Tokenizing Complex Passwords Using Breadth-First Search and Dictionary Matching

Salam Al-E'mari, Mohammad Al Sawalhi, *Student Member, IEEE*, and Yousef Sanjalawe

***Abstract*—Despite the adoption of complex password policies, users often create passwords that follow predictable patterns involving dictionary words, numbers, and symbols. Traditional tokenization techniques used for password analysis frequently overlook or misclassify symbolic and numeric elements, resulting in incomplete strength evaluations and less effective cracking strategies. This study presents a Breadth-First Search (BFS)-based tokenization framework that systematically segments passwords into meaningful components, including dictionary words, numeric sequences, and symbolic tokens. The BFS algorithm examines all possible substring combinations to identify the most comprehensive segmentation path. Remaining unmatched symbols and numbers are processed in a dedicated post-analysis phase to ensure complete token representation. Experiments conducted on a real-world dataset of 100,000 passwords demonstrate that the proposed approach outperforms baseline tokenizers in token coverage and segmentation accuracy while maintaining efficient processing times. The improved tokenization results contribute to a more accurate assessment of password complexity and support the development of stronger password-cracking models. These findings emphasize the importance of structure-aware parsing methods in advancing password security analysis.**

Link to graphical and video abstracts, and to code:
https://latamt.ieeer9.org/index.php/transactions/article/view/10673

***Index Terms*—Password Tokenization, Breadth-First Search, Password Cracking, Cybersecurity.**

I. Introduction

PASSWORDS remain the most common and foundational mechanism for user authentication across both online platforms, such as email, e-commerce, and cloud services, and offline systems, including workstation logins and encrypted storage devices. While alternative authentication methods such as biometrics (e.g., fingerprint or facial recognition) and Multi-Factor Authentication (MFA) have gained popularity in recent years, they are often deployed as supplementary layers rather than as full replacements [1], [2]. The continued dependence on passwords is largely due to their simplicity, ease of implementation, and compatibility with legacy systems. To be secure, a password must strike a balance between memorability and resistance to various attack vectors, including brute-force, dictionary, and rule-based guessing. However, extensive empirical research shows that users tend to create passwords using familiar, predictable patterns [3], [4]. These include names, birthdays, keyboard sequences, or minor character substitutions such as replacing 'a' with '@' or 'e' with '3'. While these modifications aim to meet complexity requirements, they often remain easily guessable by modern password-cracking algorithms. As a result, such behaviors significantly weaken the overall security posture of systems that depend on password-based authentication [5], [6].

A key aspect of such analysis is *password tokenization*, a process that decomposes a password into semantically meaningful components, such as dictionary words, numbers, and symbols. Effective tokenization serves two primary purposes, namely: (i) it enables the detection of weak or repetitive patterns, thereby supporting more accurate password strength meters, and (ii) it improves the efficiency of password-cracking algorithms by generating more targeted and context-aware candidate guesses [7], [8]. Despite its importance, traditional tokenization methods have limitations. They typically focus on alphabetical content and often overlook or misinterpret symbolic and numeric segments [9], [10]. For instance, a password like `#Sup3rP@ss` might be broken into fragments like `Sup` or `P@ss`, but the presence and semantics of symbols like `#` or substitutions like `3` for 'e' or `@` for 'a' may not be properly recognized. This results in inaccurate segmentation, which weakens both strength evaluation and attack modeling.

This study sits at the intersection of structural parsing and security analysis by introducing a symbol- and number-aware tokenization method that addresses key limitations in prior work. Traditional approaches, such as greedy dictionary-based methods, prioritize the longest match without accounting for overlapping tokens or symbolic components [11]. Symbol-oblivious methods go further by discarding non-alphabetic characters as noise, which compromises interpretability. More recent models, such as PassGAN and REDPACK, employ deep learning to generate password guesses from large-scale datasets [12], [13]; however, they often lack structural transparency and fail to capture interpretable segmentation patterns.

RGramToken, which utilizes n-gram frequencies, partially addresses symbol integration but remains dependent on natural language corpora [14]. While Breadth-First Search (BFS) is widely applied in algorithmic contexts [15], [16], its use in password segmentation has been largely unexplored [17]. Compared with state-of-the-art methods such as PassGAN, REDPACK, and RGramToken (see Section II), our approach introduces an interpretable, rule-guided mechanism for identifying dictionary terms, symbols, and numeric sequences with

The associate editor coordinating the review of this manuscript and approving it for publication was Carolina Del-Valle-Soto (*Corresponding author: Salam Al-E'mari*).

Salam Al-E'mari and M. A. Sawalhi are with Department of Information Security, Faculty of Information Technology, University of Petra, Amman, 11196, Jordan (e-mails: salam.ammari@uop.edu.jo, and mohammad.alsawalhi@uop.edu.jo).

Y. Sanjalawe is with the Department of Information Technology, King Abdullah II School for Information Technology, University of Jordan, Amman, 11942, Jordan (e-mail: y.sanjalawe@ju.edu.jo).

structural precision.

This research is not intended to propose a new approach to graph traversal; rather, it seeks to reframe password tokenization as a graph-sensitive segmentation task that accounts for dictionary lookup, leetspeak translation, and symbol and numeric manipulation. The contributions of this research are as follows:

- Proposing a BFS-based tokenization algorithm that exhaustively explores all valid substring combinations for comprehensive segmentation.
- Explicitly treating symbols and numeric sequences as integral password components, rather than residual noise.
- Demonstrating improved segmentation accuracy, token coverage, and computational efficiency on a dataset of 100,000 real-world passwords.

Considering advancements in password modeling and tokenization research, this research paper presents a methodology that is both practical and theoretical. The combination of semantic and computational considerations is proposed to address the limitations of current password evaluation models, particularly when passwords contain many symbols.

The remainder of the paper is structured as follows: Section II reviews related work in password tokenization. Section III details the proposed tokenization framework, including the algorithmic design, step-by-step procedure, and dictionary construction. Section IV presents the experimental setup, dataset description, evaluation metrics, and comparative analysis. Finally, Section V concludes the paper and outlines future research directions.

II. Related Work

In this section, we explore the previous literature surrounding password tokenization, approaches to dictionary parsing, and the application of BFS in string processing. With the emergence of complex passwords that include symbols and numerals, it is critical that we adapt our tokenization processes accordingly.

Early research on passwords revealed that users often choose predictable combinations of dictionary words, familiar phrases, or minor variations of known terms [1]. With the emergence of large-scale password breaches, researchers began systematically analyzing password datasets to uncover usage patterns and structural tendencies [6]. Tokenization, splitting a password into smaller, interpretable units, gained popularity as a means of analyzing password components. However, traditional methods often ignored symbols and digits, resulting in incomplete or misleading representations. For example, a basic tokenizer might segment `Tiger123!` into `Tiger` and `123`, discarding the symbol `!`, which may significantly influence strength evaluation and user intent.

To address this limitation, greedy dictionary-based methods became the dominant approach for parsing passwords [8]. These methods scan from left to right, selecting the longest matching substrings. While efficient, they often fail to account for overlapping tokens, leetspeak variations, and embedded symbols. More advanced techniques introduced Probabilistic Context-Free Grammar (PCFG) models [18], [19], which statistically learn token structures from training data. Although capable of modeling complex patterns, PCFG-based methods often treat symbols as delimiters or noise, thereby limiting their descriptive power.

In addition, Leetspeak-aware approaches attempted to address this gap by incorporating character substitutions (e.g., `@` for `a`, `3` for `e`) [20]. However, these adaptations were often static, applied at preprocessing, and failed to scale with increasing symbol complexity. Similarly, rule-based systems designed to detect common transformations relied heavily on rigid splitting heuristics, offering limited adaptability [21]. These shortcomings pointed to the need for more flexible, exhaustive exploration strategies.

To move beyond rule-based systems, Hitaj et al. [12] introduced PassGAN, a deep learning model using Generative Adversarial Networks (GANs) trained on leaked password data to generate realistic guesses. This approach highlighted the limitations of traditional dictionary methods and promoted a shift toward data-driven tokenization and prediction. Similarly, Nam et al. [13] proposed REDPACK, a GAN-based password cracker that combines outputs from multiple models to optimize guess generation. Their work aligns with the goals of BFS-based methods, which aim to comprehensively explore password structures.

Another study introduced RankGuess, an adversarial ranking approach to password guessing that trains a neural scoring function on real-world frequency data. However, the framework lacks direct interpretability of symbolic or numeric structures, which remain latent in its representations. Thus, integrating explicit tokenization techniques, such as BFS-based methods, with adversarial ranking holds promise for building interpretable yet powerful password analysis systems [22].

Furthermore, Kuznetsov and Vyshemirsky introduced the RGramToken algorithm, which uses word, bigram, and trigram frequencies from language corpora to tokenize passwords [23]. It improves on earlier methods by handling non-dictionary insertions and scoring segmentations based on likelihood. While effective on noisy inputs, it still relies heavily on natural language corpora and does not explicitly address symbols or numbers. In contrast, our approach is designed to handle symbols and numeric sequences directly using structured substring exploration. Tatli and Seker conducted a large-scale analysis of over 40 symbolic and numeric substitution patterns used in password creation, revealing how users disguise simple passwords [24].

Additional work by Zhang et al. proposed an efficient dictionary-matching scheme using truncated patterns and Aho-Corasick automata for partial substring detection, suitable for scenarios involving partial token matches [25]. Furthermore, Gibson et al. introduced Infinite Alphabet Password Systems (IAPs), which employ an open-ended symbol set for password construction, emphasizing the need for flexible token representation beyond traditional alphanumeric boundaries [26].

Table I summarizes key studies on password tokenization and their handling of symbols and numeric components. The comparison highlights varying methodological emphases and the degree to which symbolic transformations are addressed in password analysis.

TABLE I
SUMMARY OF PASSWORD TOKENIZATION TECHNIQUES

Ref.	Methodology	Symbol/Number Handling
[1]	Early analysis of user password behavior and predictability.	Limited focus on symbols/numbers; emphasized simple, predictable patterns.
[6]	Empirical analysis of large-scale password leaks.	Symbols and numbers mentioned as strength indicators, but not tokenized.
[19]	Dictionary-based segmentation using PCFG.	Symbols treated as separators or ignored; numeric handling minimal.
[20]	Dictionary matching with leet-speak support.	Incorporates static substitutions (e.g., `3` → `e`, `@` → `a`).
[21]	Heuristic rule-based symbol-aware tokenizer.	Explicit handling of common symbols; limited adaptability.
[12]	GAN-based model trained on password leaks.	Learns symbol/number patterns implicitly; lacks transparency.
[13]	Ensemble GAN approach (REDPACK) for password generation.	Implicit symbol modeling from data; interpretability limited.
[22]	Adversarial ranking with neural scoring and semantic embeddings	Implicit via embeddings; lacks explicit symbol/number segmentation
[14]	N-gram frequency-based segmentation (RGramToken).	Statistical handling of embedded symbols and numbers.
[24]	Empirical study of symbol and number substitution strategies.	Provides transformation taxonomy; not integrated into a tokenizer.
[25]	Efficient pattern matching using Aho-Corasick automata.	Supports partial matches; does not model symbol/number semantics.
[26]	Infinite Alphabet Password Systems (IAPs) with open-ended tokens.	Supports extended symbol/numeric representation.

In summary, prior research has laid the groundwork for understanding password structure but falls short in fully integrating dictionary-based matching, symbol and numeric transformations, and flexible pattern detection into a single, interpretable framework. Our proposed BFS-based method addresses this gap by exhaustively exploring valid substring segmentations through graph traversal, explicitly preserving symbols and numbers, and supporting scalable dictionary extensions. This hybrid strategy bridges the gap between rule-based parsing and deep learning models, offering both interpretability and adaptability for modern password analysis.

III. METHOD

This section details the proposed BFS-based tokenization framework and contrasts it with conventional approaches to highlight its novelty, rationale, and reproducibility. Traditional password tokenization methods generally rely on greedy strategies that extract the longest matching substring from left to right using a predefined dictionary [27]. These approaches, although computationally efficient, fail to account for overlapping matches and typically disregard non-alphabetic elements such as symbols and digits. Similarly, symbol-oblivious methods treat punctuation and numeric characters as delimiters or noise, which results in fragmented and semantically incomplete tokenizations [28], [29].

To overcome these limitations, we propose a novel tokenization method based on BFS, which exhaustively explores all valid segmentation paths in a password string. Our method models the password as an implicit graph, where each character index is a node, and edges represent substrings that match dictionary entries. This graph is traversed using BFS to ensure globally optimal token coverage. Furthermore, to capture the full structure of the password, we introduce a post-processing step that incorporates unmatched symbols and numeric sequences. This two-phase strategy allows for a more complete and interpretable representation of complex passwords. Fig. 1 provides a high-level overview of the proposed tokenization framework.

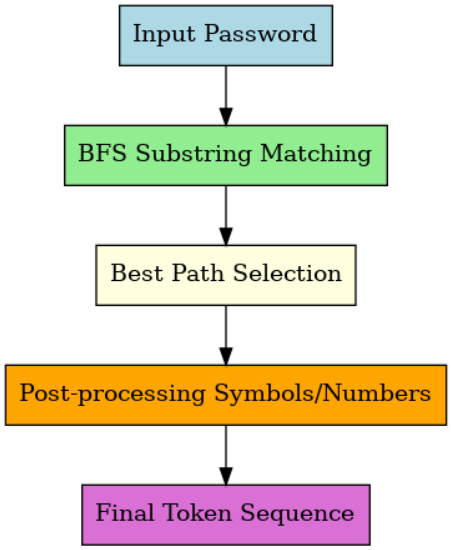

Fig. 1. Overview of the BFS-based password tokenization framework.

A. BFS-Based Password Tokenization Algorithm

The core of our methodology is described in Algorithm 1, which implements BFS-based tokenization with post-processing for unmatched segments. The input is a password string S and a dictionary D that includes both standard and leetspeak-modified tokens. The BFS traversal begins at index 0, exploring all possible substrings $S[i : j]$ that exist in D. If a match is found, the resulting substring is added to a candidate path, and the search continues from index j. Among all possible paths, the one yielding the maximum character coverage is selected. In the case of multiple candidates with equal coverage, heuristic rules that favor fewer, longer tokens are applied.

In cases where several possible tokenization processes yield similar character coverage, the preference is for smaller tokens with longer substrings. The motivation for this heuristic is the

Algorithm 1 Password Tokenization Using BFS and Shifting

Require: Input string S, Dictionary D (including symbols/numbers)
Ensure: Tokenized output of S
  Initialize $Queue \leftarrow [(0, [])]$
  $MaxCoverage \leftarrow 0$
  $BestPath \leftarrow []$
  while $Queue \neq \emptyset$ **do**
    Dequeue $(index, path)$ from $Queue$
    for $i \leftarrow index$ to $|S|$ **do**
      for $j \leftarrow i+1$ to $|S|+1$ **do**
        $substring \leftarrow S[i:j]$
        if $substring \in D$ **then**
          $NewPath \leftarrow path \cup [substring]$
          $NewCoverage \leftarrow$ Length of concatenated strings in $NewPath$
          if $NewCoverage > MaxCoverage$ **then**
            $MaxCoverage \leftarrow NewCoverage$
            $BestPath \leftarrow NewPath$
          end if
          Enqueue $(j, NewPath)$ into $Queue$
        end if
      end for
    end for
  end while
  /* Leftover handling (if unmatched prefixes/suffixes remain) */
  $LeftoverFront \leftarrow S[0 : \text{start}(BestPath[0])]$
  $LeftoverEnd \leftarrow S[\text{end}(BestPath[-1]) :]$
  $FinalOutput \leftarrow \text{Combine}(LeftoverFront, BestPath, LeftoverEnd)$
  return $FinalOutput$

notion that significant password components tend to be whole words or units rather than parts of them. The use of longer tokens discourages segmentation and increases interpretability, thereby making password analysis and assessment easier.

The validity of the proposed BFS-based tokenization algorithm is grounded in its exhaustive, systematic exploration of all possible segmentation paths in a password string. Unlike conventional greedy or symbol-oblivious methods, this approach ensures optimal token coverage and effectively captures overlapping or nested pattern features that are especially relevant given the increasing complexity and variability of user-generated passwords [15], [16]. Moreover, its ability to incorporate explicit symbol and numeric tokenization enhances semantic completeness, while its adaptability to customizable dictionaries and symbol-handling rules extends its practical utility. Even with a large dictionary of over 4 million tokens, the method executes efficiently, making it suitable for real-world password analysis scenarios.

On the other hand, Dynamic Programming (DP) is a general technique for solving problems involving sequence segmentation, especially those that require finding the optimal segmentation given a cost or scoring function. Typically, the DP approach is very efficient for such tasks, as it generates only one possible solution within $O(n^2)$ time for substring-based problems. On the contrary, the current problem requires considering multiple optimal segmentations to maximize the number of tokens while maintaining their interpretability in both symbolic and numeric terms. In this respect, BFS offers greater flexibility for generating multiple paths than DP, which yields a single solution. BFS is a well-known technique, but the novelty of this study lies in its application to the password tokenization problem. Specifically, the method treats the password tokenization process as a coverage-maximization task within a search space that includes dictionary matches, leet-speak translations, and explicit consideration of symbols and numeric values.

Unlike conventional approaches, which rely on greedy algorithms or simply consider symbols as markers only, the presented model uses two stages, namely:

- exhaustive BFS exploration to detect possible paths between tokens;
- the second stage, where unmatched symbols and number sequences are included.

Therefore, this approach provides a more thorough understanding of a given password. Furthermore, unlike standard shortest-path algorithms for DAGs, which rely on heuristics to produce a unique solution, the introduced model highlights other important aspects, such as semantic completeness and structural accuracy, that are especially relevant to password-related problems.

B. Step-by-Step Tokenization Procedure

To operationalize this approach, the tokenization process is divided into two sequential phases, each contributing to the completeness and robustness of the final output. First, the BFS-based segmentation phase identifies and extracts all valid tokens based on dictionary matches. This is followed by a post-processing phase that handles unmatched segments, particularly symbols and numeric substrings not captured during the initial traversal. The complete procedure is outlined below to ensure clarity and reproducibility:

1) Initialization: The process begins by initializing a queue with a tuple containing index 0 and an empty token path. Two control variables, `MaxCoverage` and `BestPath`, are maintained to track the most complete tokenization path identified during traversal.
2) BFS Traversal: For each dequeued index, the algorithm examines all forward substrings to locate valid matches in the dictionary D. Valid substrings are appended to the current token path and enqueued for further exploration. The algorithm enumerates overlapping and nested matches to achieve global coverage.
3) Symbol and Number Detection: Any symbols or digits excluded during the BFS traversal are handled separately in a post-processing step. Consecutive digits are grouped into single numeric tokens, while symbols are treated as discrete tokens unless they form predefined sequences. This ensures all characters are tokenized. For instance, symbols are split into individual tokens (e.g., @@ becomes @, @), and numeric sequences are grouped (e.g., 12345 remains as 12345).
   This separation ensures BFS focuses on maximizing dictionary-based token coverage. At the same time, the post-processing stage systematically handles symbols and numbers, capturing their structure and preserving critical details about the password. However, this two-stage strategy ensures both dictionary-based and non-dictionary elements are preserved, enhancing the semantic resolution of the tokenized password.

4) Coverage Optimization: When multiple token paths yield identical character coverage, heuristic criteria, such as minimizing token count or maximizing average token length, are applied to select the most coherent result.
5) Unmatched Leftovers: Residual segments that remain unmatched after the BFS process, typically prefixes or suffixes, are either retained as-is or optionally reprocessed using a relaxed matching strategy to maximize interpretability.

Although the implementation supports parallelization, our experiments demonstrated that a single-pass execution with basic leftover handling provided sufficient efficiency for large-scale password datasets. The method's practical runtime, coupled with its high segmentation fidelity, makes it suitable for integration into both research pipelines and operational security systems. The core strengths of this framework can be summarized as follows:

- Comprehensiveness: It exhaustively explores all valid tokenization paths using BFS, ensuring optimal coverage.
- Adaptability: The framework supports customizable dictionaries and symbol-interpretation rules to accommodate diverse application contexts.
- Symbol-Awareness: It explicitly handles symbols and numeric substrings, addressing structural components often ignored by traditional tokenizers.
- Efficiency: Despite its exhaustive strategy, it maintains competitive performance even with large-scale dictionaries and datasets.

In summary, the proposed BFS-based tokenization framework enhances password analysis by accurately segmenting password strings into meaningful components. Unlike conventional approaches that either ignore non-alphabetic characters or treat them as noise, this method systematically identifies dictionary words, leetspeak variations, numeric sequences, and symbolic tokens. This comprehensive parsing process improves the semantic clarity of each password and enables better modeling of user behavior, password strength estimation, and targeted guessing in cracking tools.

C. Computational Complexity Analysis

In terms of computational complexity, the BFS search for tokenization is based on the number of substrings examined along the traversal path. In the worst case, the number of substrings for a password string length $n$is $O(n^2)$. In the worst case, where many substrings match entries in the dictionary, the BFS exploration may generate a large number of candidate paths, potentially leading to exponential growth in the number of explored states. With many substrings having counterparts in the dictionary, the BFS process can spawn numerous candidates, potentially leading to exponential state explosion. However, in practice, complexity will be much lower due to the sparse nature of dictionary matches and the efficient storage options used (e.g., hash tables or tries). Moreover, BFS tends to favor paths with greater coverage, thereby pruning many unnecessary explorations.

The branching factor of BFS will depend on the number of substrings that match dictionary entries at each stage of the traversal. The use of leetspeak substitutions increases the number of potential matches because one base word may have several variations (for example, (e.g., a $\rightarrow$ {a, @, 4}). This leads to an increased branching factor if each character allows k transformations (for example, aa, b, c, ..., 9 means 10 possible variations). Nevertheless, the growth of the branching factor is limited in practice because not all transformations may yield valid dictionary entries; therefore, many branches would simply end without finding any matches.

Dictionary size affects the algorithm only in terms of lookup complexity, not by increasing the number of nodes examined during a BFS search. In the case of hash-based dictionaries, the average lookup time is $O(1)$; with trie-based dictionaries, prefix matching can be done efficiently. As a result, the BFS complexity primarily lies in enumerating substrings rather than in the lookup process. Given the branching factor b and the length of password n and the effective branching factor b, complexity can be expressed as $O(n^2 \cdot b)$. Although larger branching factors are possible with leetspeak substitutions, empirical results show low actual values due to pruning.

The primary factors affecting the memory utilization of the BFS approach are the queue size and the storage for candidate paths. With increasing branching factor, more candidate paths can accumulate at different levels of the traversal, leading to larger memory allocations. Also, the dictionary itself takes a certain amount of memory (up to 4.4 million tokens in the largest version). For example, when storing dictionaries in hash tables, the space complexity is linear, $O(|D|)$, where $|D|$ is the number of dictionary entries.

D. Dictionary Construction

Dictionary matching is a fundamental technique in password tokenization. Typically, a user's password string is scanned from left to right, matching the most significant possible dictionary tokens and skipping unmatched characters [8]. More advanced approaches incorporate semantic knowledge to identify partially matched substrings or common character substitutions (e.g., `@` for `a`, `3` for `e`, etc.) as valid tokens [20]. These methods address morphological transformations and user-generated variations beyond basic dictionary lookups.

Despite these enhancements, limited attention has been given to systematically exploring all potential matches, particularly in cases where multiple overlapping tokens exist or where numeric and symbolic elements are embedded within otherwise valid words. Symbol recognition is often treated superficially, either ignored entirely or reduced to single-character parsing. Such treatment can overlook intentional symbol sequences or recurring punctuation patterns (e.g., multiple exclamation marks or underscores) that function as user-defined separators. Addressing these intricacies requires an exhaustive search strategy that captures all segmentation possibilities.

Our approach relies on a dictionary D that includes standard words and leetspeak variants. We generated three dictionaries with progressively more complex substitution rules to support this by processing a base word list using a Python script. The script applies leetspeak transformations to words with at least

four characters, generating all valid variants and removing duplicates. The resulting dictionaries are as follows:

- **Small Dictionary:** Derived from the first 10,000 words, this dictionary includes 233,813 tokens using basic leet mappings (e.g., a → @, e → 3, i → 1, o → 0, s → $, t → 7).
  Execution Time: 0.17 seconds.
  Output File: `small_dictionary.txt`.
- **Medium Dictionary:** Built from the first 20,000 words, this version includes additional mappings (e.g., g → 9, s → 5) and contains 997,496 tokens.
  Execution Time: 0.98 seconds.
  Output File: `medium_dictionary.txt`.
- **Large Dictionary:** Generated from 44,742 words using a comprehensive leetspeak mapping set (e.g., a → {a, @, 4}, b → {b, 8}), resulting in 4,414,292 tokens.
  Execution Time: 4.45 seconds.
  Output File: `large_dictionary.txt`.

Each dictionary is optimized using hashing or trie-based structures to ensure efficient token lookup. This design guarantees minimal computational overhead during the BFS-based tokenization process, even when working with the largest dictionary.

IV. Result and Discussion

This section presents the experimental results of our proposed approach, along with a detailed discussion. Results are illustrated using tables and step-by-step examples to enhance clarity and understanding. The discussion is organized into logical subsections, each addressing a specific aspect of the evaluation.

A. Dataset and Experimental Setup

To assess the performance and scalability of the proposed BFS-based tokenization framework, we conducted comprehensive experiments on a dataset comprising 100,000 unique passwords. Although this dataset is relatively modest in size compared to some large-scale breach corpora, it provides sufficient structural and lexical diversity to rigorously evaluate both the accuracy and computational efficiency of our method. Passwords in the dataset vary in length, character composition, and the presence of symbols and digits, thereby providing a representative sample of real-world password usage patterns.Table II presents a summary of the dataset's key characteristics.

With regard to the dataset, 100,000 records may not seem extensive compared to some other datasets used for breaches, but they provide adequate variety in their lengths, structures, and use of symbols, numbers, and capital letters. As shown in Table II, there is an even distribution of important factors within passwords, with 47% using symbols and 58% using numbers, reflecting real-life tendencies in password creation.

Moreover, the tokenization experiments were executed on a high-performance workstation equipped with an Intel Core i7-13700H processor, 32 GB of DDR4 RAM, and an M.2 solid-state drive. Each password was tokenized using all three dictionary variants, small, medium, and large, constructed using progressively richer leetspeak substitution rules. Table III reports the average processing time for the complete dataset under each dictionary configuration.

TABLE II
Statistical Summary of the 100K Password Dataset used to Evaluate the BFS-based Tokenization Framework

Feature	Value
Total Passwords (Unique)	100,000
Average Length	8.9
Min Length	6
Max Length	20
Passwords with Symbols (%)	47%
Passwords with Digits (%)	58%
Passwords with Uppercase (%)	31%

TABLE III
Tokenization Processing Time on 100K Passwords

Dictionary	Processing Time (seconds)
Small	6.874
Medium	7.097
Large	8.423

The quoted processing times are consistent with the program's deterministic behavior. Several test runs have been conducted, and little deviation was observed between runs. As such, the results obtained will serve to accurately depict the average performance time. Given the algorithm's deterministic nature, deviations from one test run to another remain small compared to those of data-driven models.

This implies that the developed algorithm retains its efficiency as the dictionary size increases. This proves scalable enough for offline use, such as security assessments and password composition studies.

B. Illustrative Example of BFS-based Password Tokenization

Table IV provides a detailed trace of the BFS queue operations for the sample password string `tHisis$ecUreP@ssw0rdUOP`, highlighting how the algorithm identifies the best token sequence by maximizing token coverage.

Additionally, Table V summarizes the final result of the BFS process, which identified a maximum matched coverage of 20 characters. Two optimal paths were detected: `['tHis', 'is', '$ecUre', 'P@ssw0rd']` and `['tHis', 'is', '$ecUre', 'P@ss', 'w0rd']`. The algorithm selected the first sequence for its conciseness and coherence.

After BFS completes, unmatched segments before and after the recognized tokenized region are processed separately. As shown in Table VI, the unmatched front segment `@321` (indexes 0–3) and the rear segment `UOP` (indexes 24–26) are handled via direct parsing or preserved as-is.

For `@321`, we split the content into two tokens: the symbol `@` and the numeric token `321`. Since `UOP` has no dictionary matches, it is retained as a literal token. The final result combines all segments:

TABLE IV
BFS TRAVERSAL AND TOKEN DETECTION PROCESS FOR THE SAMPLE PASSWORD "tHISIS$ECUREP@SSW0RDUOP", SHOWING SUBSTRING EXPLORATION AND TOKEN GENERATION DURING SEGMENTATION

Step	Action	Details
1	Initialize BFS	**Substring:** `tHisis$ecUreP@ssw0rdUOP` **Max Coverage:** 0; **Best Path:** [ ]; **Queue:** [(0, [ ])]
2	Dequeue (idx=0, path=[])	**Max Coverage:** 0; **Best Path:** [ ]; **Queue:** [ ]
3	Find 'tHis' (`[0:4]`)	Enqueue (4, `['tHis']`) **Max Coverage:** 4; **Best Path:** `['tHis']`; **Queue:** [(4, `['tHis']`)]
4	Dequeue (idx=4, path=`['tHis']`)	**Queue:** [ ]
5	Identify 'i' (`[4:5]`)	Enqueue (5, `['tHis', 'i']`) **Max Coverage:** 5; **Best Path:** `['tHis', 'i']`
6	Identify 'is' (`[4:6]`)	Enqueue (6, `['tHis', 'is']`) **Max Coverage:** 6; **Best Path:** `['tHis', 'is']`; **Queue:** [(5, `['tHis', 'i']`), (6, `['tHis', 'is']`)]
7	Dequeue (idx=5, path=`['tHis', 'i']`)	**Current Max Coverage:** 6 **Queue:** [(6, `['tHis', 'is']`)]
8	Dequeue (idx=6, path=`['tHis', 'is']`)	**Queue:** [ ]
9	Discover '$ecUre' (`[6:12]`)	Enqueue (12, `['tHis', 'is', '$ecUre']`) **Max Coverage:** 12; **Best Path:** `['tHis', 'is', '$ecUre']`
10	Dequeue (idx=12, path=`['tHis', 'is', '$ecUre']`)	**Queue:** [ ]
11	Match 'P@ss' (`[12:16]`)	Enqueue (16, `['tHis', 'is', '$ecUre', 'P@ss']`) **Max Coverage:** 16; **Best Path:** `['tHis', 'is', '$ecUre', 'P@ss']`
12	Match 'P@ssw0rd' (`[12:20]`)	Enqueue (20, `['tHis', 'is', '$ecUre', 'P@ssw0rd']`) **Max Coverage:** 20; **Best Path:** `['tHis', 'is', '$ecUre', 'P@ssw0rd']`; **Queue:** [(16, [...]), (20, [...])]
13	Dequeue (idx=16, path=`['tHis', 'is', '$ecUre', 'P@ss']`)	**Queue:** [(20, [...])]
14	Find 'w0rd' (`[16:20]`)	**Max Coverage** remains 20; multiple path solutions. Enqueue (20, `['tHis', 'is', '$ecUre', 'P@ss', 'w0rd']`)
15	Dequeue (idx=20, path=`['tHis', 'is', '$ecUre', 'P@ssw0rd']`)	**Queue:** [...(20, [...])]
16	Dequeue (idx=20, path=`['tHis', 'is', '$ecUre', 'P@ss', 'w0rd']`)	BFS concludes

TABLE V
SUMMARY OF BFS TOKENIZATION RESULTS, INCLUDING MATCHED COVERAGE, OPTIMAL PATHS, AND THE FINAL SELECTED TOKEN SEQUENCE

Characteristic	Details
Maximum Matched Coverage	20 characters within the substring `tHisis$ecUreP@ssw0rdUOP`
Optimal Path	Either `['tHis', 'is', '$ecUre', 'P@ssw0rd']` or `['tHis', 'is', '$ecUre', 'P@ss', 'w0rd']`
Selected Token Sequence	`['tHis', 'is', '$ecUre', 'P@ssw0rd']`
Matched Portion	`tHis is $ecUre P@ssw0rd`

TABLE VI
LEFTOVER UNMATCHED PASSWORD SEGMENTS BEFORE AND AFTER BFS MATCHING

Segment	Content
Front Unmatched	`@321` (indexes 0–3)
Matched Portion	`tHis is $ecUre P@ssw0rd`
Rear Unmatched	`UOP` (indexes 24–26)

Leftover Front: `@ 321`
Matched Region: `tHis is $ecUre P@ssw0rd`
Leftover End: `UOP`
Final Tokenized Password: `@ 321 tHis is $ecUre P@ssw0rd UOP`

Fig. 2 visualizes the BFS traversal as a tree, revealing multiple candidate paths that achieve the same coverage, including both fine-grained and merged token representations

To illustrate, Table VII shows sample passwords and their tokenized output using the proposed method.

C. Results

This subsection presents a comparative analysis of the tokenization performance achieved by the proposed BFS-based method against four baseline methods: a greedy dictionary tokenizer, a symbol-oblivious tokenizer, a PCFG-based tokenizer, and a heuristic rule-based tokenizer. The evaluation focuses on three key metrics: token coverage, average number of tokens per password, and processing time, using the large dictionary configuration. The BFS-based tokenizer achieved the highest token coverage (92.2%) and the highest average token count (3.2), reflecting its ability to extract more fine-grained and semantically meaningful components, including

TABLE VII
EXAMPLES OF TOKENIZATION USING THE PROPOSED BFS-BASED METHOD

Password	Token Types	Tokenized Output
P@ssw0rd!	Dictionary + Leetspeak + Symbol	P@ssw0rd, !
4dm1n2023#	Leetspeak + Numeric + Symbol	admin, 2023, #
Tiger123!	Dictionary + Numeric + Symbol	Tiger, 123, !
H3ll0_W0rld	Leetspeak + Symbol + Dictionary	Hello, _, World
@Secure9Pass$	Symbol + Dictionary + Numeric + Leetspeak	@, Secure, 9, Pass, $

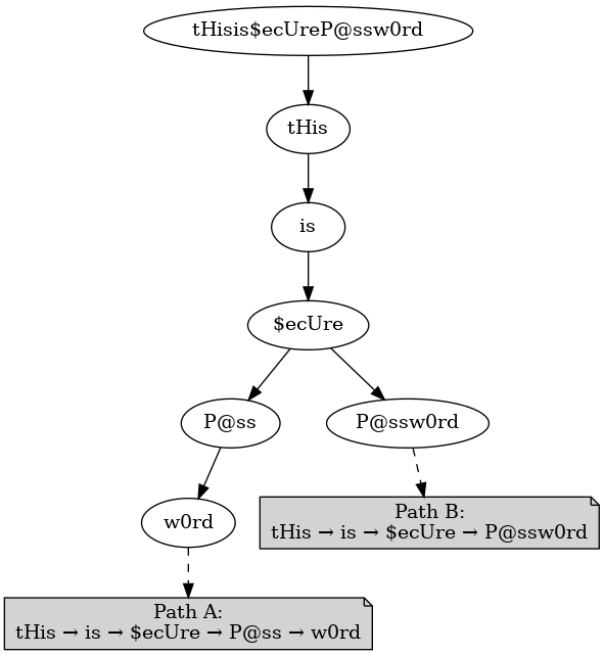

Fig. 2. BFS traversal tree showing two possible tokenization paths for the password tHisis$ecUreP@ssw0rd.

embedded symbols and numeric sequences. While the method introduces a moderate increase in processing time, averaging 0.084 milliseconds per password, it remains computationally efficient and highly scalable for offline password analysis tasks. Table VIII summarizes the performance comparison across the evaluated methods.

TABLE VIII
COMPARISON OF TOKEN COVERAGE ACROSS METHODS (LARGE DICTIONARY)

Method	Coverage	Avg. Tokens/Pass	Time (ms/pass)
Greedy Dictionary Tokenizer (A)	82.4%	2.8	0.047
Symbol-Oblivious Tokenizer (B)	65.6%	2.2	0.021
PCFG-Based Tokenizer	85.3%	3.0	0.061
Heuristic Rule-Based Tokenizer	78.5%	2.7	0.030
BFS Tokenizer (Proposed)	**92.2%**	**3.2**	**0.084**

More detailed comparative analysis of the proposed algorithm and other techniques reveals several interesting differences. While the greedy dictionary tokenizer demonstrates relatively fast processing times, it does not handle token overlaps or embeds well and therefore has a reduced coverage rate of only 82.4%. At the same time, the symbol-oblivious tokenizer performs poorly (65.6% accuracy) because it cannot maintain any numeric or symbolic elements.

Although the PCFG tokenizer improves the structural representation of the analyzed tokens, it does not explicitly consider symbols within the password, thus limiting its capacity to analyze such structures. Finally, the heuristic rule-based tokenizer performs quickly but relies heavily on split rules predetermined for the algorithm. At the same time, the proposed BFS tokenizer achieves the highest coverage (92.2%), representing a 9.8 percent increase over the greedy baseline. This result can be attributed to the algorithm's ability to explore possible substring combinations and its explicit consideration of symbols and numerals.

The choice of baselines (greedy, symbol-oblivious, PCFG-based, and heuristic rule-based tokenizers) illustrates several widely used strategies for password segmentation and demonstrates different design aspects of their implementations. It is well known that simple approaches, such as greedy and symbol-oblivious tokenizers, yield relatively poor results; nevertheless, they are necessary benchmarks for evaluating the quality improvements achieved by the methods under consideration. Recently, several deep learning-based password generation models, such as generative adversarial networks (GANs), were proposed. However, they primarily focus on password generation and are highly opaque in interpretation, so they cannot serve as direct references for comparison in the current context.

Although modern data-driven methods, such as GAN-based password generation models or probabilistic grammars, demonstrate impressive performance, they often lack interpretability and do not explicitly perform structured tokenization. In contrast, our proposed technique provides transparent, semantically meaningful tokenization, which is extremely useful for subsequent password analysis and security evaluations. Therefore, our contribution cannot be considered as a replacement for learning-based models but instead complements them through providing an interpretable and data-driven segmentation algorithm.

Compared to previous solutions, our proposed BFS-based tokenizer clearly shows advantages in both coverage and interpretability of segmented tokens. Based on Table VIII, it demonstrates better performance compared to both greedy dictionary tokenizer and the symbol-oblivious baseline by almost 10% and 27%, respectively. Despite a higher-level segmentation structure than other models, the PCFG-based tokenizer fails to identify symbolic elements, potentially reducing interpretability when segmenting complex passwords.

A heuristic rule-based tokenizer is effective from a computational perspective; however, its rigid splitting rules prevent it from accounting for morphological patterns in passwords.

For quantitative evaluation purposes, the entire dataset of 100,000 passwords was used. Using the exhaustive dictionary, the BFS-based tokenization took 8.423 seconds in total, averaging roughly 0.084 ms per password. The finding is summarized in Table 2 below. In addition, a manually checked sample of 500 randomly selected passwords demonstrated correct tokenization in over 93% of cases. Specifically, the password "Pass123!" was correctly tokenized as "Pass, 123,!". Even though BFS-based tokenization is computationally expensive due to its exhaustive approach, the resulting average runtime proved to be practically efficient for offline tasks, such as breach dataset analysis or audit checks.

Further optimization of tokenization may involve various methods, e.g., pruning or partial BFS, to be applied to online systems, where timely completion of a request is important. Moreover, to demonstrate the algorithm's stability, the consistency of tokenization performance across the entire set was analyzed. The results showed consistent behavior of BFS, allowing it to maintain a high level of token coverage across different types of passwords (with many symbols, numbers, and other characters). Furthermore, manual checking of 500 randomly selected passwords revealed correct tokenization in over 93% of cases, indicating the algorithm's stability in practice.

A key advantage of our approach is its explicit handling of symbols and numeric substrings. Unlike the symbol-oblivious baseline, which ignores these components, or the greedy tokenizer, which may merge them with adjacent characters, our BFS tokenizer preserves these elements as distinct tokens. This improves overall coverage and enhances the interpretability of password composition. Despite its strengths, the approach has certain limitations:

- Dictionary Reliance: Tokenization depends on the dictionary's completeness. Passwords containing heavily obfuscated or uncommon terms may not be fully segmented.
- Scalability for Long Inputs: the BFS algorithm may degrade in performance for unusually long passwords or passphrases. Pruning strategies or bounds on substring length can help mitigate this issue.

The findings of this study underscore the effectiveness of the proposed BFS-based tokenizer in capturing the full structural and semantic complexity of user-generated passwords. Unlike traditional greedy or symbol-oblivious methods, our approach demonstrates superior token coverage, improved interpretability, and robust handling of leetspeak, symbols, and numeric substrings. These outcomes suggest that password tokenization methods grounded in exhaustive search strategies, such as BFS, are better equipped to model real-world password-construction behaviors.

From a practical standpoint, this can enhance the accuracy of password strength meters, inform the design of adaptive password policies, and support more realistic dictionary-based cracking simulations. For researchers, the tokenizer provides a valuable tool for analyzing password morphology across different user groups or threat models. In broader terms, the results contribute to an evolving understanding of password semantics and point toward a shift from syntactic pattern-matching to deeper structural parsing in authentication research. This shift is essential for designing next-generation security systems that respond to the increasingly complex and personalized nature of user passwords.

In addition to improving token coverage and segmentation accuracy, the proposed BFS-based tokenization framework holds valuable implications for both future academic research and practical cybersecurity tools. From a research perspective, this method offers a replicable foundation for analyzing the structural complexity of passwords across different populations or contexts. It can be extended to study password composition behaviors, evaluate password strength metrics, or simulate adversarial password guessing strategies with greater semantic awareness.

Practically, the framework can enhance the effectiveness of password strength meters, breach analysis platforms, and authentication systems by providing deeper insight into the true makeup of passwords. Especially those containing symbolic and numeric substitutions that traditional methods often overlook. Moreover, as password policies evolve to require stronger, more complex user credentials, the ability to parse such inputs semantically in real time becomes increasingly important. Integrating BFS-based tokenization into live feedback systems, password hygiene training tools, or adaptive password generators could support more secure authentication environments across industries.

Overall, the BFS-based tokenizer significantly improves the interpretability and completeness of password segmentation, particularly in symbol- and number-rich contexts, offering substantial gains over traditional greedy and symbol-oblivious methods.

D. Limitations

Notwithstanding its high performance, the approach presented above suffers from certain drawbacks. Firstly, the success of the tokenization algorithm depends on the completeness and accuracy of the used dictionary. Passwords that include highly disguised or extremely rare/obscure terms cannot be fully segmented. Such a problem can be mitigated by using additional sources to extend the dictionary, such as larger, more diverse corpora or domain-specific lexicons. Moreover, dynamic updates to dictionaries based on specific password patterns can be considered.

Furthermore, the BFS-based approach may face scalability issues when applied to extremely long passwords or passphrases, as there are many more possible substring combinations. Although this approach performs quite satisfactorily for commonly used passwords, its efficiency decreases with longer inputs. To overcome this drawback, optimization techniques (such as searching-space pruning, limiting the maximum substring length, or combining BFS with other algorithms, such as heuristics) may be applied.

V. Conclusion and Future Work

This paper presented a BFS-based tokenization framework for parsing complex password strings containing dictionary

words, numeric sequences, and symbol clusters. By exhaustively exploring all possible substrings and matching them against a comprehensive dictionary including specialized tokens for symbols and numbers, our approach significantly outperforms conventional tokenization methods in segmentation coverage and accuracy. Experimental evaluation on a dataset of 100,000 passwords demonstrated that the proposed method achieves higher coverage while maintaining practical runtime performance for offline password analysis.

This refined tokenization enables a more detailed understanding of how users integrate symbols and numbers into passwords, thereby enhancing password strength estimation and improving the effectiveness of dictionary-based password-cracking techniques. Furthermore, the implications of this research are twofold. First, it offers a more reliable and interpretable analysis of password composition patterns, which is essential for both offensive and defensive cybersecurity applications. Second, the framework's extensibility makes it a valuable foundation for future password analysis tools that require deeper semantic understanding and real-time capabilities.

For future work, various improvements to this technique could prove useful. Firstly, the vocabulary can be improved by adding more realistic words, such as those derived from user behavior, specific domain vocabularies, and complex patterns like multilevel leetspeak, thereby allowing the algorithm to better capture modern password construction. Secondly, the BFS algorithm can be enhanced with pruning methods that reduce computation time, making it feasible for time-sensitive tasks such as providing instant feedback during password creation. Lastly, the inclusion of semantic properties, such as part-of-speech tagging, can improve token understanding and, in turn, the accuracy of password evaluation. Overall, these modifications will greatly increase the methodology's usefulness in both academic and practical settings. A further line of future research may explore the effectiveness of generative AI methods in optimizing this technique.

REFERENCES

[1] R. Morris and K. Thompson, "Password security: A case history," *Communications of the ACM*, vol. 22, no. 11, pp. 594–597, 1979, doi: 10.1145/359168.359172.

[2] M. Khadka, "A systematic appraisal of multi-factor authentication mechanisms for cloud-based e-commerce platforms and their effect on data protection," *Journal of Emerging Cloud Technologies and Cross-Platform Integration Paradigms*, vol. 6, no. 12, pp. 12–21, 2022. [Online]. Available: https://hammingate.com/index.php/JECTCIP/article/view/2022-12-07/10

[3] F. Di Nocera, G. Tempestini, and M. Orsini, "Usable security: A systematic literature review," *Information*, vol. 14, no. 12, p. 641, 2023, doi: 10.3390/info14120641.

[4] M. Jubur, P. Shrestha, and N. Saxena, "An in-depth analysis of password managers and two-factor authentication tools," *ACM Computing Surveys*, vol. 57, no. 5, pp. 1–32, 2025, doi: 10.1145/3711117.

[5] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proceedings of the 12th ACM conference on Computer and communications security*, 2005, pp. 364–372, doi: 10.1145/1102120.1102168.

[6] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 538–552, doi: 10.1109/SP.2012.49.

[7] K. Reaz and G. Wunder, "Expectation entropy as a password strength metric," in *2022 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2022, pp. 1–2, doi: 10.1109/CNS56114.2022.9947259.

[8] D. Florêncio and C. Herley, "A large-scale study of web password habits," *ACM Transactions on the Web (TWEB)*, vol. 10, no. 4, pp. 1–26, 2016, doi: 10.1145/1242572.1242661.

[9] C. Castelluccia and M. Duermuth, "Adaptive password-strength meters from markov models," in *International Conference on Information Security and Cryptology*. Springer, 2012, pp. 1–15. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/06_3.pdf

[10] W. Melicher, "Modeling security weaknesses to enable practical run-time defenses," Ph.D. dissertation, Carnegie Mellon University, 2019. [Online]. Available: https://www.proquest.com/docview/2311052037?fromopenview=true&pq-origsite=gscholar&sourcetype=Dissertations%20&%20Theses

[11] H. V. Cook and L. J. Jensen, "A guide to dictionary-based text mining," in *Bioinformatics and Drug Discovery*, T. I. Oprea and C. G. Bologa, Eds. New York, NY, USA: Springer, 2019, pp. 73–89.

[12] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "Passgan: A deep learning approach for password guessing," in *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17*. Springer, 2019, pp. 217–237, doi: 10.1007/978-3-030-21568-2_11.

[13] S. Nam, S. Jeon, and J. Moon, "Generating optimized guessing candidates toward better password cracking from multi-dictionaries using relativistic gan," *Applied Sciences*, vol. 10, no. 20, p. 7306, 2020, doi: 10.3390/app10207306.

[14] A. Kuznetsov and D. A. Vyshemirsky, "One approach to solving tokenization problem for analysis of large-scale collections of user-defined passwords," *Bit Numerical Mathematics*, vol. 24, pp. 50–60, 2017, doi: 10.26583/BIT.2017.2.06.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT Press, 2009. [Online]. Available: https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/

[16] M. Kurant, A. Markopoulou, and P. Thiran, "On the bias of bfs (breadth first search)," in *2010 22nd International Teletraffic Congress (LTC 22)*. IEEE, 2010, pp. 1–8, doi: 10.1109/ITC.2010.5608727.

[17] A. Z. Ismaeel and I. M. Zebari, "Comparing traversal strategies: Depth-first search vs. breadth-first search in complex networks," *Asian Journal of Research in Computer Science*, vol. 18, no. 2, pp. 60–73, 2025, doi: 10.9734/ajrcos/2025/v18i2562.

[18] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *2009 30th IEEE symposium on security and privacy*. IEEE, 2009, pp. 391–405, doi: 10.1109/SP.2009.8.

[19] M. Weir, S. Aggarwal, M. Collins, and H. Stern, "Testing metrics for password creation policies by attacking large sets of revealed passwords," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 162–175, doi: 10.1145/1866307.1866327.

[20] W. Li and J. Zeng, "Leet usage and its effect on password security," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2130–2143, 2021, doi: 10.1109/ITC.2010.5608727.

[21] A. Kanta, I. Coisel, and M. Scanlon, "A novel dictionary generation methodology for contextual-based password cracking," *IEEE Access*, vol. 10, pp. 59 178–59 188, 2022, doi: 10.1109/ACCESS.2022.3179701.

[22] T. Yang and D. Wang, "Rankguess: Password guessing using adversarial ranking," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 682–700, doi: 10.1109/SP61157.2025.00040.

[23] A. N. Kuznetsov and D. A. Vyshemirsky, "One approach to solving the tokenization problem in the analysis of large collections of user passwords," *Information Technology Security*, vol. 24, no. 2, pp. 50–60, 2017. [Online]. Available: https://www.semanticscholar.org/reader/8be709b6d19625a61c5411a44db41bfb5a0e4ff8

[24] E. I. Tatli and E. Seker, "Password replacement patterns," in *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE, 2018, pp. 1–5, doi: 10.1109/CoDIT.2018.8394966.

[25] M. Zhang, J. Fan, and D. Chen, "Efficient dictionary matching by aho-corasick automata of truncated patterns," *International Journal of Computing Science and Mathematics*, vol. 7, no. 4, pp. 323–329, 2016, doi: 10.1504/IJCSM.2016.078738.

[26] M. Gibson, M. Conrad, and C. Maple, "Infinite alphabet passwords: A unified model for a class of authentication systems," in *2010 International Conference on Security and Cryptography (SECRYPT)*. IEEE, 2010, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/5741682

[27] J. Zhang, S. Liu, L. Gong, H. Zhang, Z. Huang, and H. Jiang, "Beqain: An effective and efficient identifier normalization approach with bert and the question answering system," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2597–2620, 2022, doi: 10.1109/TSE.2022.3227559.

[28] M. Gamon, "High-order sequence modeling for language learner error detection," in *Proceedings of the Sixth Workshop on Innovative Use of NLP for Building Educational Applications*, 2011, pp. 180–189, doi: 10.5555/2043132.2043154.

[29] F. Ariai, J. Mackenzie, and G. Demartini, "Natural language processing for the legal domain: A survey of tasks, datasets, models, and challenges," *ACM Computing Surveys*, vol. 58, no. 6, pp. 1–37, 2025, doi: 10.1145/3777009.

Salam Al-E'mari earned her Ph.D. in Cybersecurity from Universiti Sains Malaysia (USM) in 2022, Penang, Malaysia. She currently serves as chair of the Information Security Department and as an Assistant Professor in the Department of Information Security at the University of Petra (UoP). Dr. Al-E'mari has made significant contributions across domains such as Blockchain, Deep Learning, Network Security, and other computer science disciplines. Before her Ph.D., Dr. Al-E'mari completed her Bachelor's and Master's in Computer Science from Yarmouk University in Jordan. Dr. Al-E'mari is a member of the international IDPS research group (https://research.ju.edu.jo).

Mohammad Al Sawalhi is an Information Security graduate from the University of Petra, Jordan, currently serving as a Laboratory Supervisor in the Information Security Department. He possesses robust technical expertise in Linux systems and penetration testing, with a dedicated focus on offensive security and vulnerability assessment. Mohammad's professional background is highlighted by intensive training at the National Cyber Security Centre (NCSC) and the German Jordanian University (GJU). He is also further developing his practical skills through an ongoing internship at Satius Security and holds specialized certifications, including eJPTv2 and eWPT.

Yousef Sanjalawe received the Ph.D. degree in cloud computing and cybersecurity from Universiti Sains Malaysia, Penang, Malaysia, in 2020. He is an assistant professor in the IT Department at the King Abdullah II School for Information Technology, University of Jordan. Dr. Yousef is the Primary Investigator (PI) for the international IDPS research group (https://research.ju.edu.jo). He previously served as the Chair of the Cybersecurity Department and as an Assistant Professor with the Department of Cybersecurity, Faculty of Information Technology, American University of Madaba. Additionally, he has supervised Ph.D. students in various fields, including cybersecurity, cloud computing, IoT, fog computing, optimization, and AI.