

# Evaluating Argon2 Adoption and Effectiveness in Real-World Software

Pascal Tippe<sup>1\*</sup> and Michael P. Berner<sup>1</sup>

FernUniversität in Hagen  
Hagen, Germany

**Abstract.** Modern password hashing remains a critical defense against credential cracking, yet the transition from theoretically secure algorithms to robust real-world implementations remains fraught with challenges. This paper presents a dual analysis of Argon2, the Password Hashing Competition winner, combining attack simulations quantifying how parameter configurations impact guessing costs under realistic budgets, with the first large-scale empirical study of Argon2 adoption across public GitHub software repositories. Our economic model, validated against cryptocurrency mining benchmarks, demonstrates that OWASP’s recommended 46 MiB configuration reduces compromise rates by 42.5% compared to SHA-256 at \$1/account attack budgets for strong user passwords. However, memory-hardness exhibits diminishing returns as increasing allocations to RFC 9106’s 2048 MiB provides just 23.3% (\$1) and 17.7% (\$20) additional protection despite 44.5× greater memory demands. Crucially, both configurations fail to mitigate risks from weak passwords, with 96.9-99.8% compromise rates for RockYou-like credentials regardless of algorithm choice. Our repository analysis shows accelerating Argon2 adoption, yet weak configuration practices: 46.6% of deployments use weaker-than-OWASP parameters. Surprisingly, sensitive applications (password managers, encryption tools) show no stronger configurations than general software. Our findings highlight that a secure algorithm alone cannot ensure security, effective parameter guidance and developer education remain essential for realizing Argon2’s theoretical advantages.

**Keywords:** password hashing · Argon2 · cryptographic adoption

## 1 Introduction

As reliance on digital systems continues to grow, ensuring secure user authentication has become a critical challenge in cybersecurity. Password hashing functions play a pivotal role in protecting credentials, yet legacy algorithms like SHA-256 exhibit persistent vulnerabilities when deployed in authentication systems. Despite their theoretical security properties, these algorithms are increasingly ineffective against modern attacks that leverage GPU and ASIC-based

---

\* Corresponding author: [firstname.lastname@fernuni-hagen.de](mailto:firstname.lastname@fernuni-hagen.de)

hardware to test billions of password candidates per second. This gap between theoretical robustness and practical resilience underscores the need for more advanced cryptographic solutions. Argon2, the winner of the 2015 Password Hashing Competition, represents a significant advancement in password hashing design by introducing memory-hardness that increases computational costs for attackers using specialized cracking hardware [2]. By requiring substantial RAM allocation during hash computation (configurable via parameters), Argon2 creates asymmetric costs that favor defenders over attackers. However, its effectiveness depends heavily on parameter selection, with configurations such as the OWASP-recommended 46 MiB memory differing significantly from the RFC 9106 proposal of 2 GiB memory. This disparity highlights the importance of standardized, context-aware parameterization frameworks that balance security and performance. While Argon2’s theoretical advantages are well-documented in academic literature, its adoption in real-world software remains inconsistent. Our preliminary repository scans suggest a persistent reliance on older algorithms like SHA-256 and PBKDF2, particularly in legacy systems where backward compatibility concerns often outweigh security considerations. Even when Argon2 is implemented, suboptimal parameter configurations are common, reflecting systemic barriers to cryptographic modernization. This study addresses four key questions: (1) How do specific Argon2 parameter configurations compare to SHA-256 in resisting GPU/ASIC-accelerated attacks under realistic password strength assumptions? (2) What is the current state of Argon2 adoption among software projects? (3) How have Argon2 parameters evolved over time to adapt to increasing computational capabilities of potential attackers? (4) Do applications with heightened security requirements, such as password managers or file encryption software, tend to implement stronger parameter configurations?

The remainder of this paper is organized as follows: Section 2 reviews related work on password hashing functions and guessing attacks. Section 3 describes our methodology, while Section 4 details our attack simulations. Section 5 presents an analysis of offline password cracking results, followed by Section 6’s exploration of real-world data collected from software repositories. Finally, Section 7 analyses the gathered data and Section 8 discusses findings before concluding in Section 9.

## 2 Background and Related Work

Online platforms continuously suffer from breaches exposing user passwords en masse. To mitigate this risk, traditional password storage relies on cryptographic hashing to prevent credential exposure. Unlike encryption, deterministic hash functions like SHA-256 produce fixed-length digests that cannot be feasibly reversed. However, attackers can enumerate password candidates until they find a matching hash value. This attack vector fundamentally depends on two factors: the computational efficiency of the hash function and the statistical distribution of password *guessability* across user populations. To prevent attackers from using precomputed rainbow tables and attacking multiple breached cre-

dentials simultaneously [14], defenders add a randomly generated string (called salt) to passwords, forcing attackers to target each individual salted password hash. While salting prevents batch attacks, it does not address the fundamental vulnerability of fast hash algorithms to brute-force and dictionary attacks. Instead of exhaustively iterating over all possible character combinations, attackers exploit users’ tendency to follow predictable patterns during password creation: combining words, replacing characters with numbers, or appending special symbols. Deprecated password metrics like Shannon entropy fail to capture these human-chosen password patterns, leading researchers to develop more accurate estimation techniques. The zxcvbn algorithm’s [19] pattern-aware entropy and models like Markov chains [18] or context-free grammars [18] better reflect real-world password weaknesses by analyzing dictionary matches, spatial keyboard patterns, and breach recurrence patterns. Bonneau [6] formalized the concept of guessability, establishing a direct connection to practical password strength measurement by quantifying the average number of guesses required to breach a target percentage of user accounts. This approach better captures systematic risks since Dell’Amico et al. [8] determined an attacker would require 149,053,078 attempts on average to crack over half the passwords in three real-world datasets (approximately  $2^{27.14}$ ). Bonneau [6] calculated median attack costs ranging between  $2^{19.8}$  and  $2^{21.6}$  attempts across multiple leaked datasets including Rock-You to crack 50% of the passwords. Florencio and Herley [9] found an average bit-strength of approximately 40.54 bits using naive calculations across half a million users over three months.

Memory-hard functions represent a paradigm shift in password hashing by imposing substantial RAM requirements. The interplay between memory-hardness and guessability becomes apparent in Blocki et al.’s framework [5], where attackers maximize compromised accounts within fixed budgets. The objective is not to linearly increase costs for both defenders and attackers by iterating hash functions, but to create asymmetric costs that disproportionately disadvantage attackers using specialized hardware such as ASICs. Argon2, the Password Hashing Competition winner in 2015, implements this approach through three tunable parameters: memory cost ( $m$ ), iterations ( $t$ ), and parallelism ( $p$ ). Argon2 comes in three variants: Argon2d (fast but vulnerable to side-channels), Argon2i (side-channel resistant but slower), and Argon2id (a hybrid approach). This study focuses exclusively on Argon2id, which RFC 9106 recommends as the default for password hashing. Its design prioritizes time-memory tradeoff resistance, forcing attackers to spend either prohibitive time or memory resources, directly impacting guessability economics. Blocki et al. [5] modeled this increased strength against guessing attacks as an optimization problem, using Bitcoin mining hardware and blockchain hashrate as proxies for determining attacker strength, finding that memory-hard hash functions substantially increase guessing costs. Argon2’s security relies on selecting appropriate values for its parameters, yet many developers in user studies struggle to implement even basic password hashing correctly [13]. Furthermore, while OWASP recommends 46 MiB of memory for general use cases [15], the RFC 9106 standard [2] advocates

for significantly higher values (2048 MiB). This  $44.5\times$  difference reflects a tension between practical deployment constraints and theoretical security requirements, potentially leaving developers uncertain about which configuration is suitable for their specific applications. Our work bridges this gap by analyzing real-world Argon2 implementation patterns across GitHub repositories, quantifying the security impact of different parameter configurations through attack simulations, and identifying systemic mismatches between academic parameter recommendations and developer implementation practices. This research extends prior work by providing concrete evidence of how theoretical security advantages translate, or fail to translate, into practical security improvements in deployed software.

### 3 Methodology

This study employs a comprehensive methodology to evaluate the technical performance and real-world adoption of Argon2 as a password hashing algorithm. The analysis is divided into two interconnected components: a security analysis of Argon2 configurations compared to SHA-256 and an empirical investigation into the adoption trends of Argon2 across software repositories on GitHub. By combining cryptographic modeling, password strength estimation, attack simulation, and repository analysis, this methodology provides a holistic view of both theoretical efficacy and practical implementation.

#### 3.1 Security Analysis Framework

The security analysis focuses on Argon2’s resistance to offline password cracking under realistic attacker constraints. The threat model assumes an attacker with offline access to hashed credentials and computational resources comparable to large-scale cryptocurrency mining operations. Attackers are assumed to prioritize cost-efficiency, spending a fixed budget for cracking passwords. We leverage public password datasets for strength estimation. To model cryptographic costs, we analyze the economics of cryptocurrency mining as a proxy for attacker resources. Bitcoin’s SHA-256 implementation serves as the baseline for traditional hashing costs, while Monero’s RandomX, a memory-hard proof-of-work algorithm based on Argon2d, provides insights into memory-dependent computation costs. These models are validated using energy consumption benchmarks from consumer-grade CPUs, ensuring real-world applicability. Password strength estimation is conducted using zxcvbn’s pattern-aware entropy metric. The RockYou 2009 dataset, leaked cleartext passwords from an online platform, is used as the baseline for password distributions, filtered to include only passwords meeting modern length requirements ( $\geq 8$  characters). For modeling enhanced password policies, we generated a synthetic dataset by doubling the zxcvbn bit-strength values from the filtered RockYou data. This transformation simulates passwords with significantly higher guessing resistance (e.g., a 20-bit password becomes a 40-bit password) while preserving the overall distribution characteristics. Attack simulations evaluate the effectiveness of different hashing configurations under

three budget scenarios for attackers: \$0.1, \$1, and \$20 per targeted account. Analyzed configurations include SHA-256 as a baseline and Argon2 implementations with both RFC 9106 recommended parameters (2048 MiB memory) and OWASP-suggested hardened parameters (46 MiB memory).

### 3.2 Data Collection and Repository Analysis

To assess Argon2’s adoption in real-world software projects, we systematically collect data from public repositories on GitHub using its REST API. GitHub was chosen due to its prominence in open-source development and its extensive repository metadata, which includes indicators such as stars that we use as a proxy for repository quality. While acknowledging that user motivations for starring repositories vary, prior research suggests that stars are more reliable indicators of relevance than other metrics like number of forks [7]. The analysis employs two complementary search methods: repository metadata search and code search. Repository searches query titles, descriptions, and topics for keywords related to password hashing algorithms (*Argon2*, *bcrypt*, *scrypt*, *yescrypt* and *PBKDF2*). The selection was driven by their prominence as widely recognized password hashing algorithms, providing a comparative baseline to evaluate Argon2’s adoption and security properties against established standards with distinct characteristics in memory-hardness and performance. Since GitHub limits search results to 1,000 entries per query, searches are segmented by repository creation date to capture a comprehensive dataset. Code searches identify instances of password hashing algorithm implementations within source code files. To address GitHub’s indexing limitations for code searches, results are segmented by programming language. Languages were selected based on their support for symbol extraction on GitHub and manual reviews of preliminary data<sup>1</sup>. To ensure accuracy in both search methods, filtering mechanisms are applied to exclude false positives (e.g., repositories unrelated to password hashing or those associated with cryptocurrencies). Automated exclusion based on keywords is supplemented by manual refinement to further reduce noise in the dataset.

### 3.3 Manual Review and Parameter Analysis

Repositories identified through searches undergo manual review to extract Argon2 parameter configurations and classify software types. This step ensures accuracy by accounting for variations in parameter naming conventions and library usage that automated tools might miss or misclassify. Additionally, this process verifies that Argon2id is used appropriately within repositories and not in contexts such as cryptocurrency mining. Repositories where parameter configurations cannot be assessed or that serve non-productive purposes (e.g., specifications or benchmarking tools) are excluded from further analysis. To focus

<sup>1</sup> Selected languages: Bash, C, C#, C++, CodeQL, Dart, Elixir, Erlang, Go, Haskell, Java, JavaScript, Kotlin, Lua, PHP, Python, R, Ruby, Rust, Scala, Starlak, Swift, TypeScript

on high-quality implementations, only repositories with significant number of stars are included in the final dataset. The extracted parameter configurations are analyzed to evaluate their alignment with recommended security practices. Repositories are categorized by software type (e.g., web applications, password managers), allowing comparisons between parameter strengths across different application domains. To analyze trends in Argon2 adoption over time and across software categories, statistical hypothesis testing is employed. Non-parametric tests such as chi-square goodness-of-fit and independence tests examine whether observed distributions deviate significantly from uniformity or exhibit associations between variables (e.g., repository type and parameter strength). A significance level of  $p=0.05$  is used throughout the analysis.

## 4 Attack Simulation Framework

The attack simulation framework evaluates Argon2’s economic resistance to offline password guessing by modeling adversarial cost structures under realistic resource constraints. Our analysis compares two recommended parameter configurations representing different security philosophies: the RFC 9106 recommendation (2048 MiB memory) prioritizing ASIC resistance through substantial memory demands, and OWASP’s pragmatic guidelines (46 MiB memory) balancing security with server resource limitations. These configurations create a  $44.5\times$  difference in memory allocation, enabling direct comparison in thwarting large-scale attacks.

### 4.1 Parameter Configurations

To explore the trade-offs between security and resource efficiency, we analyze two widely referenced Argon2 parameter configurations: the RFC 9106 recommendation and OWASP’s pragmatic guidelines. The RFC 9106 configuration prioritizes resistance to attacks by allocating 2048 MiB of memory per hash computation, thereby imposing significant memory demands on attackers and defenders. In contrast, OWASP’s configuration uses a reduced memory allocation of 46 MiB, reflecting a balance between security and server-side performance constraints. These configurations represent distinct security philosophies, with the former emphasizing robustness against specialized hardware and the latter accommodating practical deployment scenarios. Both are the first recommended configuration and use parameters  $t = 1$  and  $p = 1$  allowing a direct comparison. The  $44.5\times$  difference in memory allocation between these configurations provides a valuable basis for evaluating their relative effectiveness in thwarting large-scale attacks. In our simulations, attackers are modeled as having fixed budgets of \$0.10, \$1.00, and \$20.00 per targeted account. These budgets reflect varying levels of attacker investment, from low-cost opportunistic attacks to more resource-intensive campaigns targeting higher value accounts. The budgetary constraints are used to calculate the number of hash computations an attacker can afford under each parameter configuration, enabling direct comparisons of their economic resistance.

## 4.2 Cost per Hash Evaluation

The computational cost of Argon2 is central to its ability to resist offline attacks. To estimate this cost, we use cryptocurrency mining as a proxy for adversarial resource expenditures due to its well-documented economic metrics and operational similarities to password cracking. Specifically, we derive baseline costs for SHA-256 from Bitcoin mining data and extrapolate Argon2 costs using Monero’s RandomX algorithm, which incorporates Argon2d to create an initial cache and extends it with additional computations inside a virtual machine. For SHA-256, Bitcoin’s current network hashrate (701.72 EH/s) and block rewards as of 20 February 2025 [3] provide a per-hash cost estimate of approximately  $\$7.079 \times 10^{-19}$ . Argon2’s memory-hardness complicates direct benchmarking. However, RandomX [16] serves as a functional analog due to its use of approximately 2 GiB memory allocations and Argon2d usage as a base element. Adjusting for RandomX’s additional computational overhead (conservatively estimated at  $100\times$ ), we estimate Argon2’s base cost at  $\$2.729 \times 10^{-12}$  per hash for 2 GiB configurations with the network statistics on 20. February 2025 (4.54 GH/s, 32 blocks per hour and 232.31\$ per unit) [4]. This cost scales linearly with reduced memory allocations, allowing us to model the economic impact of different parameter settings.

To validate these estimates, we conducted energy consumption calculations using processor thermal design power (TDP) values and measured hashes per second on consumer-grade CPUs<sup>2</sup>. For example, using an energy price of \$0.05/kWh and considering only CPU power consumption, the cost per hash was calculated as  $\$4.17 \times 10^{-7}$ , which exceeds our baseline estimate derived from RandomX mining data. This discrepancy underscores the pessimistic nature of our baseline assumptions but also highlights the real-world feasibility of our cost model for very resourceful attackers.

## 4.3 Dataset Preparation

The datasets used in this study are critical for simulating realistic attack scenarios and evaluating password strength distributions under different hashing configurations. We employ two datasets: the RockYou dataset and a synthetic dataset  $D_{syn}$  derived from it. The RockYou dataset, leaked in 2009, contains over 32.6 million user passwords and is an important resource in password security research due to its size and real-world origins [6,5]. To ensure consistency and relevance to contemporary minimal security standards, we preprocessed this dataset. First, all passwords were normalized to UTF-8 encoding using Python scripts equipped with the `chardet` library to resolve character encoding inconsistencies; entries with unresolvable issues were removed (affecting 242 passwords). Next, passwords shorter than eight characters were excluded to align with modern minimum policy requirements, reducing the dataset by approximately 16.18

<sup>2</sup> Intel Core i3-7130U, AMD FX-6300, Intel Core i5-10300H, Intel Core i5-9400F, AMD Ryzen 5 2600X

million entries and yielding a curated subset of 16.42 million passwords. The filtered RockYou dataset exhibits a median password length of nine characters ( $M = 9.46, \sigma = 2.43$ ). Notable outliers include lengthy HTML fragments or URLs used as passwords that likely reflect user behavior anomalies rather than deliberate choices. These entries were retained to preserve the dataset’s authenticity despite their slight skewing effect on bit-strength calculations. Password entropy was estimated using zxcvbn’s pattern-aware algorithm and showed a mean (median) entropy of 21.9 (21.7) bits with a standard deviation of 9.6 bits and 26.8 for the third quartile and 15.6 for the first quartile. Recognizing that RockYou reflects pre-2010 user behavior patterns, we constructed  $D_{syn}$  by systematically doubling the bit-strength values of each password in the RockYou corpus while preserving its overall distribution shape. This approach accounts for improved password policies and heightened user awareness observed in recent years while maintaining compatibility with prior research methodologies. The synthetic dataset serves as an updated benchmark for evaluating Argon2’s performance in higher-security contexts. The doubled values align with results from Komanduri et al. [12] showing that complex password requirements yield on average 44.67 bit-strength passwords.

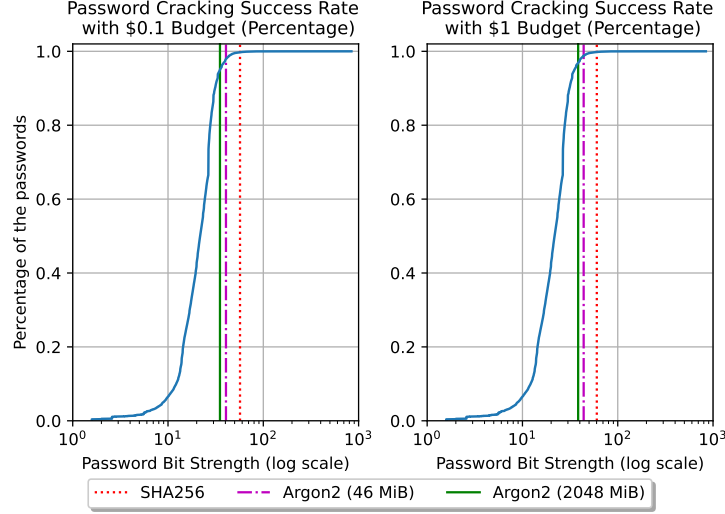
## 5 Attack Simulation Results

Our attack simulations show fundamental security tradeoffs between hashing algorithms, parameter configurations, and password strength distributions. Figures 1 and 2 illustrate the compromise rates for SHA-256, Argon2 using OWASP’s 46 MiB configuration, and Argon2 with RFC 9106’s 2048 MiB configuration across varied attacker budgets for both datasets

**RockYou Dataset:** As depicted in Figure 1, SHA-256’s susceptibility to low-cost attacks is strong, with near-total compromises (99.77%) at just \$0.10 per account, slightly worsens at a \$1.00 budget (99.83%). Conversely, Argon2 introduces modest resistance. The 46 MiB parameters reduce compromise rates to 98.81% at \$1 budgets, while the 2048 MiB configuration cuts down success to 96.89% under identical conditions, a notable 2.94% improvement compared to SHA-256 protecting more than 475,000 accounts. While Argon2 impacts the attackers success to a limited extent, the weak passwords are the decisive factor.

**Synthetic Dataset ( $D_{syn}$ ):** Figure 2 shows a shift when modeling stronger user passwords and policies. Here, SHA-256 achieves higher resistance due to improved password bit-strength. Due to the exponential effects of password bit-strength, the gap between hashing algorithms widens. At \$1.00 budgets, 88.31%, 50.74% and 38.92% of all passwords are compromised for SHA-256, Argon2 with 46 MiB and Argon2 with 2048 MiB. For the \$20 budget, this rate increases to 91.85%, 59.16% and 48.69%. This demonstrates that the stronger 2048 MiB configuration does provide stronger protection compared to 46 MiB with a 11.82 (10.48) percentage points lower compromise rate under the \$1 (\$20) budget. However, the largest difference is the change from SHA-256 to the lower Argon2 configuration indicating that it provides significantly more protection.





**Fig. 1.** Password cracking success rates for the RockYou dataset under \$0.1 and \$1 budgets for SHA-256 and Argon2 configurations (46 MiB and 2048 MiB).

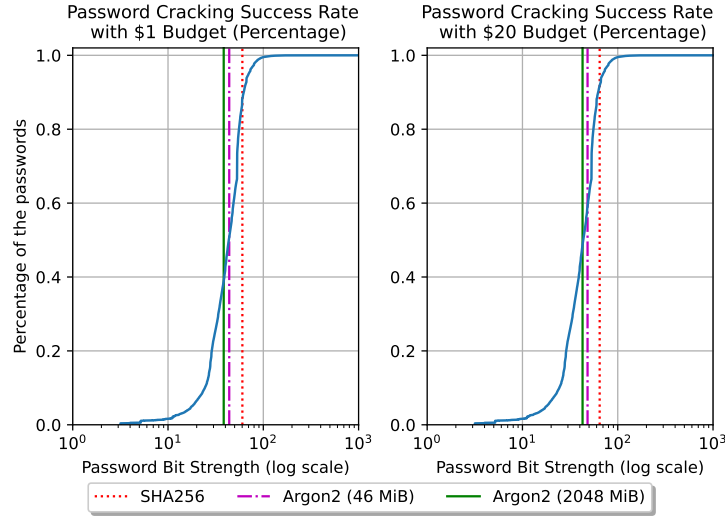
These results highlight the reliance on the strength of user passwords: For RockYou’s median 21.7-bit passwords, even the strongest Argon2 configuration couldn’t prevent attacker from cracking almost all passwords on small budgets. However, simulations using  $D_{syn}$ ’s median 43.4-bit passwords show that Argon2 additionally protects 43.16% of all accounts compared to SHA-256 with a \$20 budget. Our results show for attackers, that an increasing budget yields diminishing returns as the easy passwords are harvested fast while the increasing bitstrengths make attacks exponentially harder. At the same time, this also holds partially for defenders since strong parameter configurations do not help for weak user passwords. Using Argon2 instead of SHA-256 makes the biggest difference while the increasing server-side load does not proportionally protect more passwords but also shows diminishing returns for defenders.

## 6 Real-World Data Collection

Following our analysis of Argon2’s theoretical security properties and its resilience under simulated attacks, we shift the focus to research questions 2–4. We now analyze Argon2’s adoption trends through a systematic examination of GitHub repositories.

### 6.1 Repository Search

The dataset was constructed through systematic GitHub [10] repository searches for five password hashing algorithms (Argon2, bcrypt, scrypt, yescrypt, and



**Fig. 2.** Password cracking success rates for the synthetic dataset ( $D_{\text{syn}}$ ) under \$1 and \$20 budgets for SHA-256 and Argon2 configurations (46 MiB and 2048 MiB).

PBKDF2) across GitHub’s entire availability period (2008-2024). Table 1 shows the number of repositories for the filtering steps. For each algorithm, we executed temporal searches segmented by repository creation time, followed by a filtering phase. Effectively, we decided to include 31 repositories from one user for PBKDF2 and set the cutoff at 66 repositories and more per user. After manual review, we created a keyword list<sup>3</sup> related to common cryptocurrency themes to exclude irrelevant projects. After the filtering, we checked random samples from the results for each hashing algorithm and noticed that the 1,520 script repositories still contained many repositories unrelated to password hashing but included similar terms (i.e. *scripto*, *bash script*, *python script*), some of which we attribute to (intentional) misspellings. Therefore, we continued to filter the results and created an additional keyword list to ensure relevancy with words commonly used with hashing and key derivation functions: *password*, *hash*, *auth*, *kdf*, *key derivation*, *percival* (the author of *script*). 1,279 repositories of these 1,520 contained none of these additional keywords which we judged as too much. Therefore, we used an additional list containing similar names<sup>4</sup> that only excluded 439 repos. Among these 439 repos, all but three contained none of the relevancy words and were subsequently excluded. The three outliers were manually reviewed and one marked as relevant resulting in remaining 1,081 hits ( $1,520 - 439$ ). Out of these, only 238 contained at least one relevancy word,

<sup>3</sup> *miner*, *mining*, *proof-of-work*, *proof of work*, *currency*, *coin*, *wallet*, *bitzeny*, *doge*, *mint*, *blockchain*, *contract*

<sup>4</sup> *scripta*, *scripto*, *scripts*, *scripted*, *scripting*, *inscryption*, *scriptic*, *scripture*, *ipsa-script*

which led us to manually review the other results by analysing the URL name and project description. We excluded projects if they were surely not related to password hashing, marked them as *yes*, if we clearly connected them, and coded them as *possible*, if we could not conclude with high certainty. The latter was the case for 326 repos that did not contain any description. This resulted in 595 repositories, including the 219 possible hits, identified as scrypt password hashing repositories. Including the possible hits rather overestimates the prevalence of scrypt password hashing than underestimating it. Since Argon2 was included in five repositories with a creation date before 2015 (the year it won the Password Hashing Competition), we reviewed them manually. Two of them included Argon2 later while the others are abandoned or just include Argon2 references in non-productive parts.

**Table 1.** Repository collection and filtering statistics per algorithm

Algorithm	Initial Repos	Spam Removed (%)	Mining Filtered (%)	Final Count
Argon2	1,602	534 (33.33%)	36 (2.25%)	1,032
bcrypt	12,727	604 (4.75%)	58 (0.46%)	12,065
scrypt	2,396	528 (22.04%)	1,273 (51.13%)*	595
yescrypt	76	0 (0%)	36 (47.37%)	40
PBKDF2	1,006	0 (0%)	12 (1.19%)	994

\* Includes extended relevance checks for scrypt repositories (see Subsection 6.1).

## 6.2 Code Search

The code search was performed using the GitHub Search API for programming languages supporting symbol extraction and additional programming languages that we assessed as relevant after our manual review of repositories, as listed in Subsection 3.3. The primary search term for each query was the name of the password-hashing algorithm itself, refined with negative keywords to exclude cryptocurrency-related projects, which are outside the scope of this research. These negative keywords were the same as used in the repository search (see Subsection 6.1). Furthermore, we excluded files with the *.md* extension (to avoid *README* files and other documentation) and files located within directories containing *test* in their name to minimize irrelevant results. Due to the potential for a single repository to contain multiple instances of a given hashing function across different files, our search results often included duplicate entries for the same repository. To address this redundancy and estimate the number of unique repositories, we calculated a duplication quota based on the ratio of distinct repository IDs within the first 1,000 search results. This quota was then applied to the total number of search results to approximate the underlying number of unique repositories implementing each hashing algorithm.

Table 2 shows the repository search results. Initial searches without programming language differentiation yielded total code hits of 48,768 for Argon2, 519,168 for bcrypt, 36,592 for PBKDF2, 131,328 for scrypt, and 3,232 for yescrypt. Calculating repository redundancy required estimating a repository duplication quota based on unique repositories within the first 1,000 results (5.9% for Argon2, 1.2% for bcrypt, 13.6% for PBKDF2, 26.2% for scrypt, and 67.3% for yescrypt). This resulted in estimated total repositories of 45,891, 512,938, 31,615, 96,920, 1,057, respectively. The search separated by programming languages and hash function has 115 combinations. For 52 combinations the query results were below 1,000 and the repository number could be counted directly without using the duplication quota. The overall results are shown in Table 2. Since we could not conduct the additional filtering steps for scrypt that we did for the repository search, we used the filtering ratio from the repository search to estimate the filtered number of scrypt results. Table 3 shows the quotas for different result sizes and hashing functions.

**Table 2.** Total hits and estimated number of repos, separately for the different password hashing methods and searches.

	Argon2	bcrypt	PBKDF2	scrypt	yescrypt
<i>Code Search with Programming Language Differentiation:</i>					
Total Code Hits	41,464	226,768	75,521	88,211	905
Estimated Total Repos	33,170	213,012	64,645	64,416	531
				(29,116*)	
<i>Simple Code Search:</i>					
Total Code Hits	48,768	519,168	36,592	131,328	3,232
Estimated Total Repos	45,891	512,938	31,615	96,920	1,057
				(43,808*)	
<i>Repository Search Results:</i>					
Results from Repo Search	1,032	12,065	994	595	40

\* After applying the scrypt false positive removal rate of 54.8% as determined in Subsection 6.1 after only applying the initial cryptocurrency filter.

### 6.3 Parametrisation

For manual identification of Argon2 configurations, we decided to focus on high-quality repositories only. From the 1,068 Argon2 repositories found in the repository search, we focused on repositories with at least three star ratings, resulting in 253 remaining repositories. We further excluded repositories that were archived or not productive (e.g., described as homework assignments, demos or

**Table 3.** Repository redundancy quotas for password hashing methods from code search analysis with programming language differentiation. For large result sets ( $>3,500$  hits,  $n=30$ ) and medium-sized sets (1,000–3,500 hits,  $n=26$ ), quotas were estimated from the first 1,000 results. For small result sets ( $<1,000$  hits,  $n=52$ ), exact quotas were determined. Analysis covers 108 of 115 programming language and hash function combinations.

Repo Redundancy	Argon2	bcrypt	PBKDF2	scrypt	yescrypt	Total
Estimated ( $>3500$ , $n=30$ )	5.50% $n=3$	9.89% $n=12$	11.83% $n=7$	24.03% $n=8$	—	13.67% $n=30$
Estimated ( $\leq 3500$ , $n=26$ )	32.59% $n=9$	23.60% $n=4$	26.97% $n=7$	44.15% $n=6$	—	32.36% $n=26$
Exactly determined ( $n=52$ )	59.73% $n=11$	51.17% $n=7$	50.83% $n=9$	64.11% $n=9$	37.24% $n=16$	50.87% $n=52$
Total ( $n=108^*$ )	42.03% $n=23$	24.84% $n=23$	31.70% $n=23$	44.96% $n=23$	25.90% $n=16^*$	33.89% $n=108^*$

\* In 7 of the total 115 cases, there were no matches for yescrypt and therefore not considered here.

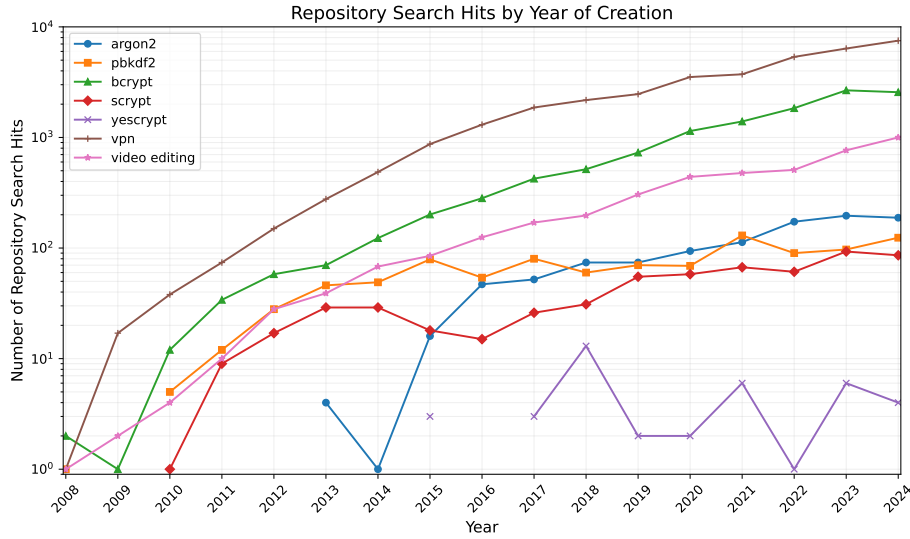
trials). We further divided the remaining 206 repositories into four equal sets based on their star count (3-4, 5-10, 11-30 and more than 30). We further excluded 21 repositories tied to cryptocurrency applications or password cracking. Afterwards, we classified each project in the following categories: components (libraries, wrapper, bindings), applications and sensitive applications (password manager and file encryption). Then we proceeded to manually extract the Argon2id configuration with a focus on iterations ( $t$ ) and memory ( $m$ ). In 24 repositories, we were unable to determine the parameters because they do not offer a (complete) default parametrisation, are specifications or benchmarking tools. If the software code used a library and didn't modify the parameter settings, we extracted the libraries default settings as parameters. In sum, parametrisation data was collected for 161 repositories.

## 7 Real-World Implementations Analysis

### 7.1 Adoption

Figure 3 shows the number of repos for the hash functions per creation year. To put this into the context of general repository developments, we decided to introduce two more search terms, with *VPN* being related to computer security and *video editing* unrelated to the field. The development of bcrypt aligns with that of these additional search terms. Argon2 also shows continuous growth at a lower rate since its inception in 2015. scrypt and PBKDF2 show notably less development which we assume is due to the introduction of Argon2. Argon2 also man-

aged to overtake the number of new repositories from 2018 onwards for scrypt, PBKDF2 and yescript. yescript, a competitor of Argon2 in the Password Hashing Competition, did not succeed in keeping up with Argon2’s adoption and has stagnating lower creation numbers. Focusing on the time between 2015 and 2024, the average number of repositories with a creation date in the respective years has a mean value of 102.7 for Argon2 ( $\sigma = 59.95$ ), 85.3 for PBKDF2 ( $\sigma = 25.47$ ) and 51 for scrypt ( $\sigma = 27.52$ ). We used the Kruskal-Wallis test to determine a statistically significant difference between the three groups ( $H(2) = 6.07, p = .048$ ) and then conducted Dunn tests to compare the hashing functions with each other. Argon2 and scrypt differ significantly ( $z = 2.13, p = .033$ ) and PBKDF2 and scrypt differ significantly ( $z = 2.13, p = .033$ ) while there is no statistically significant difference between Argon2 and PBKDF2 ( $z = 0, p = 1.0$ ). This indicates that Argon2 clearly overtook scrypt, but due to the larger variance in the number of Argon2 repositories created over the years no statistically significant overall difference is evident between Argon2 and PBKDF2.



**Fig. 3.** Repository search hits for various password hashing algorithms (2008-2024).

During the repository analysis, we found that 141 repositories offered multiple hashing algorithms. The majority (93) implements exactly two functions (of the five in the scope of this analysis), 33 offer three functions and 15 offer 4 functions. From the possible 26 combinations, only 13 appear in our dataset. yescript appears only in two of these 13 combination repos and is therefore excluded from the following significance test. A chi-square goodness of fit test shows that there is no clear evidence that the distribution differs from uniform distribution

( $\chi^2(3) = 0.25, p = .969$ ). Therefore, we did not find evidence that some of the analysed hashing functions are overrepresented in combination repositories.

The code search had more uncertainties since the number of repositories was estimated for search queries with more than 1,000 results based on the repo redundancy quota in the accessible results. Table 3 shows that the determined quotas clearly differ. Evidently, the estimated repository duplication quotas are closer to the exactly calculated ones when the number of code search results is smaller (below 3,500). This is plausible since the subset of 1,000 accessible code search results will be more representative if the selection pool is smaller (below 3,500 instead of more than 3,500). To ensure that the observed differences in repository duplication quotas are not systematically influenced by the interplay between hashing method and the size of the result set, we perform a chi-square test of independence, excluding yescrypt to avoid potential bias. The nonsignificant result ( $\chi^2(6) = 8.36, p = .213$ ) supports the validity of comparing methods, indicating that the identified trends likely reflect genuine differences rather than sampling artifacts. Overall, the results from the general code search and the one segmented by programming languages supports the results from the repository search as shown in Table 2. The repository search and simple code search results generally followed a consistent pattern. However, PBKDF2 was an exception to this trend, as the programming-segmented code search yielded more PBKDF2 results than the simple code search. We could not identify a plausible explanation for this anomaly. The other hash functions led to a smaller number of estimated repos in the segmented code search since the redundancy quota was more accurately calculated. The code search results support the repository search results for the adoption of Argon2, confirming its growing prominence among password hashing algorithms.

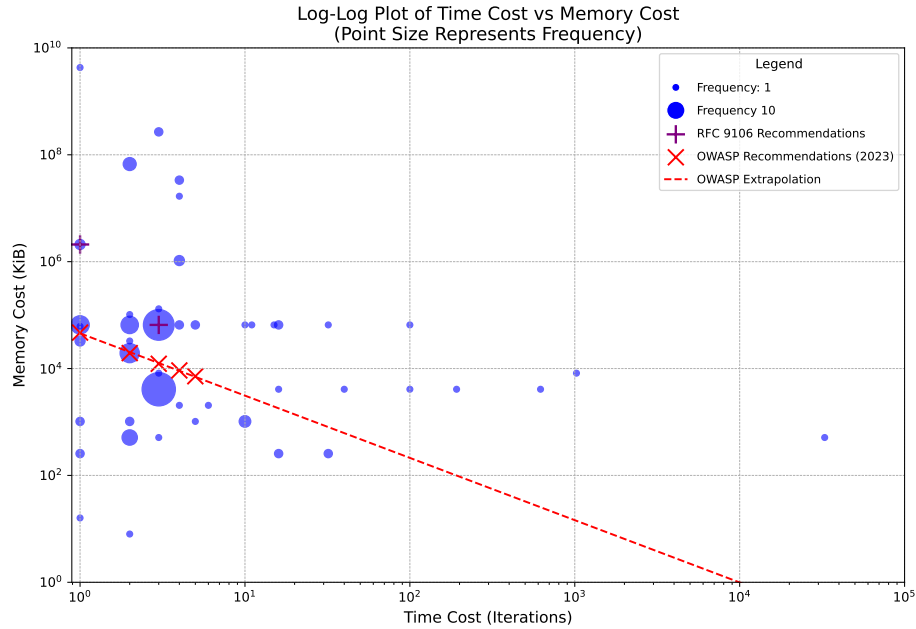
## 7.2 Parametrization over Time

During the data collection phase, we used the star count as a quality proxy to focus on high quality repositories. We set the star count for at least 3 stars to conduct the initial filtering. To test our assumption, we divided this initial set of 253 repositories into four equal size classes: 3-4 stars, 5-10 stars, 11-30 stars and more than 30 stars. We created a contingency table with the star sets and the number of productive and non-productive repositories that we filtered in the following step. A chi-square independence test shows the distribution is not independent ( $\chi^2(6) = 12.53, p = .006$ ) and that higher star counts are associated with less non-productive repositories. This supports our assumption of the star count as a quality proxy. After extracting the parameter configurations successfully from the selected 161 repositories (see 6.3), we noticed that the configurations were clustered with the following most popular configurations:

- $t = 3, m = 4096$  KiB (33 times)
- $t = 3, m = 65536$  KiB (28 times),
- $t = 2, m = 19456$  KiB (11 times),
- $t = 1, m = 65536$  KiB (10 times),

–  $t = 2, m = 65536$  KiB (9 times)

We attribute this especially to the default values of used libraries. We compared the parameter strength by linearly extrapolating the OWASP Argon2 recommendations and classifying the extracted configurations as weaker or stronger. Figure 4 shows the OWASP extrapolation and the observed repository configurations. 75 repositories were weaker and 86 stronger than the OWASP recommendations. To see the development over the years, we created a contingency table shown in Table 4 and grouped the repository creation years 2013 - 2024 in three groups: before 2018, 2019-2021, 2022-2024. The grouping helped to reach the minimum number of entries for the chi-square test and takes the publication date of RFC 9106 and OWASP recommendations into account. A chi-square independence test confirms that these factors are not independent ( $\chi^2(2) = 8.42, p = .015$ ). This shows an increase of parametrization strength over time.



**Fig. 4.** Log-log plot of time cost ( $t$ ) versus memory cost ( $m$ ). The size of each blue dot represents the frequency of data points with a specific combination of  $t$  and  $m$ . The red crosses represent OWASP recommendations, while the purple crosses represent RFC 9106 recommendations. The dashed red line extrapolates OWASP recommendations using a linear regression in log-log space.



**Table 4.** Repository age and categorical strength of parameterization.

Parameterization Strength	Repository Age			Total
	$\leq 2018$	2019–2021	2022–2024	
Weaker	38 (60.3%)	23 (41.1%)	14 (33.3%)	75 (46.6%)
Stronger	25 (39.7%)	33 (58.9%)	28 (66.7%)	86 (53.4%)
Total	63 (100.0%)	56 (100.0%)	42 (100.0%)	161 (100.0%)

### 7.3 Parametrization by Program Type

Table 5 shows the parametrization strength for the different software types. Weaker configurations are more present for components while the configurations are stronger for applications. A chi-square test confirms this hypothesis by showing a statistically significant effect of the software type variable ( $\chi^2(2) = 7.38, p = .007$ ). When the applications are separated into sensitive (file encryption, password managers) and normal password hashing applications, there is no statistically significant effect ( $\chi^2(1) = 0.002, p = .967$ ) between them. So, the hypothesis that sensitive software programs use stronger parametrizations could not be confirmed. To exclude effects from the star count on the software type, we tested with a contingency table if there is a statistically significant effect of the repository star count (divided in four equal sized groups) on the software type (application vs. component) and determined no significant effect ( $\chi^2(3) = 3.41, p = .332$ ).

**Table 5.** Cross table: Software type x categorical strength of parameterization.

Parameterizations	Sensitive Application	Application	Component	Total
Weaker	4 (26.7%)	6 (27.3%)	65 (52.4%)	75 (46.6%)
Stronger	11 (73.3%)	16 (72.7%)	59 (47.6%)	86 (53.4%)
Total	15 (100.0%)	22 (100.0%)	124 (100.0%)	161 (100.0%)

Additionally, we analysed how the parametrization is affected by the star count. Surprisingly, stronger configurations dominate the repositories with lower star counts (3-10), while repositories with more stars (more popular repositories) exhibit weaker configurations (see Table 6). A chi-square test confirmed this effect as statistically significant ( $\chi^2(3) = 8.71, p = .033$ ). While this seems counterintuitive, we noticed that the star count (popularity) is tied to the age as well. This hypothesis would fit to the finding from Subsection 7.2 that older repositories exhibit weaker configurations than newer ones. Therefore, we tested the influence of repository age on star count and confirmed with the chi-square

independence test that older repositories have a higher star count ( $\chi^2(6) = 32.53, p < .001$ ). This is also intuitive since older projects have more time to gain popularity and accrue stars. Finally, we tested the influence of repository age on software type and found that there is a statistically significant effect such that the number of components is higher for older repositories while the younger repositories include more applications ( $\chi^2(2) = 20.25, p < .001$ ). This weakens the finding that components exhibit weaker configurations since the repository age is a mitigating factor here as well, suggesting that the observed difference may be more strongly associated with when repositories were created rather than their functional purpose as components versus applications.

**Table 6.** Star categories and categorical strength of parameterization.

Parameterization Strength	Repository Star Count				Total
	3–4 Stars	5–10 Stars	11–30 Stars	>30 Stars	
Weaker	13 (34.2%)	12 (35.3%)	28 (62.2%)	22 (50.0%)	75 (46.6%)
Stronger	25 (65.8%)	22 (64.7%)	17 (37.8%)	22 (50.0%)	86 (53.4%)
Total	38 (100.0%)	34 (100.0%)	45 (100.0%)	44 (100.0%)	161 (100.0%)

## 8 Discussion

### 8.1 Main Findings

**Argon2’s security advantages over SHA-256** Our attack simulations validate Argon2’s fundamental security advantage over SHA-256, even with modest parameters. For synthetic datasets modeling modern password policies ( $D_{\text{syn}}$ ), Argon2 (46 MiB) reduces compromise rates by 37.57% versus SHA-256 at \$1 account budgets. However, memory allocation effectiveness exhibits diminishing returns: while 2048 MiB configurations provided 23.3% (at \$1) and 17.7% (at \$20) additional protection over 46 MiB, their 44.5× greater memory demands impose impractical scaling costs for many systems. Crucially, no configuration sufficiently mitigates risks for weak passwords, emphasizing that algorithm selection cannot compensate for poor user credential practices.

**Growing adoption trend** Argon2 adoption has steadily increased since its introduction in 2015, surpassing competing algorithms like scrypt and PBKDF2 in the number of new GitHub repositories created annually starting in 2018. However, its adoption lags behind bcrypt, which remains the most widely implemented password hashing algorithm likely due its older age and familiarity of developers. Our analysis of over 161 manually reviewed repositories revealed that Argon2 is present in a diverse range of software projects, with both applications (38 repositories) and components (124 repositories) incorporating it into

their cryptographic workflows. Interestingly, the frequency of Argon2’s coexistence with other algorithms (e.g., bcrypt, PBKDF2, scrypt) in multi-algorithm implementations underscores its growing acceptance as part of a broader cryptographic toolbox. However, the absence of statistically significant overrepresentation of any specific combinations suggests that Argon2 adoption is not yet widespread enough to dominate as a preferred choice.

**Parameter evolution** Our analysis of Argon2 parameter configurations in real-world implementations reveals a shift towards stronger configurations over time. Before 2018, 60.3% of repositories adopted weaker-than-OWASP-recommended settings, but this proportion decreased to 33.3% in repositories created after 2022. This trend aligns with the publication of the RFC 9106 standard in 2021 and evolving OWASP guidelines, which have likely increased awareness of the importance of using secure configurations. The observed clustering of common configurations suggests a heavy reliance on default library settings rather than deliberate customization by developers.

**Context-dependending Configuration** Contrary to our expectations, sensitive applications such as password managers and file encryption software did not consistently implement stronger Argon2 parameter configurations compared to general-purpose applications. While 73.3% of sensitive applications used stronger settings than OWASP recommendations, this proportion was similar to general application repositories (72.7%). This finding highlights unclear practices for parameter selection, even among software with higher security stakes. Interestingly, components (e.g., libraries and cryptographic bindings) exhibited weaker parameterization (52.4% below OWASP standards), possibly due to the need to balance performance and usability across diverse deployments. It is worth noting that this correlates with older repositories implementing weaker configurations on average and younger repositories increasingly being applications. Similarly, repositories with a higher star count tend to implement weaker configurations which is likely also connected to repository age.

## 8.2 Practical Recommendations

Our comprehensive analysis reveals that while Argon2 offers theoretical security advantages, these are not fully realized in practice. To bridge this gap, we offer actionable recommendations. Argon2’s benefits are amplified when combined with robust user passwords. Studies indicate that password meters and password policies significantly increase password strength, aiding users in creating more secure credentials [12,17]. Developers should implement strength estimation tools, enforce password policies, and integrate with techniques like password blacklisting to ensure users generate stronger passwords. Facilitating secure implementation requires simplifying the developer experience. Integrating comprehensive documentation and automated parameter selection tools into cryptographic libraries can increase adoption [1]. Providing sane default configurations (OWASP and RFC 9106 recommendations) directly within libraries streamlines the process, reducing configuration errors. Additionally, automated checks in vulnerability

scanners or compilers could flag weak settings, ensuring password-hash hardening remains a priority. Argon2 parameters may create a configuration challenge for some developers. Implementing adaptive benchmarking tools, as noted in some repositories, automates parameter selection tailored to local environments. To protect servers from potential denial-of-service attacks resulting from intensive hash computation, partial client-side hashing can be considered. Clients precompute their passwords with Argon2 before transmitting the resulting hash to the server, which then performs fast hashing functions [11]. While OWASP suggests a conservative 46 MiB of memory on the client side, current devices often possess capabilities to accommodate higher allocations, allowing for more robust defense multipliers without compromising usability.

### 8.3 Limitation and Future Work

The economic models for hash computation costs use cryptocurrency mining practices as a proxy for large-scale computational attacks. While this provides a validated cost structure, it does not fully capture the nuances of different attackers that can hardly compete with large centralized mining pools. Using cryptocurrency mining as a proxy for attacker costs introduces uncertainty into our budget-scenario analyses, which could systematically underestimate the true protection levels afforded by Argon2 in production environments. Also, the used budget may vary significantly: While \$1 might be appropriate for a low-relevance credentials, accounts with cryptocurrency assets carry significantly more wealth that in turn justifies substantially increased attack budgets. The password datasets we employed (RockYou and  $D_{syn}$ ) serve as benchmarks for password strength but carry inherent limitations. RockYou is decades old and its password distribution might not accurately reflect current practices. While our synthetic dataset addresses this by doubling bit-strength values, it remains an approximation. Real-world password behavior, influenced by contemporary policies, user awareness, and cultural factors, might deviate significantly from our modeled datasets.

Our real-world study focuses on open-source projects hosted on GitHub. This selection bias may skew our results towards projects developed in a particular community culture, potentially missing trends in proprietary software or repositories on alternative platforms. The reliance on GitHub’s prominence means that our findings might underestimate the adoption and implementation trends of Argon2 in closed-source or enterprise environments, where different regulations and development practices could influence cryptographic choices. Moreover, developer motivations for starring repositories are heterogeneous (e.g., bookmarking, acknowledgment of quality, personal interest), adding variability to our quality proxy metric. Additionally, the manual extraction process restricted our parameter analysis to 161 high-star repositories, potentially missing patterns present in a broader sample. While we believe this selection provides a representative set of quality implementations, broader sampling might reveal different distributions in parameter configurations. Addressing these limitations requires further research, potentially incorporating a broader range of software implementations,

cost models derived from real-world attackers and conducting user studies with programmers to explore Argon2 transition barriers.

## 9 Conclusion

This research evaluated Argon2’s cryptographic effectiveness for password hashing and relevant implementation trends in real-world software environments. Through attack simulations we demonstrated that Argon2 provides substantial security advantages over SHA-256, with the 2048 MiB RFC 9106 configuration reducing compromise rates by 46.99% compared to SHA-256 at \$20 attack budgets in datasets modeling modern password policies. The OWASP-recommended configuration offers less protection for robust user passwords compared to RFC 9106 values. However, even the strongest Argon2 configurations cannot compensate for weak user passwords, demonstrating the need for robust user passwords. Beyond technical performance, the analysis of GitHub repositories revealed that real-world adoption of Argon2 has grown steadily, surpassing other modern algorithms like scrypt and PBKDF2 in new implementations since 2018. Despite this, bcrypt remains the dominant choice, and parameter configurations in many repositories still fall short of OWASP and RFC 9106 recommendations. Moreover, parameter strength has improved over time, aligning with updated standards and adoption, but a significant number of implementations have weaker-than-recommended configurations. Sensitive applications do not consistently implement stronger Argon2 configurations compared to others, challenging assumptions about the correlation between software security demands and cryptographic diligence. These results suggest that while Argon2 holds significant cryptographic advantages for password hashing, its real-world security effectiveness depends on proper parameterization and user practices.

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Acar, Y., Backes, M., Fahl, S., Garfinkel, S.L., Kim, D., Mazurek, M.L., Stransky, C.: Comparing the Usability of Cryptographic APIs. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 154–171. IEEE Computer Society (2017)
2. Biryukov, A., Dinu, D., Khovratovich, D., Josefsson, S.: Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications. RFC 9106, IRTF (2021), <https://www.rfc-editor.org/info/rfc9106>
3. Bitbo: Bitcoin Market and Mining Statistics. <https://bitbo.io> (2025), accessed: 2025-02-20
4. BitInfoCharts: Monero (xmr) statistics. <https://www.bitinfocharts.com/monero/> (2025), accessed: 2025-02-20
5. Blocki, J., Harsha, B., Zhou, S.: On the Economics of Offline Password Cracking. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 853–871 (2018)

6. Bonneau, J.: The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In: 2012 IEEE Symposium on Security and Privacy. pp. 538–552 (2012)
7. Borges, H., Tulio Valente, M.: What’s in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *Journal of Systems and Software* **146**, 112–129 (2018)
8. Dell’Amico, M., Michiardi, P., Roudier, Y.: Password Strength: An Empirical Analysis. In: Proceedings of the 29th Conference on Information Communications. pp. 983–991. INFOCOM’10, IEEE Press (2010)
9. Florencio, D., Herley, C.: A Large-Scale Study of Web Password Habits. In: Proceedings of the 16th International Conference on World Wide Web. pp. 657–666. WWW ’07, Association for Computing Machinery, New York, NY, USA (2007)
10. GitHub, Inc.: Github. <https://github.com> (2025), accessed: 2025-01-28
11. Harsha, B., Blocki, J.: Just In Time Hashing. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 368–383 (2018)
12. Komanduri, S., Shay, R., Kelley, P.G., Mazurek, M.L., Bauer, L., Christin, N., Cranor, L.F., Egelman, S.: Of Passwords and People: Measuring the Effect of Password-Composition Policies. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2595–2604. CHI ’11, Association for Computing Machinery, New York, NY, USA (2011)
13. Naiakshina, A., Danilova, A., Gerlitz, E., von Zezschwitz, E., Smith, M.: "If you want, I can store the encrypted password": A Password-Storage Field Study with Freelance Developers. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. pp. 1–12. CHI ’19, Association for Computing Machinery, New York, NY, USA (2019)
14. Oechslin, P.: Making a Faster Cryptanalytic Time-Memory Trade-Off. In: Boneh, D. (ed.) *Advances in Cryptology - CRYPTO 2003*. pp. 617–630. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
15. OWASP: Password Storage Cheat Sheet. [https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password\\_Storage\\_Cheat\\_Sheet.md](https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md) (2023), accessed: 2025-02-02
16. tevador: RandomX: Proof of work algorithm based on random code execution. <https://github.com/tevador/RandomX> (2023), accessed: 2025-01-10
17. Ur, B., Kelley, P.G., Komanduri, S., Lee, J., Maass, M., Mazurek, M.L., Passaro, T., Shay, R., Vidas, T., Bauer, L., Christin, N., Cranor, L.F.: How Does Your Password Measure Up? The Effect of Strength Meters on Password Creation. In: Proceedings of the 21st USENIX Conference on Security Symposium. p. 5. Security’12, USENIX Association, USA (2012)
18. Weir, M., Aggarwal, S., Medeiros, B.d., Glodek, B.: Password Cracking Using Probabilistic Context-Free Grammars. In: Proceedings of the 2009 30th IEEE Symposium on Security and Privacy. pp. 391–405. SP ’09, IEEE Computer Society, USA (2009)
19. Wheeler, D.L.: Zxcvbn: Low-Budget Password Strength Estimation. In: Proceedings of the 25th USENIX Conference on Security Symposium. pp. 157–173. SEC’16, USENIX Association, USA (2016)