Understanding User Passwords Through Parsing Tree

Ding Wang, Xuan Shan, Yuxuan Wu, and Chunfu Jia

***Abstract*—Passwords are today's dominant form of authentication, and password guessing is the most effective method for evaluating password strength. Most password guessing models (e.g., PCFG, Markov, and RFGuess) regard passwords as sequences composed of basic units (i.e., characters/segments), with the information flow being unidirectional (i.e., predicting the next unit based on the preceding units). However, modeling passwords in a single direction fails to capture users' password creation behavior that is actually impacted by the full password context. Through an in-depth analysis of real-world passwords, we reveal that users often create passwords around a central keyword, like common words, names, or dates, and then embellish them with numbers or symbols. Based on this observation, we, for the first time, attempt to parse passwords as trees. Unlike existing sequence models, trees can reveal the semantic connections within passwords and the logical thought processes users follow when creating passwords. For instance, in the password `iloveyou`, the basic units `i` and `you` are semantically dependent on the predicate `love`, forming a natural tree structure.**

We propose a trawling guessing model called PassTree and a targeted guessing model based on personally identifiable information (PII), named PassTree-PII. Our extensive experiments demonstrate the effectiveness of our models: (1) PassTree outperforms its leading counterparts by 0.38%-2.51% when guessing numbers are below 10^7; (2) PassTree-PII achieves a cracking rate comparable to the state-of-the-art RFGuess-PII proposed in USENIX Security'23, but operates significantly more efficiently, using only 0.87% of the memory and being 16.60 times faster. Our work provides a new perspective on understanding user passwords and demonstrates a feasible technical route of applying tree structures to password guessing.

***Index Terms*—Password guessing, parsing tree, trawling guessing, targeted guessing.**

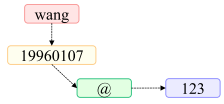
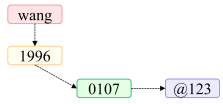
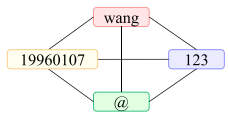
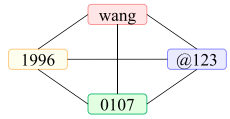
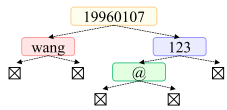
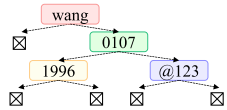
I. Introduction

PASSWORDS continue to be the dominant authentication mechanism due to their implementation simplicity [1], ease of modification [2], and cost-effectiveness [3]. This trend is expected to persist in the foreseeable future [3], [4], [5], [6]. Passwords need to be strong enough to resist guessing attacks while also being easy for users to memorize. The growing number of accounts has led to a tendency towards adopting easily guessed passwords or incorporating personally identifiable information (PII) into passwords [7], [8], [9], significantly compromising password security [10], [11], [12].

In the broad field of password security and cracking research, the vast majority of password guessing models primarily treat passwords as simple sequences of characters for their analyses and processing (as illustrated in Table II). For instance, Markov-based password guessing algorithm [13] assumes that the probability distribution of each character of a password depends on a specific length of sequence segment preceding it; FLA [14], based on recurrent neural networks, posits that the occurrence probability of any given character is influenced by all of its preceding characters. Although effective for deducing simple or common passwords, this sequence-centric method overlooks the inherent complexity and diversity in password creation, thereby neglecting some critical research perspectives.

The construction of passwords not only mirrors individual behavioral habits but is also shaped by a range of factors encompassing linguistics, psychology, and social engineering [15], [16], [17], [18]. Users often incorporate familiar elements from their lives, like names, dates, and personal interests, into their

TABLE I
The Differences Among Three Modeling Approaches†

Type	PCFG Segmentation	BPE Segmentation
Sequence		
Graph		
Tree		

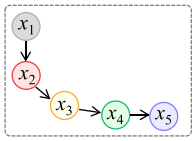
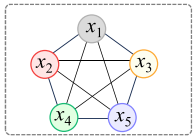
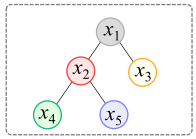
† To compare the differences among various modeling approaches, the given password `wang19960107@123` is assumed to be segmented into `wang 19960107 @123` using Probability Context-Free Grammars (PCFG) [25], and into `wang 1996 0107 @123` using Byte-Pair Encoding [26]. A sequential model treats the password as a unidirectional sequence. Graph-based modeling requires the addition of an undirected edge between each chunk, even if the chunks appear unrelated. A tree-based approach can capture the hierarchical structure of the password by selecting the most critical chunk as the root node (see Section III-B) with the remaining chunks constructed into subtrees in a recursive manner.

Received 29 May 2024; revised 29 October 2024; accepted 10 March 2025. Date of publication 18 March 2025; date of current version 4 September 2025. This work was supported by the National Natural Science Foundation of China under Grant 62172240 and Grant 62222208, in part by the Open Foundation of Key Laboratory of Cyberspace Security, Ministry of Education of China, and in part by the Henan Key Laboratory of Network Cryptography under Grant KLCS20240306. *(Xuan Shan and Yuxuan Wu contributed equally to this work.) (Corresponding author: Chunfu Jia.)*

The authors are with the College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China, also with the Key Laboratory of Cyberspace Security, Ministry of Education of China and Henan Key Laboratory of Network Cryptography, Tianjin 300350, China, and also with the Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin 300350, China (e-mail: wangding@nankai.edu.cn; shanxuan@mail.nankai.edu.cn; wuyuxuan@mail.nankai.edu.cn; cfjia@nankai.edu.cn).

Digital Object Identifier 10.1109/TDSC.2025.3552583

TABLE II
DIFFERENT TYPES OF PASSWORD MODELING APPROACHES AND CORRESPONDING TYPICAL LITERATURE†

	Illustration	Char-level literature	Chunk-level literature
Sequence		Narayanan *et al.* (2005) [13] Dürmuth *et al.* (2015) [34] Melicher *et al.* (2016) [14] Xu *et al.* (2017) [37] Fang *et al.* (2018) [38] Hita *et al.* (2019) [40] Pasquini *et al.* (2021) [42] Wang *et al.* (2023) [30]	Weir *et al.* (2009) [25] Veras *et al.* (2014) [35] Li *et al.* (2014) [36] Wang *et al.* (2016) [8] Zhang *et al.* (2018) [39] Liu *et al.* (2018) [41] Xu *et al.* (2021) [26] Cheng *et al.* (2021) [32]
Graph		Pasquini *et al.* (2020) [33]	Yu and Liao (2019) [43]
Tree		None	None

† The basic units that compose the password, denoted as x_i, can be either characters or chunks. We categorize all PCFG-based algorithms and their variants as *Chunk-level sequence structures* due to two main reasons: (1) The core of PCFG is to assign specific 'tags' to fragments of different character compositions/semantics, which closely mirrors the segmentation of chunks by Xu et al. [26] using the BPE algorithm, as both methods focusing on character segments; (2) Although PCFG can produce grammatical derivations such as $\mathsf{L}_4 \rightarrow$`Wang`, these derivations usually do not exceed two levels (e.g., $\mathsf{S} \rightarrow \mathsf{L}_4$, $\mathsf{L}_4 \rightarrow$`Wang`), essentially forming a dual sequence *rather than adhering to the potentially infinite growth characteristic of tree structures.*

passwords [8], [19]. The combination of these elements extends well beyond what a simple sequence of characters can convey. For instance, AlSabah et al. [16] observe that over 30% of passwords contain names, more than 5% incorporate two-digit birth years, and 4% include complete phone numbers. Moreover, the preferences exhibited by users from different cultural backgrounds in their password selection extend beyond the reach of sequence analysis [16], [20], [21]. For example, Chinese users adopt acronyms from the Pinyin version of Chinese sentences as passwords, like `brysjhhrhl` derived from a well-known poem "白日依山尽, 黄河入 海流". Similarly, Korean users, guided by their language's structure, may use unique combinations of initials, vowels, and consonants (e.g., `chosung`, `joongsung`, and `jongsung`), which differ markedly from Chinese Pinyin patterns in their passwords [20].

Given these observations, traditional password guessing models, which simplify passwords to character sequences, struggle with complex or unconventional passwords. This shortcoming has driven researchers to seek advanced models that better capture the logic of password generation and user behavior patterns. Passwords closely resemble human-generated texts in both structure and semantic information [22], [23], [24]. In 2013, Rao et al. [23] found that users subconsciously tend to use grammar that resembles natural language when creating long passwords (aka. passphrases): over 18% of user passwords contain grammatical structures (e.g., adjective+noun). Given the high similarity between passwords and natural language texts, coupled with the thorough previous research on the semantic analysis of passwords, we, *for the first time*, explore the feasibility of applying tree structure to password modeling. Building upon this framework, we develop efficient password-guessing algorithms leveraging the hierarchical tree structure. Tree structures, with their natural ability to reflect hierarchical structures and semantic relationships, offer a logically sound approach to emulating the human process of password generation.

A. Related Work

When creating passwords, users often choose simple, easy-to-remember strings, use personally identifiable information (PII), or reuse/modify existing passwords. This behavior makes passwords susceptible to various guessing attacks, including *trawling guessing* [14], [25], [26] and *targeted guessing* [8], [27], [28]. The former is guessing passwords for non-specific users, and the latter targeted guessing uses personal information to crack specific users' passwords.

Most current password-guessing models are sequential and can be categorized by their basic processing units: *Character-level* password models (e.g., Markov [29], FLA [14], and RFGuess [30]) primarily model relationships between individual characters. These models predict the next character based on the known prefix, generating passwords character by character. In contrast, *Chunk-level* model divides passwords into several chunks according to certain rules and explores the relationships between them. In 2009, Weir et al. [25] introduced Probabilistic Context-Free Grammars (PCFG), forming the basis of chunk-level models. PCFG divides passwords by character type, letters (L), digits (D), and special characters (S). For example, the password `wang123!!!` would be divided into $\mathsf{L}_4\mathsf{D}_3\mathsf{S}_3$ (subscripts indicate chunk lengths).

Subsequently, several improved Chunk-level models based on PCFG have emerged. At CCS'16, Wang et al. [8] proposed the TarGuess framework, suitable for attack scenarios

involving various types of personal information. At CCS'21, Xu et al. [26] introduced subword segmentation in password guessing, proposing a Byte-Pair Encoding (BPE) based chunk-level feature extraction algorithm and developing three chunk-based password guessing models. The BPE method divides passwords based on character continuity rather than character types, resulting in finer-grained segmentation compared to PCFG. Cheng et al. [31], [32] employ Point-wise Mutual Information (PMI) for segmentation and propose WordPCFG [32] and WordMarkov [31].[1]

Graph-based methods have so far been used only for password analysis, and have not been applied to password guessing. Pasquini et al. [33], for instance, hypothesize that each character in a password depends on the complete context (i.e., *all* the other characters), leading to the representation of passwords as undirected complete graphs. However, in a complete graph, not every "edge" is critical (i.e., some characters have little impact on other characters in the password), especially in long passwords. For example, the blank in `dearbo□k123`[2] is likely to be `o`, and this judgment does not *significantly* depend on the starting and ending characters of the password.

Table I illustrates the differences in password modeling methods. *Sequential models* capture only unidirectional information and fail to leverage the hierarchical structure of passwords. *Graph-based models*, on the other hand, require adding edges between every chunk, even with weak associations, leading to an exponential increase in edges as the number of chunks grows. This results in significant computational overhead and can obscure relevant feature extraction. In contrast, our tree-based methods naturally incorporate a hierarchical structure, making them ideal for representing passwords by organizing keywords, prefixes, and suffixes as root and child nodes (see Section II-A for details). This tree representation closely aligns with users' password construction habits and the inherent grammatical patterns in passwords.

B. Motivations

Tree structures excel at revealing the hierarchical organization of information. This makes them invaluable in fields such as natural language processing (NLP), decision tree development, and the construction of knowledge systems. The logic and thought processes inherent in creating passwords also echo tree structures' characteristics. For instance, users typically begin with a base (e.g., a common word, name, or date) and then add characters or modify existing ones to meet complexity restrictions, bypass blocklist detection, or improve password strength [44], [45], echoing a root-to-leaf progression. For another example, some grammatically structured, long passwords (e.g., `abigerbetterpassword` and `thereisnomored0ts`) [23] naturally conform to a tree structure. These long passwords are commonly utilized in brain wallets.[3]

Through an in-depth analysis of real-world passwords, we, for the first time, *model passwords as trees and explore password guessing algorithms based on the tree structure*. Generating guesses based on a tree structure is challenging because of two key issues: (1) *Node Definition*. Passwords lack the clear segmentation and grammar of natural languages. Defining the role and information of each tree node is critical. Yet, how existing segmentation methods are applied to construct tree structures and the effectiveness of such trees in enhancing guessing success rates remains unclear. (2) *Contextual Understanding*. The password parsing tree carries all detailed contextual relationships within the password, including node characteristics and their linkages. Incorrectly extracting context features may impair model performance, leading to either overly complex or overly simplistic password guesses. We primarily focus on improving the efficiency and success rate of password guessing, while also providing a new perspective on password security through the interpretability of the tree structure.

C. Contributions

We summarize our main contributions as follows:

- *Password parsing tree:* Drawing upon an in-depth characteristics analysis of real-world passwords, we, *for the first time*, parse passwords as trees. Compared to traditional sequence models, the password parsing tree reveals the semantic connections within passwords and the logical thought process users follow when creating passwords.
- *A new technical route for password guessing:* Based on the password parsing tree, we design two password guessing models: PassTree (for trawling guessing) and PassTree-PII (for targeted guessing based on PII). Furthermore, through comparative evaluation, we provide the optimal model configuration: the BPE password segmentation methods and stemlikeness calculation based on frequency.
- *Extensive experiments:* We evaluate the superiority of PassTree and PassTree-PII on 11 large-scale real-world datasets. PassTree has a clear advantage in small-scale guessing, improving by 0.38%-2.51% compared to its foremost counterparts. The cracking rate of PassTree-PII is close to that of RFGuess-PII [30] (with minor differences ranging from −0.04% to 0.30%), but it has a significant advantage in terms of memory and time consumption.

II. Password Parsing Tree

In this section, we analyze the characteristics of users' real-world passwords, demonstrating that users often revolve around a central keyword, which is then embellished with numbers or symbols. Based on this finding, we propose a method for parsing passwords as trees.

A. Analysis of Real-World Passwords

Nearly all websites enforce specific requirements on password content, strength, and complexity by employing blocklists, strength meters, and composition restrictions [47], [48]. In 1999, Adams et al. [49] reveal that in response to these requirements, users tend to comply with the minimum effort, employing tricks such as adding numbers or special characters

[1] Cheng et al. [31], [32] denote the segments extracted with PMI as "words", but their essence is the same as "chunks". To avoid confusion, we uniformly refer to them as "chunks".

[2] The symbol □ denotes an unknown character.

[3] In 2015, Castellucci [46] cracked a brain wallet containing 250 bitcoins, secured with the passphrase `how much wood could a woodchuck chuck if a wood chuck could chuck wood`

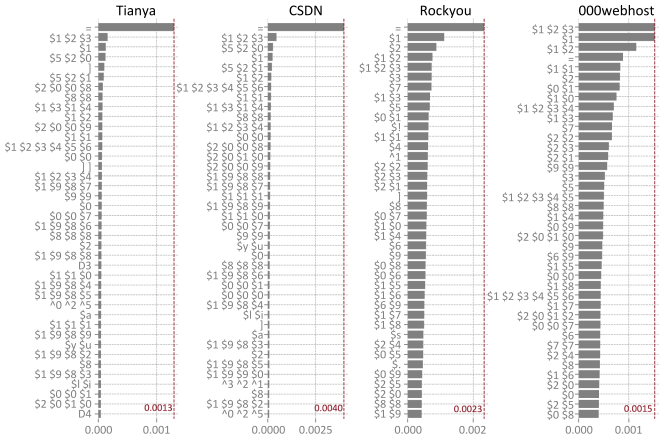


Fig. 1. Top-40 modifying rules and their probability distributions of each dataset. Symbols follow the conventions of hashcat's built-in rule set, where = means doing nothing, $ signifies appending a suffix, and ^ denotes prepending a prefix. Suffix appending and prefix prepending dominate the Top-40 modification rules.

at the beginning/end of passwords or incorporating keyboard patterns.

To gain a deep understanding of users' typical habits in password creation, we utilize the Password Analysis and Cracking Kit (PACK) as the analysis tool.[4] PACK takes an attack dictionary A and a target analyzed dataset B as input. For each password PW in dataset B, PACK identifies the most similar password in dictionary A through a spell correction algorithm and further adopts the reverse Levenshtein paths algorithm to automatically the transformation rule applied between the two passwords (e.g., appending `1` is noted as `$1`).[5] For instance, consider the password `dearbook123`. If the dictionary includes the term `dearbook`, PACK interprets the password as being generated by appending the numerical sequence `123` (denoted as `$123`) to the keyword `dearbook`.

Following the approach of Wang et al. [8], we create Chinese and English popular password sets with size 10^4 to serve as dictionaries for PACK. Then, we derive corresponding collections of cracking rules suited for Tianya, CSDN, Rockyou, and 000webhost, gaining the probability distributions of Top-40 rules (see Fig. 1) Our selected datasets are extensive, offering broad coverage of various languages and types of websites, and they are widely utilized in numerous password research studies (e.g., [26], [27], [30], [47]). These datasets capture the password construction habits of users from both English and Chinese language backgrounds, as well as those with differing levels of security awareness, ranging from ordinary social portal users to professional website developers. This diversity ensures the comprehensiveness and reliability of our analysis.

As illustrated in Fig. 1, suffix appending (indicated by symbol `$`) and prefix appending (indicated by symbol `^`) dominate the entire Top-40 modification rules and suffix appending is the most favored modification. This highlights a widespread strategy for enhancing password complexity to satisfy security needs: **users often create passwords around a central keyword, like common words, names, or dates, and then embellish them with numbers or symbols**. This insight forms the foundation for constructing PassTree.

Further analysis of Fig. 1 offers noteworthy observations. When it comes to suffix additions, numerals dominate, including sequences (e.g., `123`, `12`), specific years (e.g., `2009`, `1987`), and meaningful digits (e.g., `520`, `1314`). For the 000webhost, which has specific character composition requirements, the most favored suffixes are `123`, `1`, and `12`. *All of these observations indicate the vulnerability of passwords to rule-based dictionary attacks, as users tend to employ predictable rules to modify their passwords, thereby compromising the security of their passwords*.

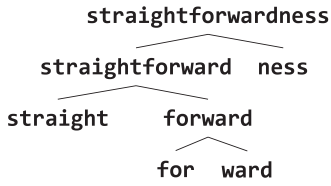


Fig. 2. The hierarchical segmentations of `straightforwardness`.

B. Parsing Process

Compared to natural language, passwords are unique primarily in their length and lack of clear lexical segmentation. In 1996, De Marcken [50] introduce a theory for unsupervised lexicon acquisition focusing on two central concepts: *composition* and *perturbation*. Composition suggests lexicons arise from existing ones (e.g., `joystick=joy+stick`), whereas Perturbation indicates that these combinations yield new meanings (`joystick` conveys more than just `joy` and `stick` combined). Expanding on this concept, Creutz and Lagus [51] proposed a recursive model for vocabulary extraction. This model posits that a word can be a single string or recursively constructed from two subwords, which, in turn, can be divided into smaller units. For instance, as illustrated in Fig. 2, `straightforwardness` is composed of the stem `straightforward` and the affix `ness`.

The *composition and perturbation theory* is also applicable to passwords: passwords can be seen as the concatenation (Composition) of chunks, but due to the presence of Perturbation, the overall strength of a password is not merely the sum of the strengths of its constituting chunks. Users often create passwords around a core idea, such as a common word, name, or date, adding digits, special characters, or other elements (see more details Section II-A). Therefore, passwords can be parsed as trees with stems and affixes (the selection of stems and affixes is described in Section III-B), and the parsing process can be formalized as context-free grammars, as follows:

- ⟨PW⟩→⟨STEM⟩;
- ⟨STEM⟩→⟨STEM⟩+⟨STR ⟩+⟨STEM⟩+⟨STR⟩|Null;
- ⟨STR⟩→ [\0x20-\0x7E]$^+$.

⟨PW⟩ symbolizes the password and acts as the starting variable. ⟨STEM⟩ represents the stem and can be derived through production rules (e.g., `008hotboy\`⟨STEM ⟩→`008\`⟨ STEM⟩ + `hotboy\`⟨STR ⟩+ Null, `008\`⟨ STEM⟩ → Null+`008\`⟨STR

[4]PACK toolkit: https://github.com/iphelix/pack

[5]A detailed explanation of rule representation can be found in https://hashcat.net/wiki/doku.php?id=rule_based_attack

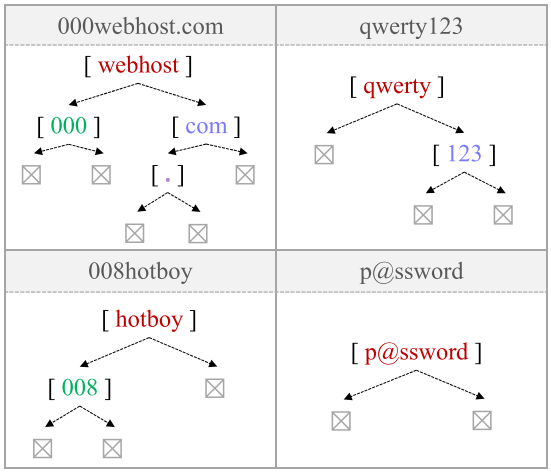


Fig. 3. Four examples of password parsing trees. ☒ represents the leaf node that cannot be further derived (i.e., Null). The selection of the root node is described in detail in Section III-B.

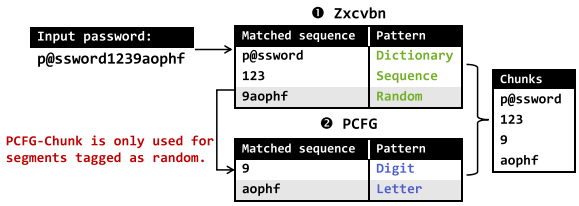


Fig. 4. The workflow of Zxcvbn-Chunk. Chunks tagged as "random" represent sequences that the dictionary doesn't match.

⟩+Null). ⟨ STR⟩ denotes a printable ASCII string, and Null refers to an empty string, both of them serve as a terminal symbol that cannot be further derived.

In Fig. 3, we display four examples of password parsing trees based on our proposed grammar rules. By conducting an in-order traversal of the password tree, we retrieve the original password. For example, traversing the tree for `000 webhost . com` yields ☒000☒webhost ☒.☒com☒, reconstructing the original password.

III. PassTree: A Password Guessing Model Based on Parsing Tree

In this section, we outline the foundation of our guessing model, focusing on password segmentation and root node selection. Next, we present two tree-based password guessing models: PassTree for trawling guessing and PassTree-PII for PII-based targeted guessing.

A. Password Segmentation Method

Most password security analysis focuses on individual characters, tracking their frequency of use, position preference, and inter-character probabilities. For example, uppercase characters tend to appear at the beginning of passwords [52]. However, growing evidence shows that character-level scrutiny neglects passwords' semantic and structural aspects [19], [23], [24], [35]. When creating passwords, people tend to use words (e.g., `password`) or phrases (e.g., `letmein`) with specific meanings, and sequences with particular patterns (e.g., keyboard sequences like `qwert`, or rhyming patterns like `4ever`) [19], naturally forming chunk structures within the passwords.

TABLE III
Hierarchy of Segmentations (Take `p@ssword1239` as an Example)

Level	Method	Example	Detailed description
#1	Char-level	`p @ s s 1 2 3 9`	Character-by-character slicing.
#2	PCFG-Chunk	`p @ ssw 1239`	Based on character type.
#3	BPE-Chunk	`p@ss 123 9`	Iteratively merge the most common tokens‡.
#4	BPE^{+}-Chunk	`p@ss 1239`	Aggregate consecutive chunks with the same type.
#5	Zxcvbn-Chunk	`p@ss123 9`	Based on dictionaries.

‡ Tokens are the basic units that encompass both characters and segments.

Probabilistic Context-Free Grammar (PCFG) proposed by Weir et al. [25] divides passwords into multiple segments based on character types. Building upon PCFG, enhanced guessing models are derived by incorporating additional semantic and PII tags (e.g., PCFGv4.1 [53] and Targuess-I [8]). Diverging from PCFG's character-based segmentation, Xu et al. [26] utilize Byte Pair Encoding (BPE) to segment passwords into multiple chunks, where a chunk is defined as *a sequence of frequently paired characters*. Essentially, chunks segmented by BPE and segments used in PCFG serve the same purpose and can be regarded as one concept.

Xu et al.'s [26] password segmentation approach, relying on BPE, necessitates manually selecting an average chunk length (denoted as Avg_len) as the termination condition, and the parameter setting directly affects the guessing success rate. Specifically, the optimal Avg_len is 1.8 for Chunk-level Markov, and 4.5 for Chunk-level PCFG, while no universal Avg_Len setting works well for Chunk-level FLA across all datasets.

Considering that BPE-based methods require training on different datasets and their parameters are not universal, we design a dictionary-based segmentation approach using Zxcvbn [54] that does not require additional training. (see Fig. 4). The dictionary-based segmentation algorithm identifies semantic units within passwords using a preset dictionary, effectively recognizing standard vocabulary, slang. Moreover, the dictionary approach not only reduces errors in understanding password semantics but also quickly adapts to emerging or variant vocabularies through lexicon updates.

Furthermore, based on the grain size of the segmentation and the required "materials", we categorize password segmentation algorithms into five levels. Each level denotes distinct strategies and the resources relied upon for computation, ranging from basic character-level segmentation to more complex semantic analysis (see Table III for detailed information):

- *Level-1:* Char-level. The most refined of segmentation, treating each character as the basic unit of the password, including 96 printable ASCII characters.
- *Level-2:* PCFG-Chunk. Segments passwords by character types, ignoring relationships between characters. Thus, it struggles to deal with sequences with diversity yet regular patterns like keyboard sequences (e.g., `1q2w3e`) or leet (e.g., `p@ss`).

TABLE IV
THE MAGNITUDE OF CHUNK COLLECTION WITH VARIOUS CHUNK METHODS†

Dataset	Tianya	CSDN	Rockyou	000webhost
Char-level	96	96	96	96
PCFG-Chunk	9,377,935	3,059,678	8,379,191	6,586,810
BPE-Chunk	14,857	20,078	13,047	41,786
BPE$^+$-Chunk	9,354,524	3,041,877	7,994,924	6,080,337
Zxcvbn-Chunk	4,865,352	2,054,088	5,626,514	7,424,101

† Avg_Len. of the BPE-Chunk is 3.0.

- *Level-3:* BPE-Chunk. Segmentation is based on the closeness of token associations. BPE is a representative algorithm. The PMI-based segmentation proposed by Cheng et al. [32] also falls into this category. Such algorithms require parameter configuration.
- *Level-4:* BPE$^+$-Chunk. A supplement to the BPE-Chunk, partially solving the problem of incomplete segmentation and the loss of latent patterns due to inappropriate parameter settings. It iteratively merges all consecutive chunks with the same type (e.g., `1999 10 15`→`19991015`).
- *Level-5:* Zxcvbn-Chunk. Segment passwords using a dictionary to maximize the retention of semantic information, and no parameter settings are required.

To clearly compare the granularity across various segmentation methods, we record the sizes of chunk collections obtained from segmentations at five levels across Tianya, CSDN, Rockyou, and 000webhost. As illustrated in Table IV, the PCFG-Chunk segmentation method typically generates the largest set of chunks. The size of chunk collection obtained by BPE$^+$-Chunk method is larger than that of BPE-Chunk method.

B. Root Node Selection

The stem forms the core of a word, serving as the foundation for morphological changes and derivations. The stem inherently possesses the capability to form words and usually remains constant despite any changes in the word's form. For instance, the stem `play` is consistent across variations such as `player`, `played`, and `playing`. Regarding the selection of stems, there's an intuitive principle: *stems are usually longer than their accompanying affixes* [51]. This is because the stem is the fundamental part that carries the core meaning of a word, while affixes are shorter letter combinations attached to the stem to modify the meaning or create a new word. Nevertheless, this principle is not without exceptions, particularly in certain complex compound words where the affix may approach or equal the stem in length.

The stem of a password determines the structure of the entire password tree, as the stem of a string is the root node of each subtree it corresponds to. To evaluate how likely it is for a chunk within a password to be the considered stem, we propose *stemlikeness* as a metric. We heuristically select two key features of password chunks to calculate stemlikeness:

- *Length:* In recognizing that longer words typically convey more specific and rich information in natural language, we consider length a crucial metric for evaluating the potential of a chunk to become a password stem. This is quantified using the formula $Stemlikeness(chunk)$=Sigmoid$(chunk.length-3)$.
- *Frequency:* The frequency of chunks in passwords mirrors user preferences. We can measure the likelihood of a chunk becoming a password stem by $Stemlikeness(chunk) = chunk.freqency$.

It should be noted that we adopt the Sigmoid function (i.e., $\sigma(x)=\frac{1}{1+e^{-x}}$) to normalize length values to [0,1]. This function, being strictly monotonic, ensures its output gradually approaches 1 as $chunk.length$ increases, but the growth rate slows down, preventing the overemphasis on the priority due to excessively long chunks. To identify meaningful stems, typically not very short, we've incorporated a length threshold in our metric (i.e., $chunk.length-3$).

Additionally, we assess stemlikeness by multiplying the normalized length value by the chunk's frequency: $Stemlikeness(chunk)$=Sigmoid$(chunk.length-3) \times chunk.frequency$. This formula favors chunks that are not only of suitable length but also commonly found in passwords, identifying potential stems like `123456`, `iloveyou`, and `angel`.

Algorithm 1: Password Tree Construction.

```
function BuildTree(ChunkList)
  if ChunkList is empty then
    returnNull
  end if
  StemIndex ← FindStemIndex(ChunkList)
    /* The index of the chunk with maximum
    stemlikeness.*/
  Root ← new TreeNode(ChunkList[StemIndex])
  Root.left← BuildTree(ChunkList[0:StemIndex])
    /* Recursively constructs the left subtree.*/
  Root_Node.right←
  BuildTree(ChunkList[StemIndex+1:END])
    /* Recursively constructs the right subtree.*/
  returnRoot
end Function
```

C. PassTree: A Trawling Guessing Model

Password Tree Construction: Constructing the pass word tree is vital for training. As described in Section II-B, the workflow of building the password tree is essentially an iterative construction of a binary tree. We begin with the list of password chunks, selecting the stem (i.e., the chunk with the highest stemlikeness) as the root node. The left and right subtrees are then constructed recursively by applying this selection process to the chunk lists adjacent to the current stem. This method parses a password as a tree that carries all detailed contextual relationships within the password, including node characteristics and their linkages. The details of constructing password tree are provided in Algorithm 1.

Tree Templates Extraction: The height of the password tree and the nodes it contains reveal the complexity of passwords. We obtain the structural characteristics of the tree by level-order traversal. More specifically, we traverse and record the chunk

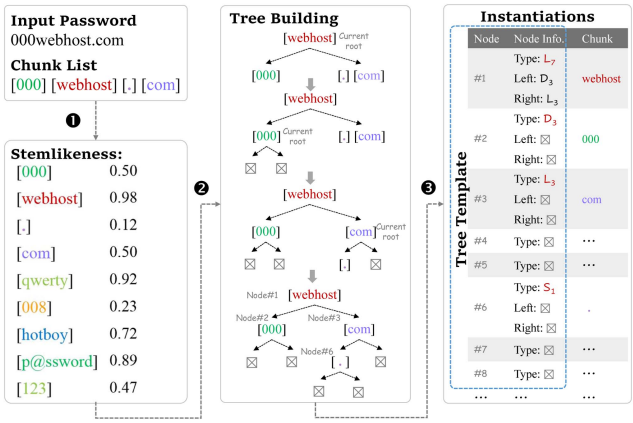


Fig. 5. The training process of PassTree. During tree construction, the chunk with the highest stemlikeness is selected as the root node. Each node is numbered in a hierarchical order, and chunk distribution is documented according to the node type and its children's types.

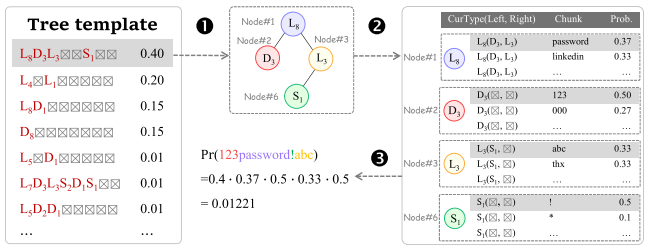


Fig. 6. An illustration of generating passwords by PassTree. The tree template stores the nodes in a hierarchical order, with ⊠ representing a leaf node. The content for a node's chunk is determined by its type and the types of its two child nodes.

type each node represents in level-order, marking leaf nodes as Null (notated as ⊠ in Figs. 5 and 6) We follow the same method as Xu et al. [26] to represent password chunks:

- L_n, U_n, D_n, S_n denote password chunks of lowercase letters, uppercase letters, digits, and symbols of length n, respectively.
- DM_n, TM_n, FM_n indicate password chunks combining two, three, and four character types of length n, correspondingly.

Taking the password `000` `webhost` `.` `com` forming the tree shown in Fig. 5 as an example, its tree template is $\mathsf{L}_7\mathsf{D}_3\mathsf{L}_3$ ⊠ ⊠S_1⊠. To enhance the interconnectivity between nodes in the password tree, the distribution of chunks for each node is influenced by its type and the types of its two child nodes, as detailed in Fig. 5.

Generation: In the generation's initial phase, we select the tree template with the highest probability. When selecting the chunk for a specific node, except for the type of the current node (e.g., L_3), the types of the left and right child nodes are also taken into account. Furthermore, we allocate an update index (i.e., pivot) to each password tree, to determine which node is ready to be updated. We place each password tree into a priority queue and repeatedly pop the tree with the highest probability. By performing an in-order traversal, we obtain the complete guessed password. The detailed process is illustrated in Algorithm 2, and Fig. 6 provides a clear visual representation of generating the guessed password `123password!abc`.

Algorithm 2: The Generation Process of PassTree.

```
function GENERATION(Trees,Nodes)
    /* Trees = [(Template_1,tPro_1), ... (Template_n, tPro_n)]*/
    /* Nodes record the distribution of different types of chunks within each node.*/
  Initialize an empty queue Q.
  for (Template,tPro) in Trees do
    /* Traverse each node of Template.*/
    while index <len(Template) do
      Node_i=Template[index]
      tNode_i=GetType(Node_i)
      /* Get the types of left and right child nodes of Node_i.*/
      tLeft,tRight=GetChildNodeType(Template,Node_i)
      /* Get the highest-probability chunk meeting all constraints, and 0 denotes top priority among nodes of the same class.*/
      Template[index]=GetChunk(tNode_i,tLeft, tRight,0)
      Template[index].piror=0
    end while
    Q.enqueue(Template,0)
  end for
  Initialize an empty guessing passwords list G.
  while not Q.isEmpty() do
    Tree,pivot=Q.dequeue()
    /* Guesses are obtained by performing an in-order traversal.*/
    G.append(InOrderTraversal(Tree))
    /* The pivot value helps determine which Node in the Tree needs to be updated.*/
    while pivot <len(Tree) do
      Tree'=DeepCopy(Tree)
      tNode= GetType(Tree'[pivot])
      Tree'[pivot].piror+=1
      prior= Tree'[pivot].prior
      tLeft,tRight=GetChildNodeType(Tree',tNode)
      /* Each node's chunk will be updated in accordance with its priority value prior.*/
      Tree'[pivot]=GetChunk(tNode,tLeft, tRight,prior)
      Q.enqueue(Tree',pivot)
      pivot+=1
    end while
  end while
  return G
end Function
```

D. PassTree-PII: A Targeted Guessing Model

Nearly all trawling password guessing algorithms can be adjusted into targeted guessing algorithms based on PII. For example, the original PCFG [25] and the PCFG-based TarGuess-I [8], as well as the original Markov [13] and Targeted-Markov [47].

TABLE V
BASIC INFORMATION OF THE PASSWORD DATASETS LEAKED FROM 11 WEB SERVICES ("PWS" REPRESENTS PASSWORDS)†

Dataset	Language	When leaked	Original PWs	After cleaning	Removed	Service	With PII
Sohu	Chinese	Oct. 2015	14,755,046	14,607,567	1.00%	Portal	✗
Tianya	Chinese	Dec. 2011	30,816,592	30,807,530	0.03%	Social forum	✗
Dodonew	Chinese	Dec. 2011	16,282,286	16,026,270	1.57%	E-commerce & Gaming	✗
Rockyou	English	Dec. 2009	32,603,387	32,451,348	0.47%	Social forum	✗
000Webhost	English	Oct. 2015	15,299,907	15,183,445	0.76%	Web hosting	✗
LinkedIn	English	Jan. 2012	54,656,615	54,534,564	0.23%	Job hunting	✗
12306-PII	Chinese	Dec. 2014	129,303	129,303	0%	Train ticketing	✓
CSDN-PII	Chinese	Dec. 2011	77,216	77,216	0%	Programmer	✓
Rootkit-PII	English	Feb. 2011	69,330	69,090	0.01%	Hacker forum	✓
ClixSense-PII	English	Sep. 2016	2,222,045	2,222,045	0%	Paid task platform	✓
COMB-PII	Mixed	Feb. 2021	863,239	863,239	0%	Mixed	✓

† We remove passwords that contain non-ASCII characters or longer than 32.

The essence of these adaptations involves the integration of PII as fundamental units, as detailed in [8], [47]. Next, we provide a comprehensive explanation of our PassTree-PII.

The core of PII-based guessing algorithms lies in identifying and matching PII in passwords. Existing approaches are generally classified into three categories: (1) Length-based matching methods [9] focus on fields at least three characters long, using the nearest-longest matching strategy (e.g., `helloalice816` was tagged as `hello`N_5B_3); (2) Type-based matching methods [8] categorize PII and its variants more finely, still adopting the nearest-longest strategy for precise labeling (e.g., `zhang01021993` could be tagged as either N_3B_2 or A_1B_5). Notably, in length-based matching [9], the numeric subscripts denote length, while in type-based matching [8], they represent the variant type of the PII field. (3) Approximate optimal matching methods that minimize information entropy [30], using frequency statistics to disambiguate labels.

We choose the matching method that minimizes information entropy (i.e., Category-3) for two primary reasons: firstly, in length-based matching, the setting of the length threshold influences the results, leading to significant shortcomings in generating password guesses [8]. Secondly, type-based matching can produce multiple potential labels for the same password due to the prioritization in label matching. The approach of minimizing information entropy addresses these challenges effectively.

Building on the PassTree algorithm, we extract the PII and its variants from passwords as outlined in [30] and assign corresponding tags. Personal information tags are considered the same level as L, D, S. Following this, similar to the processing flow of the PassTree algorithm for trawling guessing, we proceed with algorithm training and the generation of guessed passwords.

IV. EXPERIMENT

In this section, we introduce the datasets used for the experiment, the detailed experimental setup, and the analysis of the results of both trawling and PII-based targeted guessing experiments. Additionally, we thoroughly investigate how the tree structure and the size of the training set influence PassTree's performance.

A. Datasets and Ethical Consideration

Datasets: We evaluate PassTree and leading guessing models on 11 large-scale real-world datasets (refer to Table V), a total of 167.75 million (M) passwords. Following Wang et al.'s methodology [19], we exclude non-ASCII and passwords exceeding 32 characters in length, as they are unlikely generated by humans. The datasets containing personally identifiable information (PII) are marked with the suffix "-PII", and these datasets are widely utilized in related research of targeted password guessing (e.g., [8], [30], [55], [56]). The PII we use in the following evaluation includes email addresses, usernames, names, birthdays, and phone numbers.

Ethical considerations: Although the datasets used in our study have been widely utilized in previous password research [8], [26], [30], it is crucial to acknowledge that these datasets contain sensitive information. To address potential privacy concerns, we only disclose aggregated statistical data and ensure individual account information remains confidential. To further protect against data breaches, these datasets are processed on computers that are not connected to the Internet. Despite the possibility that these datasets might have been previously exploited for nefarious purposes, our research aims to help security administrators and users evaluate password security and discourage the use of weak passwords. The public availability of our datasets from multiple online sources also guarantees the replicability of our results.

B. Experimental Setup

We select advanced and representative password guessing models corresponding to the respective attack scenarios for comparison with PassTree (trawling guessing) and PassTree-PII (PII-based targeted guessing). It is important to note that password guessing can be categorized according to attack scenarios. Some advanced password guessing models are not selected because they do not fit the attack scenarios addressed in this paper (e.g., Pass2Edit [27] and POINTERGUESS [28] are designed for attacks that exploit users' existing passwords).

Experimental setup for trawling guessing: Based on the comparative analysis in Section IV-C, we select the most favorable segmentation method and stemlikeness calculation method for PassTree. This involve utilizing the Byte-Pair-Encoding (BPE)

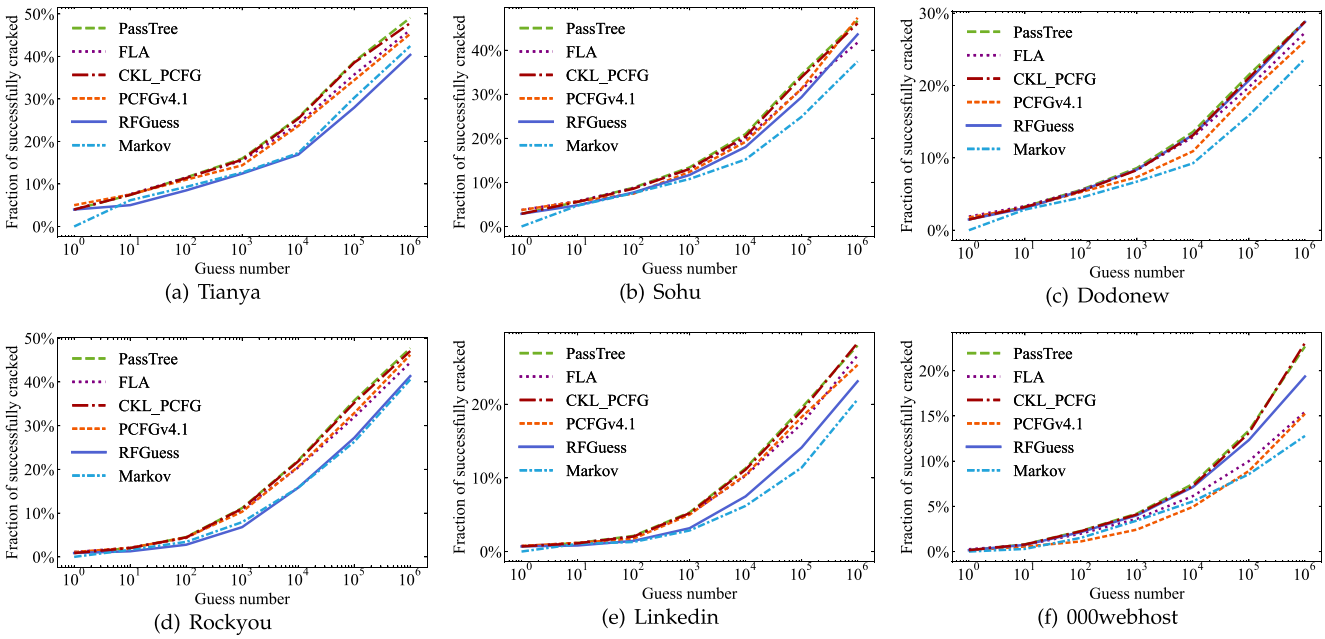


Fig. 7. Experiment results of our PassTree and other guessing models (i.e., FLA [14], CKL_PCFG [26], PCFGv4.1 [53], RFGuess [30] and Markov [29]). PassTree generally has an advantage when guess number $\leq 10^6$.

algorithm to tokenize passwords and calculating stemlikeness based on frequency of occurrence. Furthermore, for the reference evaluation algorithms, we follow either the recommended parameter configurations by the original authors or those validated through experiments to ensure fair comparisons:

- Xu et al. [26] suggest that the CKL_PCFG performs optimally when the average length of the chunks Avg_len is 4.5. We adopt the same termination condition as recommended.
- For FLA [14], we adopt the recommended network structure, which consists of three layers of LSTM with 200 neurons per layer and two layers of fully connected layers.
- RFGuess [30] is configured with 30 decision trees, as recommended.
- As for the Markov algorithm, we follow the research findings by Wang et al. [19] and set the order to 4 because this parameter configuration yields optimal results for small-scale guessed numbers ($\leq 10^6$).

We extract a total of 10 million passwords from the dataset for training, and then randomly select 100,000 passwords from the remaining dataset for testing.

Experimental setup for targeted guessing: Similar to trawling guessing, we calculate stemlikeness based on frequency of occurrence for PassTree-PII. Due to the relatively small data size in this scenario, the BPE algorithm is unable to reach the predetermined termination condition. Therefore, we choose to use the Zxcvbn-Chunk partition method. The targeted password guessing algorithms based on personally identifiable information (PII) used for comparison are Targuess-I [8] and RFGuess-PII [30]. Targuess-I is currently the most influential algorithm in the field of targeted guessing, serving as the foundation for many PII-based targeted guessing algorithms. RFGuess-PII is a recently proposed random forest-based guessing algorithm that represents the state-of-the-art in PII-based targeted password guessing algorithms. Due to the low efficiency of RFGuess-PII when the data scale is large, we partition all datasets except Clixsense-PII into training and test sets at a 4:1 ratio. For the Clixsense-PII dataset, we extract five million passwords for training and 100,000 passwords for testing.

TABLE VI
THE POPORTION OF PASSWORD COMPLEXITIES ACROSS DIFFERENT DATASETS†

Dataset	1class8	1class16	3class12	4class8
000webhost	86.09%	6.94%	8.55%	1.57%
Linkedin	70.32%	0.64%	1.08%	1.05%
Rockyou	50.49%	0.76%	0.64%	0.16%
Dodonew	69.29%	0.49%	0.43%	0.05%
Sohu	53.55%	0.41%	0.40%	0.06%
Tianya	50.63%	0.61%	0.70%	0.06%

† Background color indicates the dataset with the highest proportion. Here 1class8 indicates that the password contains at least one type of character and has a minimum length of eight. The other categories follow the same logic.

C. Analysis of Trawling Guessing

The cracking rate curves for each model can be seen in Fig. 7 and detailed cracking rate values are summarized in Table VII. PassTree exhibits a consistent advantage when the number of guesses is small ($\leq 10^6$). Compared to CKL_PCFG [26], PassTree shows an average performance improvement of 1.25% with only 10 guesses, 2.51% improvement with 100 guesses, 2.21% improvement with 10^3 and 10^4 guesses, 1.71% improvement with 10^5 guesses, and still achieves a 0.38% improvement with 10^6 guesses. It is generally observed that PassTree

TABLE VII
CRACKING RATES OF DIFFERENT ALGORITHMS (TRAINING SET SIZE = 10^6)†

Dataset	Algorithm	10^2	10^3	10^4	10^5	10^6
Tianya	PassTree	11.55%	15.99%	25.62%	38.77%	49.09%
	CKL_PCFG [26]	11.39%	15.72%	25.35%	38.54%	47.85%
	PCFGv4.1 [53]	11.01%	14.39%	23.63%	34.42%	45.32%
	RFGuess [30]	8.49%	12.45%	16.90%	27.98%	40.37%
	Markov [13]	9.32%	12.69%	17.27%	30.29%	42.45%
	FLA [14]	11.50%	15.48%	24.14%	35.74%	46.22%
Sohu	PassTree	8.82%	13.42%	21.00%	34.52%	46.67%
	CKL_PCFG [26]	8.58%	13.07%	20.44%	33.91%	46.12%
	PCFGv4.1 [53]	7.45%	12.39%	19.23%	31.30%	47.37%
	RFGuess [30]	7.73%	11.70%	18.06%	29.37%	43.60%
	Markov [13]	7.58%	10.79%	15.22%	24.91%	37.47%
	FLA [14]	8.73%	13.06%	19.97%	31.19%	41.74%
Dodonew	PassTree	5.57%	8.60%	13.58%	21.49%	28.84%
	CKL_PCFG [26]	5.39%	8.31%	13.11%	21.06%	28.82%
	PCFGv4.1 [53]	5.27%	7.35%	10.91%	18.94%	26.14%
	RFGuess [30]	5.44%	8.39%	13.17%	20.71%	28.78%
	Markov [13]	4.50%	6.72%	9.25%	15.86%	23.70%
	FLA [14]	5.51%	8.35%	12.86%	19.78%	27.31%
Rockyou	PassTree	4.53%	11.18%	22.04%	35.74%	47.71%
	CKL_PCFG [26]	4.45%	11.04%	21.85%	35.21%	47.05%
	PCFGv4.1 [53]	4.42%	10.33%	20.61%	32.87%	46.20%
	RFGuess [30]	2.78%	6.81%	15.83%	27.21%	41.33%
	Markov [13]	3.43%	7.95%	15.82%	26.30%	40.54%
	FLA [14]	4.34%	10.79%	20.45%	31.95%	44.43%
Linkedin	PassTree	2.13%	5.31%	11.31%	19.51%	27.99%
	CKL_PCFG [26]	2.04%	5.21%	11.12%	19.09%	28.34%
	PCFGv4.1 [53]	1.75%	5.01%	10.41%	18.26%	25.35%
	RFGuess [30]	1.45%	3.17%	7.49%	14.12%	23.12%
	Markov [13]	1.30%	2.86%	6.20%	11.39%	20.66%
	FLA [14]	2.03%	5.08%	10.30%	17.34%	26.63%
000webhost	PassTree	2.28%	4.20%	7.48%	13.40%	22.68%
	CKL_PCFG [26]	2.25%	4.11%	7.24%	13.12%	23.08%
	PCFGv4.1 [53]	1.12%	2.43%	4.95%	8.92%	15.24%
	RFGuess [30]	2.15%	4.01%	7.14%	12.34%	19.36%
	Markov [13]	1.61%	2.93%	4.96%	7.83%	12.50%
	FLA [14]	1.98%	3.62%	6.14%	10.01%	15.45%

† Background color indicates the model with the highest cracking rate.

TABLE VIII
IMPACT OF STEMLIKENESS CALCULATION METHODS (M REPRESENTS A MILLION)†

Dataset	Stem-likeness	10^4	10^5	10^6	Dataset	Stem-likeness	10^4	10^5	10^6
Tianya 1 M	Freq.	25.00%	34.09%	44.39%	Rockyou 1 M	Freq.	21.50%	29.95%	39.73%
	Len.	24.99%	34.32%	44.28%		Len.	21.49%	29.68%	39.64%
	Freq.&Len.	25.00%	33.88%	44.35%		Freq.&Len.	21.50%	29.87%	39.64%
Tianya 5 M	Freq.	25.73%	38.09%	47.75%	Rockyou 5 M	Freq.	22.12%	35.20%	46.17%
	Len.	25.73%	38.09%	47.56%		Len.	22.12%	35.20%	45.90%
	Freq&Len.	25.73%	38.09%	47.76%		Freq&Len.	22.12%	35.20%	46.01%
Tianya 10 M	Freq.	25.62%	38.77%	49.09%	Rockyou 10 M	Freq.	22.04%	35.74%	47.71%
	Len.	25.62%	38.77%	49.00%		Len.	22.04%	35.73%	47.60%
	Freq.&Len.	25.62%	38.77%	49.06%		Freq.&Len.	22.04%	35.74%	47.69%

† Background color indicates the model with the highest cracking rate.

generally performs better in English scenarios compared to Chinese scenarios. *The outstanding performance of PassTree in small-scale guessing, especially with 10^4 guesses or less, indicates its suitability for online guessing scenarios with strict limitations on the number of allowed guesses.*

Considering the variations in cracking rates across different datasets, we adopt the method used in [14] to quantify the complexity of passwords in various datasets. We categorize the passwords based on the number of character types and their lengths, dividing them into four classes: 1class8, 1class16, 3class12, and 4class8 (where xclassy indicates that the password contains at least x types of characters and has a minimum length of y). As shown in Table VI, the model achieves the lowest cracking rates on 000webhost with the most complex passwords, while it attains higher cracking rates on datasets with relatively simpler password compositions (e.g., Rockyou, Tianya). Additionally, users' cultural backgrounds and security awareness, as well as websites' password policies, all influence password strength [19].

1) Analysis of Password Tree: The structure of the password tree depends on the password segmentation method and the metrics used for calculating stemlikeness. Here, comparative experiments are conducted to identify the optimal segmentation method and stemlikeness calculation metrics, thereby determining the optimal approach for constructing the password tree. Additionally, an in-depth analysis of the passwords cracked by PassTree is conducted from the perspective of the password tree.

Impact of segmentation method: We combine the Rockyou and Tianya datasets to create a mixed dataset (i.e., "Mixed" in Figs. 9 and 8). Then we extract 10 million passwords from Rockyou, Tianya, and the mixed dataset separately as the training set, and 100,000 passwords from the remaining data as the test set. The results (shown in Fig. 8) indicate that BPE-Chunk yields significantly better results compared to the other methods. When the guess number $\leq 10^4$, the results for different segmentation methods are quite similar. However, as the number of guesses increases, the advantages of BPE-Chunk and Zxcvbn-Chunk become more pronounced, while PCFG-Chunk performs the worst. BPE$^+$-Chunk exhibits notable differences in performance compared to other partitioning methods when the guess number $\leq 10^4$. One possible reason is that BPE$^+$-Chunk merges two consecutive chunks of the same type, which disrupts the original features of BPE-Chunk. Consequently, the frequency of common short chunks decreases, making it difficult to generate guesses with corresponding chunks under small-scale guessing scenarios.

Impact of metrics for stemlikeness: Stemlikeness can be considered as the weight of chunks, and we compare the cracking rates of the PassTree using different metrics for stemlikeness to identify the optimal metric. The results in Table VIII indicate that the choice of different metrics has minimal impact on the results, and there is almost no effect when the number of guesses is less than 10^4. When the number of guesses exceeds 10^4, the impact of stemlikeness on the cracking rate is generally within 0.1%. However, it is still observable that using frequency as stemlikeness is most advantageous for our PassTree.

Analysis of root nodes: In Section II-A, it is mentioned that users construct passwords by adding prefixes or suffixes to keywords. The hierarchical structure of the password tree positions these keywords at the root nodes. Therefore, we take an analysis of the content at the root nodes of the password tree. More specifically, password trees are constructed from the

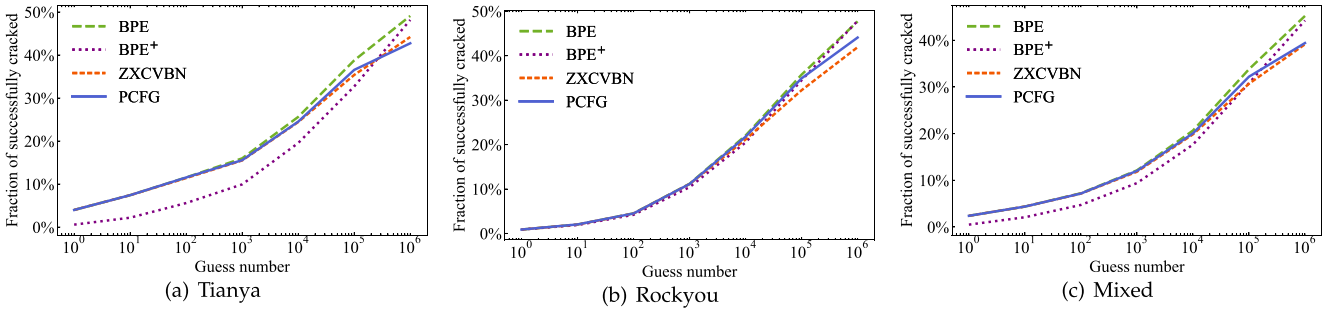


Fig. 8. Impact of four segmentation methods (i.e., BPE [26], BPE$^+$, ZXCVBN [54] and PCFG [25]) on our PassTree.

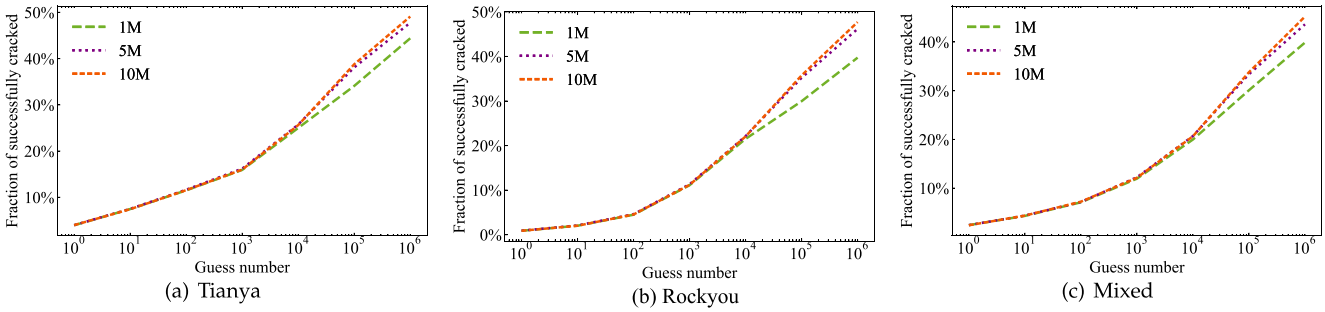


Fig. 9. Impact of the training set size (i.e., one million, five miilion and ten million) on our PassTree. M represents a million.

TABLE IX
TOP-10 COMMON CHUNKS OF THE ROOT NODE†

Rank	ALL	L	D	S	U	DM
1	123456	wang	123456	#$%^	COM	a123456
2	123456789	password	123456789	******	NULL	abc123
3	111111	zhang	111111	.....	MA	123456a
4	198	ita	198	??????	NA	a321654
5	000000	ing	000000	````	PI	123qwe
6	123123	iloveyou	123123	-----	BABY	123abc
7	5201314	ie	5201314	********	IN	q123456
8	1986	man	1986	++++++	TA	asd123
9	1984	love	1984	#$%&	BU	1q2w3e4r
10	1985	chen	1985	....	LOVE	1qaz2wsx
Proportion	100%	31.90%	46.88%	0.15%	0.67%	19.36%

All indicates no consideration of type restrictions.
† L,D,S,U represent chunks consisting of lowercase letters, digits, special characters, and uppercase letters, respectively. Chunks contain two types are denoted by DM. Considering that single characters lack semantic meaning, only chunks of length greater than one are displayed.

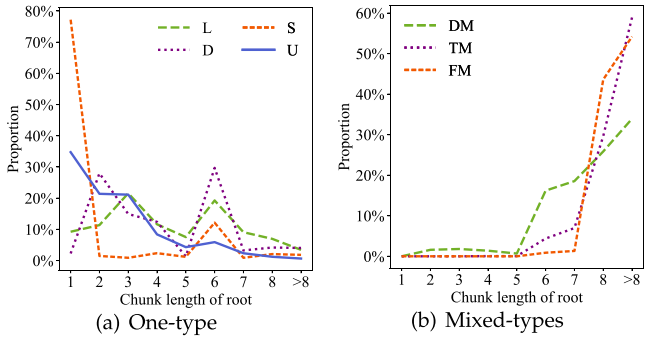


Fig. 10. The length distribution of chunks corresponding to the root node in password trees.

passwords cracked by our PassTree, followed by an enumeration of the different chunks at the root nodes.

As shown in Table IX, the common types of chunks for root nodes are digits and lowercase letters. The root nodes are typically consisted of simple numerical sequences (e.g., `123456`, `111111`), years (e.g., `1986`, `1984`), and semantically meaningful numbers (e.g., `5201314`). Among the letter-based root nodes, surnames (e.g., `wang`, `zhang`) and words (e.g., `man`, `love`) are most popular. Additionally, users have a preference for combining simple numerical and alphabetical sequences (e.g., `abc123`, `123qwe`) at the root node.

We analyze the length distribution of chunks corresponding to the root node. As shown in Fig. 10, chunks made up of lowercase letters and digits show a similar length distribution, with most lengths being less than six. Chunks made up of uppercase letters and special characters tend to be shorter in length. One possible reason is the fine-grained segmentation achieved by BPE-based approach. In contrast, chunks of mixed types are predominantly combinations of single-type chunks (see Table IX) and generally have longer lengths compared to single-type counterparts.

2) Impact of Training Set Size: To investigate the impact of training set size on PassTree, we extract 1 million, 5 million, and 10 million passwords from both Rockyou and Tianya datasets as training sets. Subsequently, a random sample of 100,000 passwords is chosen from the remaining data to serve

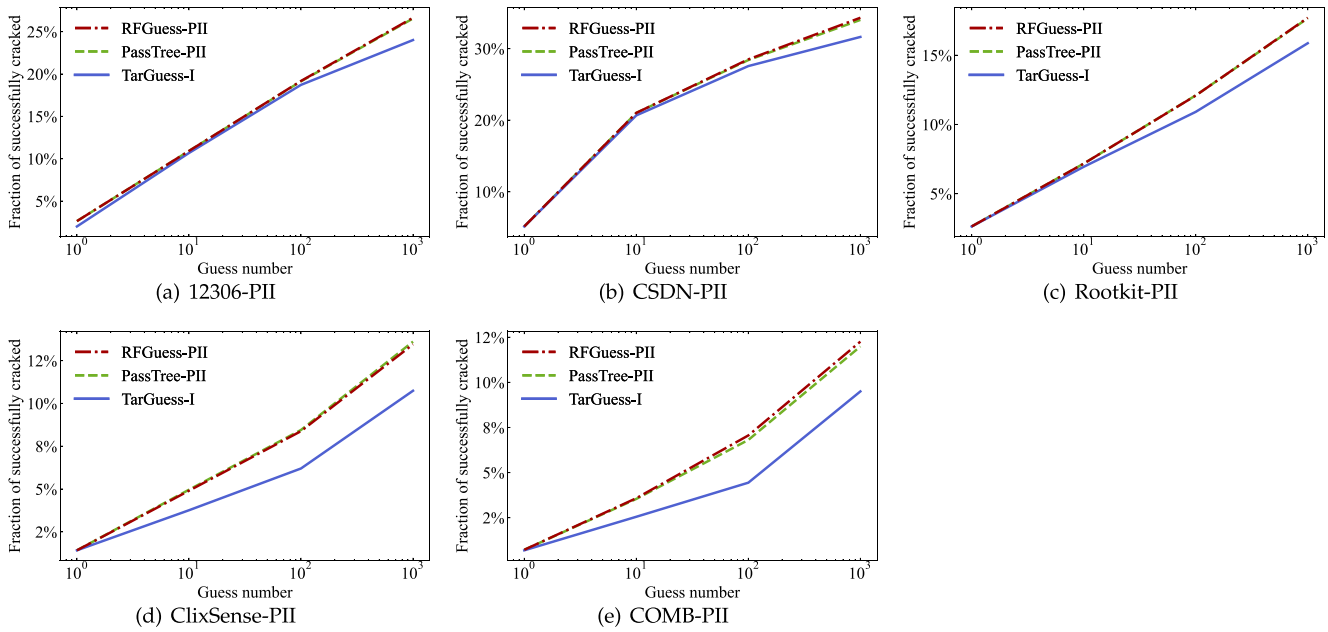


Fig. 11. A comparison of our PassTree-PII and other guessing models (i.e., RFGuess-PII [30] and TarGuess-I [8]) under targeted guessing based on personally identifiable information (PII). The guessing success rate curve of PassTree-PII generally closely overlaps with that of RFGuess-PII, with minor differences ranging from -0.04% to 0.30%.

as the testing set. The experimental results are presented in Fig. 9.

The results show that increasing the size of the training set can improve the cracking accuracy of PassTree. The improvement becomes more significant when the number of guesses exceeds 10^4. This is because increasing the training set size allows PassTree to learn password features more extensively. When the number of guesses is less than 10^4, most guesses generated by the model are likely to originate from the high-frequency passwords in the dataset. In such cases, the performance of models trained on datasets of different sizes tends to be similar, as the cracking rates do not vary significantly.

However, as the number of guesses increases, PassTree needs to crack passwords with lower frequencies, requiring a more extensive guessing space. Unfortunately, with a training set size of 1 million, PassTree cannot fully learn the password features, resulting in a smaller guessing space. Indeed, the cracking performance of PassTree with training set sizes of 5 million and 10 million is relatively close. The improvement brought by increasing the dataset size exhibits diminishing returns, where further increasing the training set size leads to limited gains while incurring additional time and resource consumption for training. Taking these factors into account, it is reasonable to select a training set size of 10 million for conducting large-scale comparative experiments.

D. Analysis of PII-Based Guessing

As shown in Fig. 11, our proposed PassTree-PII achieves an average cracking rate of 20.77% with 1,000 guesses, which is significantly higher than Targuess-I (with an average cracking rate of 18.35%). Moreover, the cracking rate curve of PassTree-PII closely overlaps with that of the RFGuess-PII, with differences ranging from -0.04% to 0.30%. However, in terms of guessing speed and memory consumption (refer to Table X), our PassTree-PII performs remarkably well. It consumes only 0.87% of the memory used by RFGuess-PII, while being 16.60 times faster in terms of guessing speed. Our PassTree-PII *not only maintains a leading cracking success rate but also greatly enhances computational efficiency and resource utilization, making it an excellent choice for specific scenarios (such as time-sensitive or resource-constrained scenarios).*

TABLE X
COMPUTATIONAL RESOURCES OF DIFFERENT GUESSING ALGORITHMS†

Algorithm	PassTree-PII	Targuess-I [8]	RFGuess-PII [30]
Memory allocation (Training)	40 MB	23.7 MB	8,000 MB
Training time	39 s	4.36 s	97 s
Memory allocation (Generation)	73 MB	20 MB	8,300 MB
Generation speed (/10^4PWs)	94s	10min 52s	26min

†We choose CSDN-PII dataset to to test the resource consumption of different models. The device is: Intel(R)Core(TM) i7-9750H CPU@2.60GHz 2.59GHz.

V. DISCUSSION

Here, we discuss potential areas for future expansion of PassTree, as well as how to apply our models for passwords strength evaluation and honeywords generation.

Future work: Currently, PassTree is applied to trawling guessing and PII-based targeted guessing. In future work, we can extend the application of our model to additional password guessing scenarios. For example, in password reuse attacks [27], an attacker uses a password leaked from site A to attack the

user's password on another site B. We can extract the tree structure from the leaked passwords and then modify each chunk corresponding to the nodes in the tree. When modifying, we can choose chunks that are similar to the original but have higher probabilities at the corresponding nodes. Performing an in-order traversal of the modified password tree would yield the guessing passwords.

In conditional password guessing (CPG), the attacker try to recover the complete password based on partial information (e.g., length or some characters). We can treat unknown character chunks as arbitrary chunk and hence construct a password template. Since each password template may correspond to multiple tree structures, we can merge all possible passwords generated from these tree structures in descending order of probability to obtain the corresponding guess sequence.

Password strength evaluation: Password strength meter (PSM) is typically used to quantify the strength of a password based on the number of guesses required to crack it. The two models proposed (referred to as PassTree and PassTree-PII) are probabilistic models that can be used to construct Password Strength Models (PSMs). More specifically, the evaluated passwords are first segmented to obtain password templates, and corresponding password trees are then constructed based on these templates. Subsequently, the probabilities of the tree structures and the probabilities of the chunk contents at each node are obtained separately. The final probability of the password is the product of all these probabilities. Finally, the Monte Carlo method [57] is employed to estimate the number of guesses required to crack the password, based on its calculated probability.

To further enhance efficiency, the trained model is stored separately for tree structures and node contents, allowing the two components of each password's probability to be computed in parallel. Although both RFGuess [30] and our PassTree are applicable for developing PSMs for trawling guessing and PII-based password guessing, PassTree exhibits notable advantages in terms of speed and space efficiency (see Table X).

Honeywords: Honeywords are utilized for detecting password leakage [58]. This technology stores k-1 decoy passwords (k is suggested to be 40 in [59]) together with the user's real password. Even if an attacker gains access to the password file, she needs to distinguish which is the user's real password. Once the number of logins using honeywords for a particular user surpasses the per-user threshold (e.g., 3), the system can detect a password leakage for that user. Similarly, when the number of logins using honeywords reaches the system-wide threshold (e.g., 10^4), the system determines a data set leak. We can model a user's password as a tree and then modify the content of different nodes (e.g., choosing similar but distinct chunks). Our model can generate honeywords that have a similar structure to the original password but are not identical.

VI. Conclusion

In this paper, we conduct a comprehensive analysis of user password characteristics and propose, for the first time, a method to parse passwords as trees. Compared to traditional sequence models like Markov and FLA, our tree-based parsing method reveals semantic connections and logical patterns within passwords, providing insights into how users construct them. Furthermore, we explore applying parsing trees to password guessing and design the trawling guessing model PassTree and PII-based targeted guessing model PassTree-PII. Through extensive experimental evaluations, we demonstrate the effectiveness of these two guessing models: PassTree achieves an average cracking rate of 37.16% within 10^6 guessing attempts, while PassTree-PII achieves an average cracking rate of 26.39% within 1,000 guessing attempts. These results indicate that both models perform on par with the leading approaches while demanding significantly less in terms of computational resources (e.g., consuming 0.87% of the memory used by RFGuess-PII while generating guesses 16.6 times faster).

References

[1] Q. Wang, D. Wang, C. Cheng, and D. He, "Quantum2FA: Efficient quantum-resistant two-factor authentication scheme for mobile devices," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 1, pp. 193–208, Jan./Feb. 2023.

[2] G. Liu, X. Gao, and H. Wang, "An investigation of identity-account inconsistency in single sign-on," in *Proc. World Wide Web Conf.*, 2021, pp. 105–117.

[3] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. IEEE Symp. Secur. Privacy*, pp. 553–567.

[4] E. Chase, "The password is dead; long live the password," Apr. 2017. [Online]. Available: https://sbscyber.com/resources/the-password-is-dead-long-live-the-password

[5] N. Cicchitto, "Passwords are not dead: The increasing need for SSO," Jan. 2022. [Online]. Available: https://solutionsreview.com/identity-management/passwords-are-not-dead-the-increasing-need-for-sso/

[6] L. Lassak, E. Pan, B. Ur, and M. Golla, "Why aren't we using passkeys? Obstacles companies face deploying FIDO2 passwordless authentication," in *Proc. 33rd USENIX Conf. Secur. Symp.*, 2024, pp. 1–18.

[7] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, "Beyond credential stuffing: Password similarity models using neural networks," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2019, pp. 417–434.

[8] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 1242–1254.

[9] Y. Li, H. Wang, and K. Sun, "A study of personal information in human-chosen passwords and its security implications," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[10] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 1–15.

[11] M. Golla et al., ""What was that site doing with my facebook password?": Designing password-reuse notifications," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2018, pp. 1549–1566.

[12] B. Ives, K. R. Walsh, and H. Schneider, "The domino effect of password reuse," *Commun. ACM*, vol. 47, no. 4, pp. 75–78, 2004.

[13] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2005, pp. 364–372.

[14] W. Melicher et al., "Fast, lean, and accurate: Modeling password guessability using neural networks," in *Proc. 25th USENIX Conf. Secur. Symp.*, 2016, pp. 1–17.

[15] A. S. Brown, E. Bracken, S. Zoccoli, and K. Douglas, "Generating and remembering passwords," *Appl. Cogn. Psychol.*, vol. 18, no. 6, pp. 641–651, 2004.

[16] M. AlSabah, G. Oligeri, and R. Riley, "Your culture is in your password: An analysis of a demographically-diverse password dataset," *Comput. Secur.*, vol. 77, pp. 427–441, 2018.

[17] J. J. Yan, A. F. Blackwell, R. J. Anderson, and A. Grant, "Password memorability and security: Empirical results," *IEEE Secur. Priv.*, vol. 2, no. 5, pp. 25–31, Sep./Oct. 2004.

[18] R. Shay et al., "Encountering stronger password requirements: User attitudes and behaviors," in *Proc. 6th Symp. Usable Privacy Secur.*, 2010, pp. 1–20.
[19] D. Wang, P. Wang, D. He, and Y. Tian, "Birthday, name and bifacial-security: Understanding passwords of Chinese web users," in *Proc. USENIX Conf. Secur. Symp.*, 2019, pp. 1537–1555.
[20] K. H. Hong, U. G. Kang, and B. M. Lee, "Enhanced evaluation model of security strength for passwords using integrated korean and english password dictionaries," *Secur. Commun. Netw.*, vol. 2021, pp. 3122627:1–3122627:13, 2021.
[21] E. Gerlitz, M. Häring, and M. Smith, "Please do not use !?_ or your license plate number: Analyzing password policies in german companies," in *Proc. USENIX Conf. Usable Privacy Secur.*, 2021, pp. 17–36.
[22] D. Florêncio, C. Herley, and P. C. van Oorschot, "An administrator's guide to internet password research," in *Proc. USENIX Conf. Large Installation Syst. Admin.*, 2014, pp. 35–52.
[23] A. Rao, B. Jha, and G. Kini, "Effect of grammar on security of long passwords," in *Proc. 3rd ACM Conf. Data Appl. Secur. Privacy*, 2013, pp. 317–324.
[24] R. Veras, C. Collins, and J. Thorpe, "A large-scale analysis of the semantic password model and linguistic patterns in passwords," *ACM Trans. Priv. Secur.*, vol. 24, no. 3, pp. 20:1–20:21, 2021.
[25] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2009, pp. 391–405.
[26] M. Xu, C. Wang, J. Yu, J. Zhang, K. Zhang, and W. Han, "Chunk-level password guessing: Towards modeling refined password composition representations," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2020, pp. 5–20.
[27] D. Wang, Y. Zou, Y. Xiao, S. Ma, and X. Chen, "Pass2edit: A multi-step generative model for guessing edited passwords," in *Proc. USENIX Conf. Secur. Symp.*, 2023, pp. 983–1000.
[28] X. Kedong and D. Wang, "Pointerguess: Targeted password guessing model using pointer mechanism," in *Proc. USENIX Conf. Secur. Symp.*, 2019, pp. 5555–5572.
[29] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2014, pp. 689–704.
[30] D. Wang, Y. Zou, Z. Zhang, and K. Xiu, "Password guessing using random forest," in *Proc. USENIX Conf. Secur. Symp.*, 2023, pp. 965–982.
[31] J. Xie, H. Cheng, R. Zhu, P. Wang, and K. Liang, "Wordmarkov: A new password probability model of semantics," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2022, pp. 3034–3038.
[32] H. Cheng, W. Li, P. Wang, and K. Liang, "Improved probabilistic context-free grammars for passwords using word extraction," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2021, pp. 2690–2694.
[33] D. Pasquini, G. Ateniese, and M. Bernaschi, "Interpretable probabilistic password strength meters via deep learning," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2020, pp. 502–522.
[34] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, and C. Abdelberi, "OMEN: Faster password guessing using an ordered markov enumerator," in *Proc. Int. Symp. Eng. Secu. Softw. Syst.*, 2015, pp. 119–132.
[35] R. Veras, C. Collins, and J. Thorpe, "On semantic patterns of passwords and their security impact," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 1–18.
[36] Z. Li, W. Han, and W. Xu, "A large-scale empirical analysis of chinese web passwords," in *Proc. USENIX Conf. Secur. Symp.*, 2014, pp. 559–574.
[37] L. Xu et al., "Password guessing based on LSTM recurrent neural networks," in *Proc. IEEE Int. Conf. Comput. Sci. Eng., IEEE Int. Conf. Embedded Ubiquitous Comput.*, 2017, pp. 785–788.
[38] Y. Fang, K. Liu, F. Jing, and Z. Zuo, "Password guessing based on semantic analysis and neural networks," in *Proc. Chin. Conf. Trusted Comput. Inf. Secur.*, 2018, pp. 84–98.
[39] M. Zhang, Q. Zhang, X. Hu, and W. Liu, "A password cracking method based on structure partition and bilstm recurrent neural network," in *Proc. Proc. Int. Conf. Commun. Netw. Secur.*, 2018, pp. 79–83.
[40] B. Hitaj, P. Gasti, G. Ateniese, and F. Pérez-Cruz, "PassGAN: A deep learning approach for password guessing," in *Proc. Int. Conf. Amer. Clin. Neurophysiol. Soc.*, 2019, pp. 217–237.
[41] Y. Liu et al., "Genpass: A general deep learning model for password guessing with PCFG rules and adversarial generation," in *Proc. Proc. IEEE Int. Conf. Comput. Commun.*, 2018, pp. 1–6.
[42] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti, "Improving password guessing via representation learning," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2021, pp. 1382–1399.
[43] X. Yu and Q. Liao, "Understanding user passwords through password prefix and postfix (P3) graph analysis and visualization," *Int. J. Inf. Sec.*, vol. 18, no. 5, pp. 647–663, 2019.
[44] D. Wang, D. He, H. Cheng, and P. Wang, "fuzzyPSM: A new password strength meter using fuzzy probabilistic context-free grammars," in *Proc. IEEE/IFIP Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2016, pp. 595–606.
[45] J. Tan, L. Bauer, N. Christin, and L. F. Cranor, "Practical recommendations for stronger, more usable passwords combining minimum-strength, minimum-length, and blocklist requirements," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2020, pp. 1407–1426.
[46] R. Castellucci, "Cracking CryptoCurrency brainwallets," Aug. 2015. [Online]. Available: https://defcon.org/html/defcon-23/dc-23-index.html
[47] D. Wang and P. Wang, "The emperor's new password creation policies: An evaluation of leading web services and the effect of role in resisting against online guessing," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2015, pp. 456–477.
[48] S. Alroomi and F. Li, "Measuring website password creation policies at scale," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2023, pp. 3108–3122.
[49] A. Adams and M. A. Sasse, "Users are not the enemy," *Commun. ACM*, vol. 42, no. 12, pp. 40–46, 1999.
[50] C. De Marcken, "Unsupervised language acquisition," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, USA, 1996.
[51] M. Creutz and K. Lagus, "Inducing the morphological lexicon of a natural language from unannotated text," in *Proc. Int. Interdiscipl. Conf. Adaptive Knowl. Representation Reasoning*, 2005, pp. 51–59.
[52] Q. Dong, C. Jia, F. Duan, and D. Wang, "RLS-PSM: A robust and accurate password strength meter based on reuse, leet and separation," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 4988–5002, 2021.
[53] lakiw, "Probabilistic context free grammar (PCFG) password guess generator," Dec. 2019. [Online]. Available: https://github.com/lakiw/pcfg_cracker
[54] D. L. Wheeler, "Zxcvbn: Low-budget password strength estimation," in *Proc. USENIX Conf. Secur. Symp.*, 2016, pp. 157–173.
[55] Z. Xie, M. Zhang, A. Yin, and Z. Li, "A new targeted password guessing model," in *Proc. Proc. 25th Australas. Conf. Inf. Secur. Privacy*, 2020, pp. 350–368.
[56] Q. Dong, D. Wang, Y. Shen, and C. Jia, "PII-PSM: A new targeted password strength meter using personally identifiable information," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.*, 2022, pp. 648–669.
[57] M. Dell'Amico and M. Filippone, "Monte carlo strength evaluation: Fast and reliable password checking," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2015, pp. 158–169.
[58] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2013, pp. 145–160.
[59] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang, "How to attack and generate honeywords," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2022, pp. 966–983.

Ding Wang received the PhD degree in information security from Peking University, in 2017, and was supported by the "Boya Postdoctoral Fellowship" in Peking University from 2017 to 2019. Currently, he is a full professor with Nankai University. As the first author (or corresponding author), he has published more than 90 papers at venues like IEEE S&P, *ACM Computing Classification System*, NDSS, USENIX Security, *IEEE Transactions on Information Forensics and Security* and *IEEE Transactions on Dependable and Secure Computing*. His research has been reported by more than 200 medias like Daily Mail, Forbes, IEEE Spectrum and Communications of the ACM, appeared in the Elsevier 2017 "Article Selection Celebrating Computer Science Research in China", and resulted in the revision of the authentication guideline NIST SP800-63-2. He has been involved in the community as a PC Chair/TPC member for more than 60 international conferences such as NDSS 2023-2025, ACM CCS 2022, RAID 2024/2023, PETS 2022-2024, ACSAC 2020-2024, AsiaCCS 2022/2021, ICICS 2019-2024, and SPNCE 2020–2022. He has received the ACM China Outstanding Doctoral Dissertation Award, the Best Paper Award at INSCRYPT 2018, the Outstanding Youth Award of China Association for Cryptologic Research, the Young Scientist Nomination Award for Powerful Nation, and the First Prize of Natural Science Award of Ministry of Education. His research interests include passwords, authentication, and provable security.

Xuan Shan received the bachelor's degree from the College of Computer Science, Nankai University, Tianjin, China, in 2021. She is currently working toward the master's degree. She has published a paper with USENIX Security 2023. Her research interests include password-based authentication and password security.

Yuxuan Wu received the bachelor's degree from the College of Computer Science, Nankai University, Tianjin, China, in 2023. He is currently working toward the master's degree from the College of Cyber Science, Nankai University, Tianjin, China. He has published a paper at ICICS 2022. His research interests include password-based authentication and deep learning-based password guessing.

Chunfu Jia is a PhD supervisor, a professor, and the Head of Department with the Department of Cyber Sciences, Nankai University. His main research interests include network and system security, applied cryptography, and malware analysis. His work has appeared in prestigious venues such as CCS, NDSS, S&P, USENIX Security, ESORICS, Asia CCS, *IEEE Transactions on Dependable and Secure Computing*, and *IEEE Transactions on Information Forensics and Security*.