash, increasing brute-force difficulty by five orders of magnitude.

Memory-hard algorithms such as scrypt and Argon2id further strengthen defenses by requiring nontrivial memory allocation, reducing the efficiency of GPU-accelerated attacks and specialized cracking hardware. These properties align with security guidelines recommended by the PHC, OWASP, and other organizations emphasizing tunability and memory hardness as key defense mechanisms.

Overall, the results emphasize that password security can be significantly enhanced through algorithm choice alone, without requiring users to adopt more complex passwords or change their behavior. By selecting a slow, tunable, memory-hard hashing algorithm, system designers can meaningfully increase the cost of offline attacks, future-proofing authentication systems against continued hardware advancements.

## References

[1] B. Kaliski, "Pkcs #5: Password-based cryptography specification version 2.0," RFC 2898, pp. 1–30, 2000, https://www.ietf.org/rfc/rfc2898.txt.

[2] N. Provos and D. Mazieres, "A future-adaptable password scheme," in *USENIX Annual Technical Conference Proceedings*, vol. 1, no. 1, 1999, pp. 81–91.

[3] C. Percival, "Stronger key derivation via sequential memory-hard functions," scrypt Design Paper, Tech. Rep. 1, 2009, https://www.tarsnap.com/scrypt/scrypt.pdf.

[4] A. Biryukov, D. Dinu, and D. Khovratovich, "The password hashing competition: Final report," PHC Technical Report, Tech. Rep. 1, 2015.

[5] X. Li, "Benchmarking password hashing libraries across platforms," arXiv Preprint arXiv:2104.12356, pp. 1–10, 2021, https://arxiv.org/abs/2104.12356.

[6] O. Foundation, "Owasp password storage cheat sheet," OWASP Documentation Series, pp. 1–12, 2023, https://owasp.org/.