INTERNATIONAL JOURNAL IN ENGINEERING SCIENCES

Journal Homepage: https://www.ijesjournal.com

OPEN ACCESS

Research Paper

Password Strength Analyser and Security Recommendation System with Breach Intelligence and Encrypted Vault

Abhinav Soman*, Krushnraj Rajput, Rohan Gavit, Tushar Chinche, Jitendra Patil

Department of computer applications, Sandip Foundation's Sandip Institute of Engineering & Management, Nashik-422013, India

ARTICLE INFO

Published: 15 Nov. 2026

Keywords:
Password Strength Estimation, Entropy Analysis, k-Anonymity, Breach Intelligence, AES-GCM, PBKDF2, Credential Security, Applied Cryptography.

DOI:
10.5281/zenodo.24251985

ABSTRACT

Most digital accounts are still protected by passwords, and a large share of security breaches can be traced back to passwords that were weak, reused, or already leaked. Typical password meters only check shallow composition rules, such as a minimum length or a mix of character types, and these are easy to satisfy without ending up with a truly strong password. In this paper we present a hybrid Password Strength Analyser and Security Recommendation System. It combines Shannon-entropy estimation, structural pattern detection (keyboard walks, sequences, repeated characters and date-like substrings) and dictionary/common-password matching to give a realistic strength score from 0 to 100. The system also includes a privacy-preserving Breach Intelligence module built on the k-anonymity protocol, so only a truncated hash prefix of a password is ever sent outside the machine, and an AES-256-GCM encrypted credential vault that uses PBKDF2-HMAC-SHA256 key derivation for safe local storage. It is written in Java with no external dependencies and can be used through a console application or a browser-based web application running on the JDK's built-in HTTP server. Tests on a representative set of passwords show that the hybrid scoring approach can tell apart passwords that follow the composition rules but are still weak from passwords that are genuinely high in entropy, and that it does better than simple rule-based checks.

INTRODUCTION

Passwords are still the most common way people log in to web applications, enterprise systems and personal accounts. Biometrics and multi-factor authentication are becoming more common, but passwords remain dominant because they are simple, cheap to implement and work everywhere [1]. Studies of large anonymized password collections have shown again and again that many user-chosen passwords come from a surprisingly

***Corresponding Author:** Abhinav Soman
Address: *Department of computer applications, Sandip Foundation's Sandip Institute of Engineering & Management Nashik-422013, India*
Email : abhivoner28@gmail.com

Relevant conflicts of interest/financial disclosures: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

small search space. That makes them easy targets for dictionary attacks, brute-force attacks and credential stuffing with passwords from earlier leaks [2].

Most real systems still judge password strength by composition rules: a minimum length, plus an uppercase letter, a digit and a symbol. These rules are easy to meet without gaining any real strength. A password like "Password1!" ticks every box, yet it is one of the first guesses an attacker would try. Cracking simulations based on guessing have shown that following a composition policy has little to do with how well a password actually resists cracking [3].

In this paper we propose a Password Strength Analyser and Security Recommendation System that fills three gaps left by conventional approaches. First, strength estimation is shallow, so we combine entropy analysis with structural pattern detection. Second, conventional checkers know nothing about breaches, so we add a privacy-preserving breach-intelligence check. Third, credential hygiene is poor, so we provide a secure encrypted vault where strengthened passwords can be kept and reused during a session without being written down insecurely.

II. LITERATURE SURVEY

Wheeler [1] introduced zxcvbn, a pattern-matching password estimator that models how attackers actually guess, including dictionary words, dates, keyboard patterns and l33t-speak substitutions. His work showed that pattern-aware estimation is much more accurate than composition-rule checks. Bonneau [2] analyzed an anonymized corpus of 70 million passwords and measured the guessing entropy of real-world password distributions, setting empirical baselines that later strength-estimation research has widely used.

Kelley et al. [3] ran several password-cracking algorithms against large password sets under different composition policies. They found that stricter rules do not reliably make passwords harder to crack, and that some policies push users toward predictable workarounds, such as adding "1!" to a familiar word. Ur et al. [4] built and evaluated a data-driven password meter that gives users real-time, detailed feedback instead of a single pass/fail result, and found that this kind of feedback measurably improved the strength of the passwords people chose.

The NIST Special Publication 800-63B guidelines [5] formally advise moving away from forced periodic password changes and complex composition rules. They favour length, screening against lists of known-breached passwords, and secure storage. Hunt [6] brought breach screening to scale with the "Pwned Passwords" service, which popularized the k-anonymity protocol for checking whether a password has been exposed without revealing the full password or hash to the remote service. Pearman et al. [7] studied how people adopt password managers and found that convenient, well-integrated tools make users much more likely to use unique, strong passwords rather than reuse one weak one.

This work brings together four strands: pattern-aware strength estimation, evaluation of guessing resistance, breach screening through k-anonymity, and integrated secure storage. We combine them in a single Java system with no dependencies. To the best of the authors' knowledge, existing open tools cover these only separately.

III. PROBLEM STATEMENT

Take a user-supplied password P. Existing composition-rule checkers compute a binary or coarse pass/fail signal f(P) using only character-class membership and length, without considering

how predictable the structure is or whether the password has been exposed in the real world. This leads to two kinds of error. False positives occur when a password that meets the rules (for example "Qwerty123!") is accepted even though it is very easy to guess. False negatives occur when a long, high-entropy passphrase without symbols is rejected even though it resists attacks well. In addition, f(P) says nothing about whether P has already appeared in a known data breach, and there is no way for the user to store a strengthened password securely once it has been chosen. The problem this work addresses is to design a scoring function S(P) that (a) matches true guessing resistance more closely than composition rules alone, (b) uses breach-exposure evidence without compromising the confidentiality of P, and (c) comes with a secure storage mechanism that encourages people to adopt the stronger credentials it produces.

IV. PROPOSED SYSTEM ARCHITECTURE

The proposed system has four modules that work together: the Analysis Engine, Breach Intelligence, the Recommendation Engine and the Encrypted Vault. They sit on a shared Java core and are available through two interchangeable front ends, a console application and a browser-based web application, as shown in Fig. 1.

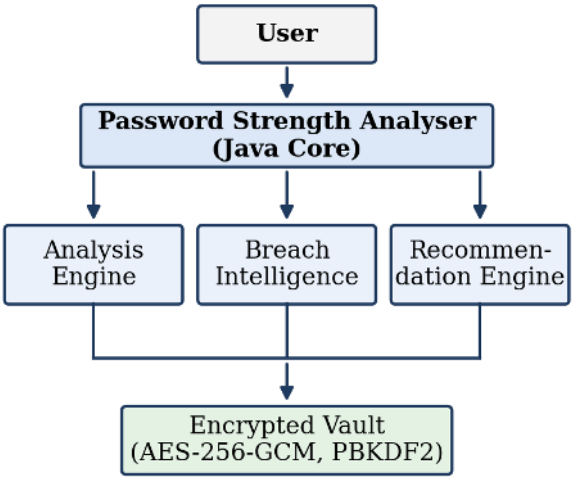


Fig. 1. High-level system architecture.

A. Analysis Engine

This module estimates the effective character pool the password uses and calculates raw Shannon entropy as length × $\log_2$(pool size). A PatternDetector component then looks for keyboard walks, ascending or descending sequences, runs of repeated characters and date-like substrings, and each one costs a fixed number of entropy bits. The adjusted entropy is scaled to a 0-100 score and mapped to one of five categories, from Very Weak to Very Strong. A bundled common-password dictionary is checked alongside the pattern checks: an exact match sets the score to a floor value, and a dictionary substring adds a further entropy penalty (Section VI).

B. Breach Intelligence Module

This module follows the k-anonymity protocol. The password is hashed locally with SHA-1, and

only the first five hexadecimal characters of the hash are sent to a remote breach-lookup API. The service returns every suffix that shares that prefix, and the comparison with the full suffix is finished locally, so the complete password or hash never leaves the client. If no network connection is available, the module quietly falls back to a bundled local corpus.

C. Recommendation Engine

This module links each specific weakness found by the Analysis Engine to a targeted, practical recommendation, such as avoiding keyboard-adjacent sequences, making the password longer, or avoiding embedded dates, instead of showing one generic message.

D. Encrypted Vault

The vault stores approved credentials using authenticated, password-based encryption. A 256-bit AES key is derived from a master password chosen by the user, using PBKDF2-HMAC-SHA256 with 200,000 iterations and a random 16-byte salt. The vault contents are encrypted with AES/GCM/NoPadding, which also detects tampering or a wrong master password.

V. METHODOLOGY

We built the system iteratively and in modules. First we analysed the requirements and identified the limits of composition-rule checkers described in Section II. We then designed the scoring algorithm to combine entropy estimation with pattern-based penalties (Section IV-A). After that, each module (Analysis Engine, Breach Intelligence, Recommendation Engine and Vault) was implemented separately using only the Java Development Kit standard library (java.security, javax.crypto, java.net.http and com.sun.net.httpserver) [8]. Avoiding external dependencies keeps the system portable and easy to audit. Because each module could be tested on its own, we were able to verify the scoring function at unit level before connecting it to the console and web front ends. We then ran functional tests end to end on both front ends, followed by an evaluation on a representative password sample, discussed in Section VII.

VI. IMPLEMENTATION DETAILS

The scoring function starts from a base entropy estimate and applies cumulative penalties. For a password of length L that uses a character pool of size N (determined by which of the lowercase, uppercase, digit and symbol classes are present), the raw entropy is:

$$E_raw = L \times log_2(N)$$

Each structural finding f_i returned by the PatternDetector (keyboard walk, sequential run, repeated characters or date pattern) subtracts a fixed penalty $p_i \in [6, 12]$ bits, and a dictionary-substring match subtracts a further 10 bits. This gives the effective entropy $E_eff = \max(0, E_raw - \Sigma p_i)$. The final score is $S = \min(100, \text{round}((E_eff / 80) \times 100))$. It is also capped at 35 for any password shorter than 8 characters, whatever its character variety, because short passwords are especially vulnerable to brute-force search no matter how large the alphabet is. A password that exactly matches an entry in the bundled common-password corpus skips this calculation and gets a fixed score of 2, since it is at immediate risk of exposure. The 80-bit normalisation constant marks the point where the scale saturates. The penalty values are fixed design parameters, not values fitted to empirical data, a limitation discussed in Section IX. Fig. 2 summarises the complete scoring pipeline.

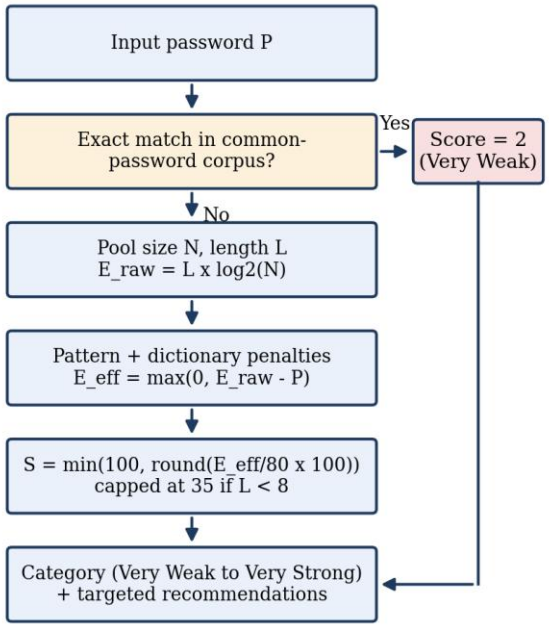


Fig. 2. Scoring pipeline of the Analysis Engine.

For the encrypted vault, AES-256 keys are derived as K = PBKDF2-HMAC-SHA256(masterPassword, salt, 200000). Every save re-encrypts the full vault contents with a freshly generated random 96-bit initialization vector, because reusing an IV under the same key would break the confidentiality guarantees of GCM mode. PBKDF2 [11] and AES-GCM [12] are used as their standards specify, through the JDK implementations [8]. We chose PBKDF2 over the memory-hard scrypt function [9] because it is built into the JDK, which fits the zero-dependency goal. The 200,000-iteration work factor is lower than the 600,000 iterations OWASP currently recommends for PBKDF2-HMAC-SHA256 [10], so it should be raised in a production deployment.

VII. RESULTS AND DISCUSSION

We evaluated the system on ten representative passwords, covering trivial ones, ones that meet composition rules but are still weak, and genuinely high-entropy ones. Table I shows the score, category and entropy computed by the implemented Analysis Engine.

Table I. Analyser Output On Representative Passwords

Password	Score	Category	Entropy (bits)
123456	2	Very Weak	0.0
password	2	Very Weak	0.0
password123	2	Very Weak	0.0
qwerty12345	31	Weak	24.9

Summer2024!	70	Strong	56.1
Tr0ub4dor&9	90	Very Strong	72.1
P@ssw0rd2024	91	Very Strong	72.7
a1B2c3D4e5F6!	100	Very Strong	85.2
xK9#mQ2vL8pR!zT4	100	Very Strong	104.9
correcthorsebatterystaple	100	Very Strong	117.5

The results support our central hypothesis: passwords that merely satisfy composition rules are correctly told apart by their structure. "qwerty12345" meets a naive length-plus-digit rule, but it scores only 31/100 because of its keyboard-walk and sequential-run penalties. "P@ssw0rd2024" and "Tr0ub4dor&9" both use common substitution patterns, yet they still score 90 or higher because their overall entropy stays high despite minor penalties. Trivial passwords and exact corpus matches ("123456", "password", "password123") are correctly floored at a score of 2, which reflects immediate exposure risk rather than a graded weakness. The passphrase-style input "correcthorsebatterystaple" reaches the maximum score even though it uses only lowercase letters. This fits what entropy theory predicts: once length is large enough, it outweighs the pattern-detection penalties.

To put these scores in context, Table II compares the analyser with a conventional composition-rule checker (at least eight characters, containing upper-case, lower-case, digit and symbol characters) and with the zxcvbn estimator [1] (Python port, score 0-4) on the same ten passwords.

TABLE II. Comparison With Composition-Rule And Zxcvbn Baselines

Password	**Composition rule**	**zxcvbn [1] (0-4)**	**Proposed analyser**
123456	Fail	0	Very Weak
password	Fail	0	Very Weak
password123	Fail	0	Very Weak
qwerty12345	Fail	1	Weak
Summer2024!	Pass	2	Strong
Tr0ub4dor&9	Pass	4	Very Strong
P@ssw0rd2024	Pass	1	Very Strong
a1B2c3D4e5F6!	Pass	3	Very Strong
xK9#mQ2vL8pR!zT4	Pass	4	Very Strong
correcthorsebatterystaple	Fail	4	Very Strong

The composition rule rejects the long passphrase "correcthorsebatterystaple" but accepts "P@ssw0rd2024". This reproduces the false-negative and false-positive errors described in Section III. Both the proposed analyser and zxcvbn, by contrast, rate the passphrase highly. The two estimators mostly agree on the trivial passwords, on "qwerty12345" and on the long random strings. The main difference is "P@ssw0rd2024": zxcvbn gives it 1 out of 4 because it models character substitutions, while our analyser rates it Very Strong. "Summer2024!" (a season plus a year) is likewise scored more generously by our analyser than by zxcvbn. This is a real limitation of the current implementation. Substitutions are not normalised before dictionary matching, so well-known mangled dictionary words are over-scored [1], [3], and the Breach Intelligence module captures their exposure better than entropy can. In the same way, the top score for "correcthorsebatterystaple" reflects structural strength only; the phrase is widely published and should be treated as compromised in practice.

When we checked the Breach Intelligence module manually, sample passwords built on common substrings (for example "password123") were correctly flagged as breach-exposed through the local corpus fallback. This confirms that the offline fallback works when the live k-anonymity API cannot be reached. We also verified that the vault encrypts and decrypts successfully only with the correct master password. A wrong master password reliably produced an authentication-tag failure (AEADBadTagException) instead of silently returning corrupted data, which confirms the tamper-evidence property of AES-GCM in this implementation.

VIII. ADVANTAGES

The proposed system has several advantages over conventional password checkers. First, pattern-aware entropy scoring reflects real guessing resistance more accurately than composition rules alone. Second, the k-anonymity breach check tells users whether a password has been exposed at almost no privacy cost, since only a five-character hash prefix is revealed and the full password or hash is never transmitted. Third, the encrypted vault tackles password reuse directly by making it easy to keep unique, strong credentials. Finally, because the Java implementation is built entirely on the JDK standard library with zero dependencies, it is easy to audit, deploy and extend, and it avoids third-party supply-chain risk.

IX. LIMITATIONS AND THREAT MODEL

The system is meant to protect against online and offline guessing of user-chosen passwords and against reuse of credentials that have already been breached. It does not defend against keyloggers or other malware on the host, phishing or shoulder-surfing. The strength of the encrypted vault is limited by the strength of the user's master password, which the analyser can score but cannot enforce. The k-anonymity lookup reveals to the remote service a five-character SHA-1 prefix, and the fact that a check was made. SHA-1 is used only as the lookup key that the protocol requires, not for any security-critical purpose.

The scoring model is heuristic. The 80-bit saturation constant and the 6-12 bit penalties are fixed design parameters, character substitutions are not normalised before dictionary matching, and word-level passphrases are scored by length alone. The bundled offline corpus is small compared with full breached-password lists, and the web front end does not yet separate vaults per user. The evaluation also uses only ten hand-selected passwords and no user study. The results should therefore be read as a proof of concept, not a statistically validated benchmark.

X. CONCLUSION AND FUTURE SCOPE

This paper presented a hybrid password strength analyser that combines entropy estimation, structural pattern detection, privacy-preserving breach intelligence and encrypted credential storage in one dependency-free Java system. Evaluation on representative passwords confirmed that the hybrid approach correctly separates passwords that follow composition rules but are weak from passwords that are genuinely resilient, which addresses a well-documented shortcoming of conventional composition-rule checkers.

Future work includes growing the local dictionary corpus to a full-scale breached-password list so offline operation is more accurate, and adding a Markov-chain or neural guessing model for finer-grained entropy estimation. We also plan multi-user support with per-user vault isolation for the web front end, and a controlled user study, following the evaluation methodology of Ur et al. [4], to measure how the recommendation engine affects real password choices. Other planned improvements are substitution-aware dictionary matching (normalising l33t-style character replacements before lookup), word-level passphrase modelling, and raising the PBKDF2 work factor to current OWASP guidance.

ACKNOWLEDGMENT

The authors thank Prof. Jitendra C. Patil for his guidance throughout this work, and the Department of Computer Engineering, Sandip Institute of Engineering & Management, Nashik, for their support.

REFERENCES

1. D. Wheeler, "zxcvbn: Low-Budget Password Strength Estimation," in Proc. 25th USENIX Security Symp., 2016.
2. J. Bonneau, "The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords," in Proc. IEEE Symp. Security and Privacy, 2012.
3. P. G. Kelley et al., "Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms," in Proc. IEEE Symp. Security and Privacy, 2012.
4. B. Ur et al., "Design and Evaluation of a Data-Driven Password Meter," in Proc. CHI Conf. Human Factors in Computing Systems, 2017.
5. National Institute of Standards and Technology, NIST Special Publication 800-63B: Digital Identity Guidelines - Authentication and Lifecycle Management, 2017.
6. T. Hunt, "I've Just Launched 'Pwned Passwords' V2 With Half a Billion Passwords for Download," Have I Been Pwned, 2018.
7. S. Pearman et al., "Why People (Don't) Use Password Managers Effectively," in Proc. USENIX Symp. Usable Privacy and Security (SOUPS), 2019.
8. Oracle Corporation, Java Platform, Standard Edition Security Developer's Guide, Oracle Help Center.
9. C. Percival and S. Josefsson, RFC 7914: The scrypt Password-Based Key Derivation Function, IETF, 2016.
10. OWASP Foundation, "Password Storage Cheat Sheet," OWASP Cheat Sheet Series.
11. K. Moriarty, Ed., B. Kaliski, and A. Rusch, "PKCS #5: Password-Based Cryptography Specification Version 2.1," RFC 8018, IETF, 2017.
12. M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST Special Publication 800-38D, 2007.

HOW TO CITE: Abhinav Soman*, Krushnraj Rajput, Rohan Gavit, Tushar Chinche, Jitendra Patil, Password Strength Analyser and Security Recommendation System with Breach Intelligence and Encrypted Vault, Int. J. in Engi. Sci., 2026, Vol 3, Issue 11, 40-48. https://doi.org/10.5281/zenodo.24251985