Tasic *et al.*
EURASIP Journal on Information Security (2024) 2024:35
https://doi.org/10.1186/s13635-024-00183-2

EURASIP Journal on Information Security

RESEARCH

Open Access

Reinforcing cybersecurity with Bloom filters: a novel approach to password cracking efficiency

I. Tasic[1], A. Villafranca[2] and Maria-Dolores Cano[2*]

Abstract

Safeguarding digital information against unauthorized access is critical in the industry context. Techniques for cracking passwords are essential tools for both attackers and defenders. This study explores the utilization of Bloom filters, a probabilistic data structure known for its space and time efficiency, to refine the password-cracking process. We demonstrate that by employing Bloom filters, it is possible to significantly enhance the performance of password-cracking techniques in terms of speed and memory consumption. By conducting a comparative analysis with prevalent techniques such as hash tables and binary search, we demonstrate the superior performance of Bloom filters. The experimentation, utilizing a publicly available dataset of leaked password hashes, indicates a significant improvement in cracking efficiency. The findings contribute to the broader cybersecurity goal of developing resilient systems against password-related breaches, underscoring the importance of integrating cutting-edge research and practical applications to fortify digital defenses.

Keywords Cybersecurity, Password cracking, Bloom filters, Cyber threats protection, Critical infrastructure security

1 Introduction

Over the last decade, there has been a noticeable surge in cybercrime, creating ongoing and shifting challenges on a global scale for people, corporations, and state entities. The diverse aspects of cyber threats, notably those involving denial of service and social engineering, have led to widespread and enduring concern among security professionals, reaching what could be considered a pandemic in cybersecurity (World Economic Forum and [26]). In the wake of these threats, the cybersecurity field is transitioning to a paradigm where the integrity and origin of information become as critical as its confidentiality and availability. This shift is in response to an environment where misinformation and the rapid spread of unverified information can have far-reaching impacts. Consequently, there is an increasing emphasis on developing digital literacy and media savvy within the public and private sectors, promoting resilience against the tactics that threaten trust in the digital sphere.

Recent trends reveal that financially motivated attacks are most prevalent, with organized crime groups as prime actors. The second place in threats is for basic web application attacks, while social engineering ranks prominently in incidents and breaches. These often hinge on compromised credentials or advanced password-cracking methods, underscoring the urgency of evolving our cybersecurity defenses [23–26] Analyzing the more relevant incidents in industry related to cyber-attacks, it is found that many of them are related to passwords or credential. Raising industry awareness of the importance of applying the latest trends in cybersecurity and implementing best practices in terms of passwords

*Correspondence:
Maria-Dolores Cano
mdolores.cano@upct.es
[1] Business and Economics School, UCAM Universidad Católica San Antonio de Murcia, Murcia 30107, Spain
[2] Department of Information and Communication Technologies, Universidad Politécnica de Cartagena, Cartagena 30202, Spain

while encouraging the use of password auditing tools is strategic.

This paper introduces an innovative method to optimize password cracking, which is central to bolstering cybersecurity. Our approach employs Bloom filter data structures to enhance the efficiency of matching guessed passwords against known datasets. We demonstrate the efficacy of this method by carrying out a password-cracking procedure using a publicly available dataset of leaked password hashes. Its performance is evaluated by comparison with the most common methods employed in the related literature, namely, hash tables, binary trees, binary, and linear search. Additionally, we compare the Bloom filter approach with a Cuckoo filter, a similar probabilistic data structure, to further assess the efficiency and scalability of the proposed method. Our findings indicate that by using Bloom filters we can significantly improve the password-cracking process, which is critical in preempting unauthorized data access. Such advancements are essential in an era where cybersecurity in the industry must adapt rapidly to the ever-accelerating pace of technological and criminal innovation.

The rest of the paper is organized as follows. In Section 2, we summarize recent trends in industrial cyber security. The operation of Bloom filters is described in Section 3. Section 4 explains the research approach. The assessment of the proposal and its associated results are presented and analyzed in Section 5. The paper ends by emphasizing the most significant outcomes.

2 Recent trends in industrial cyber security

The industrial cybersecurity landscape is experiencing a profound transformation, driven by the widespread adoption of mobile technologies, a significant increase in computational power, and a shift towards digital-first practices across various sectors. This evolution has introduced complex challenges, including an increased risk of data breaches, intensified by the centralization of data in cloud services due to the prevalence of remote work and reliance on mobile devices. Moreover, the sophistication of cyberattacks has been notably enhanced by integrating artificial intelligence, machine learning, and automation, leading to more effective and rapid attack processes. This escalation of cyber threats is occurring alongside a significant gap in cybersecurity talent and expertise, highlighting a disconnect where the development of cybersecurity skills is not keeping pace with rapid digital transformations.

In response to these emerging threats, industries have adopted various innovative strategies. Zero-Trust Architecture (ZTA) implementation significantly shifts from traditional perimeter-based defenses, emphasizing the continuous verification of all users, assets, and resources to mitigate unauthorized access and internal threats. Behavioral Analytics is crucial in monitoring user behavior and device health, identifying anomalies that enable timely interventions against potential security breaches. Elastic Log Monitoring, using open-source platforms, has become essential for consolidating and analyzing log data across organizations in real time.

Homomorphic encryption has emerged as a pivotal technology in maintaining data confidentiality, allowing the manipulation of encrypted data without the need for decryption and thus ensuring data privacy while maintaining functional access. A risk-based approach to process automation focuses on automating lower-risk processes, thereby allocating more resources to critical security operations. Defensive AI and machine learning have become instrumental in detecting evolving attack patterns and refining cybersecurity workflows.

To combat the growing threat of ransomware, industries are developing both technical and organizational strategies, including resilient data repositories and multifactor authentication systems. The Secure Software Development Lifecycle approach, which integrates cybersecurity into every stage of software development, ensures that security considerations are a foundational element. The adoption of "X as a Service" and 'Infrastructure as Code' models has streamlined security management, leveraging cloud-based services and standardizing infrastructure processes for improved security [9]. The Software Bill of Materials [15] has become an essential tool in mitigating supply chain risks and preparing for regulatory scrutiny by documenting all software components.

An example of such applications can be found in the development of smart healthcare systems that focus on lightweight mutual authentication schemes for IoT-enabled infrastructures, ensuring the security of medical data [8]. In the Industrial Internet of Things (IIoT), the emphasis has been placed on secure communications and multifactor authentication [13]. For critical infrastructures, comprehensive SCADA security frameworks and attack-tree methodologies have been instrumental in analyzing cyber threats and developing effective countermeasures [21, 22]. Research on smart grid devices, including the use of tools like Shodan, has identified vulnerabilities and recommended enhanced security practices [2].

In summary, addressing cybersecurity challenges in the industrial sector requires a comprehensive and dynamic approach that combines advanced technologies with strategic methodologies. Continuous adaptation and innovation in cybersecurity practices are essential to counter complex and evolving cyber threats in various industrial contexts. Passwords are a key topic in industry

cyber security. Among the most recent significant cyber incidents worldwide [7], many are related to stolen credentials or passwords and they are always included in best-practices publications [1, 11, 12]. Integrating academic research and practical applications is crucial in shaping effective cybersecurity strategies, ensuring robust protection against emerging password-related threats. The future of industrial cybersecurity hinges on the ongoing evolution and adaptation of these strategies, keeping pace with the changing digital landscape and the sophisticated nature of these cyber threats.

3 Bloom filters

Bloom filters [4] function as probabilistic tools designed for efficiently checking if an element belongs to a set. Their key benefits include quick processing times and minimal use of memory. The unique aspect of this method is that it completely eliminates the chance of false negatives while allowing for a manageable rate of false positives. In other words, when querying the presence of an item, the filter will either accurately indicate the item is definitely not in the set or suggest it might be, albeit with a risk of error in the latter case.

The operation of a standard Bloom filter is as follows. We create a bit array of m bits. Then, for each item x to be inserted into the membership set, we assess the hashes of x applying k independent hash functions, namely, $h_1(x)$, $h_2(x)$, and so on until $h_k(x)$. The resulting hashes will be the indexes of the bit array where a value of 1 will be set. Afterward, if we want to check a membership, we calculate the k hashes of the item to be checked and verify the corresponding positions in the bit array; if all the corresponding bits are set to 1, then, this item is likely in the set (false positive rate > 0), otherwise the item is definitively not in the set (false negative rate = 0).

Let us give an example. Let us assume that we have selected a bit array of 18 bits ($m = 18$) as depicted in Fig. 1; initially, the array is set to 0. Then, we choose two hash functions ($k = 2$), e.g., we select h1 as a Fowler–Noll–Vo (FNV) hash function [10] and h2 as a CRC32

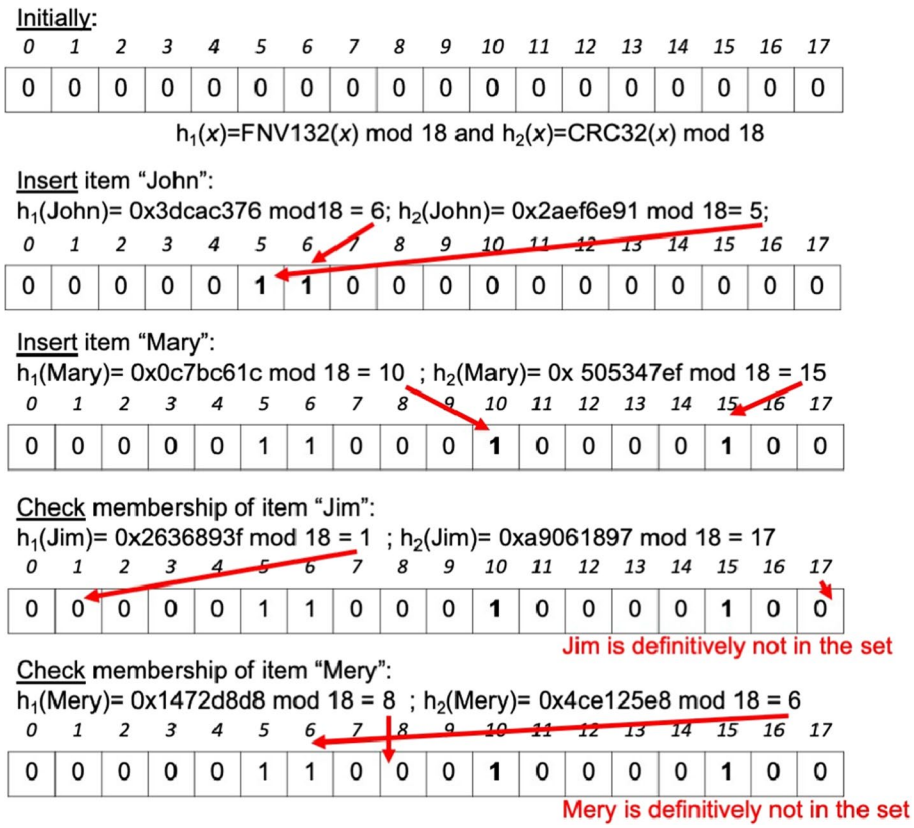


Fig. 1 This illustration demonstrates the functionality of a Bloom filter through a step-by-step process. Initially, we have an array of 18 positions (indexed from 0 to 17), all set to 0, indicating an empty data structure. To insert item "John", two hash functions (h1 and h2) are applied to the string "John", resulting in positions 6 and 5 being set to 1 in the bit array. To insert item "Mary", the same hash functions are applied to "Mary" changing the bits at positions 18 and 15 to 1. To determine if "Jim" is in the set, the hash functions are calculated for "Jim", pointing to positions 1 and 17. Since position 1 is still set to 0, the filter can definitively say that "Jim" is not in the set. When checking "Mery", the hash functions point to positions 8 and 6. Even though position 6 is set to 1 (from when "John" was added), because position 8 is set to 0, the filter can definitively say "Mery" is not in the set

hash function [5] (observe that we apply a mod 18 for the purposes of the example). We want to insert the item "John" in the set, so we assess h1(John). The output is 6, thus we set the position 6 of the array to 1. Again, we assess h2(John). The output is 5 so we set the position 5 of the array to 1. We proceed in the same way to insert the item "Mary", resulting in changing positions 10 and 15 of the array to 1. Now, let us assume that we want to check if the item "Mery" is in the set. We calculate the hashes of this item, which are 8 and 6. If positions 8 and 6 in the array were both set to 1, then "Mery" would be likely in the set, otherwise, "Mery" will definitively not belong to this set. As we can see in Fig. 1, bits 8 and 6 have a value of 0 and 1, consequently, this item is not in the set with likelihood 1.

Bloom filters are distinguished by certain fundamental traits: firstly, increasing the size of the bit array reduces the likelihood of false positives; secondly, using more hash functions tends to slow down the filter and leads to quicker saturation; and thirdly, too few hash functions result in a higher rate of false positives. A notable limitation of traditional Bloom filters is their inability to remove items without reconstructing the entire filter. This issue prompted the development of various Bloom filter adaptations to address such shortcomings, though these modifications often come with trade-offs in performance or increased memory requirements. The probability of encountering a false positive in a Bloom filter, denoted as ε, where

$$\varepsilon \approx \left(1 - \left(1 - {}^{1}/_{m}\right)^{k \bullet n}\right)^{k}, \tag{1}$$

and k represents the number of hash functions used, n is the expected number of items to be added, and m is the bit array's size. Researchers have documented these and other characteristics of Bloom filters, emphasizing considerations for their practical application [6, 14, 19, 20, 27]. Table 1 outlines key features of Bloom filters for consideration during their deployment.

Table 1 Some considerations for standard Bloom filters. m≡ size of the array also known as space; n ≡ number of items; k ≡ number of hash functions; – ≡ it cannot be performed

Description	Standard Bloom filter
Lookup operation	$O(k)$
Insert operation	$O(k)$
Delete operation	–
Number of hash functions k_{opt} to minimize false positive rate	$({}^{m}/_{n}) \bullet \ln 2$

4 Our proposal

This section describes how to speed up the password-cracking process by including Bloom filters. We hypothesize that the time required to perform password cracking will be significantly reduced if this type of data structure is employed. The authors are aware that the commercial software Hashcat [3] offers the use of a Bloom filter within its operation. However, to the authors' knowledge, there are no scientific works in the related literature addressing the use of Bloom filters in password cracking. Therefore, to test the hypothesis and evaluate the performance of our proposal, we proceed as follows (see Fig. 2).

Initially, we chose a collection of password data that includes NTLM hashes, which are believed to have been

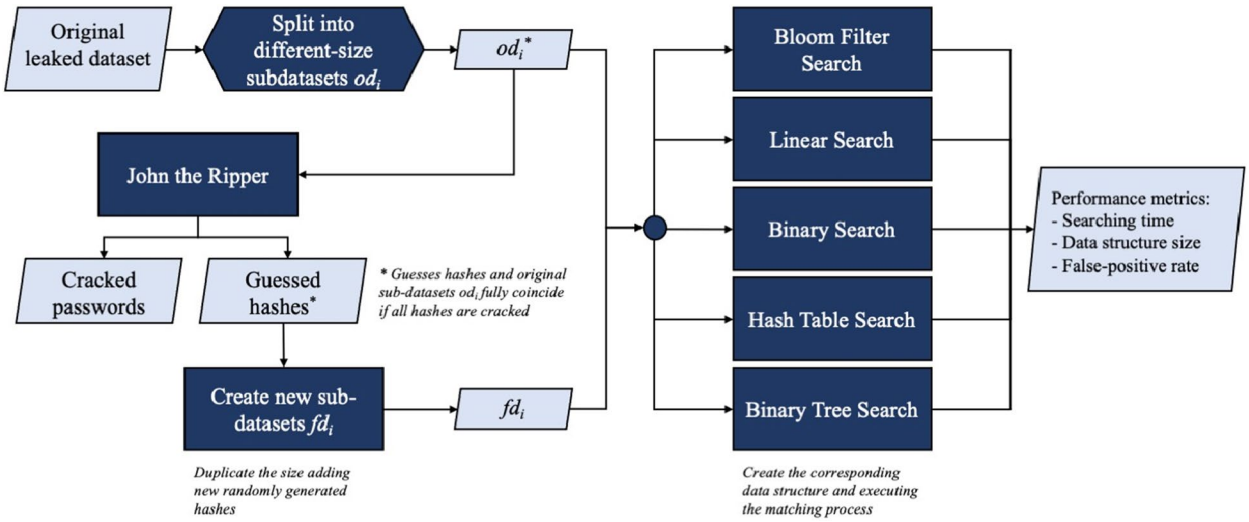


Fig. 2 Methodology applied

Table 2 Dataset. The first column enumerates each case for reference. The second column lists the number of NTLM hash values included in each case/file. The third column reveals the number of hashes that were successfully decrypted using John the Ripper. It is noteworthy that the number of cracked hashes matches the number of original hashes, with a 100% success rate in decryption. The fourth column shows the aggregated count of hashes, combining the original (cracked) hashes with the new randomly generated ones, to represent the total processing volume. The last column indicates the size of the fd *i* files in MB

i	# hashes of the original leaked file	#cracked hashes using JtR	#total number of hashes	Size of the filled file fd_i (MB)
1	3,897,669	3,897,669	7,795,337	340
2	1,948,834	1,948,834	3,897,667	168
3	974,417	974,417	1,948,833	84
4	487,208	487,208	974,415	42
5	243,604	243,604	487,207	20.7
6	121,802	121,802	243,603	10.3
7	60,901	60,901	121,801	5.1
8	30,450	30,450	60,899	2.6
9	15,225	15,225	30,449	1.3
10	7612	7612	15,223	0.6587
11	3806	3806	7611	0.3318
12	1903	1903	3805	0.1679
13	237	237	474	0.0202
14	118	118	236	0.0157
15	60	60	120	0.0054

exposed through various cyber incidents.[1] This collection of hashed passwords is being used strictly for academic inquiry; no personal identification data is being utilized, investigated, or revealed in our research. NTLM is a protocol used for verifying the identity of users, operating on a system where the user is presented with a challenge to confirm knowledge of their own password. These NTLM hash functions are still in use within current Windows operating systems. They are known for their rapid processing speed and the fact that they do not incorporate a salt in their hashing process. This original dataset is divided into several sub-datasets od_i with variable sizes (from millions of hashes to several dozens) as shown in Table 2.

Second, we choose John the Ripper (JtR) (Open [16]) as the password-cracking tool. JtR is a well-known, open-source, password-security auditing, and password-recovery software available for many operating systems. Briefly, during the cracking process, a file with hashes is passed to JtR as input. JtR generates possible passwords, also known as candidates, and calculates their hashes (guessed hashes). These hashes are compared with the hashes entered as input. If there is a match between a guessed hashed and an input hash, then that password has been cracked. It is at this point in the process, the matching, where the use of Bloom filter-type data structures can be beneficial, replacing the algorithms that are generally used to carry out this matching, such as hash tables or binary search algorithms [17]. Thus, instead of directly modifying JtR to include Bloom filters, we decided to use JtR to crack the hashes, obtaining the corresponding cracked passwords and storing the guessed hashes in a temporary file. Note that guessed hashes and od_i fully coincide if all input hashes are cracked. As a note, JtR was used in incremental mode, which combines both a dictionary attack and a brute force attack, and we employed the well-known RockYou dictionary, which contains millions of common passwords, which allowed us to attempt a more targeted attack before resorting to brute force. Then, we add approximately the same number of hashes the guessed hashes file contains but created completely randomly. The new sub-dataset is called fd_i. For instance, if od_1 has 3,897,669 hashes, all cracked with the corresponding guessed hashes, the new sub-dataset fd_1 contains 7,795,337 hashes, of which approximately half are randomly generated and the other half come from the original, leaked hashes file. The reason for adding these false hashes is to get closer to a real situation where not all passwords could be cracked, at least in a reasonable time, so that the possibility of false positives becomes feasible. Finally, the hashes in fd_i are randomly shuffled.

Third, we call different types of search algorithms to simulate the matching process in a different program. For each algorithm, the corresponding data structure needs to be created and then the matching process runs using as input od_i and fd_i. The Bloom filter data structure is implemented using the *pyBloomlive* library [18]. In the software implementation, the function to generate the Bloom filter uses as input parameters the od_i file, the capacity n, and the error rate ε. The capacity refers to the maximum number of elements that the filter can handle before the probability of false positives ε increases significantly. It is a very important parameter as selecting it appropriately ensures that the expected number of elements n can be handled without sacrificing the error rate ε. For our case, the Bloom filter capacity is equal to the number of hashes loaded from the file that has been selected to keep ε as low as possible. The error rate input parameter in a Bloom filter refers to the probability that the filter incorrectly reports that an element is present in the set when actually it is not, i.e., that there is a false positive. This parameter directly affects memory since a

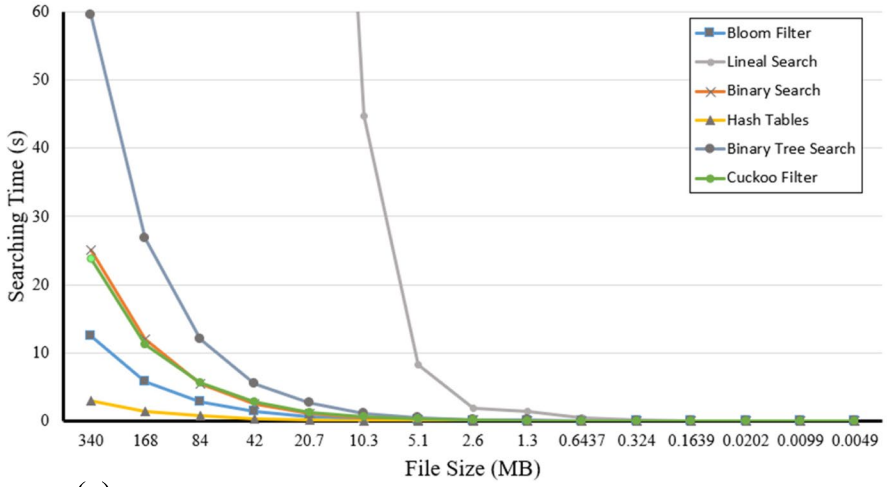
[1] https://haveibeenpwned.com/Passwords

low error rate reduces the probability of false positives but will require more memory resources m. For our case, we have set the error rate to 0.001, this indicates that we are willing to accept a rate of around 0.1% probability of false positives but that, on the other hand, we will consume more memory. Regarding the hash functions, the library uses SHA-1.

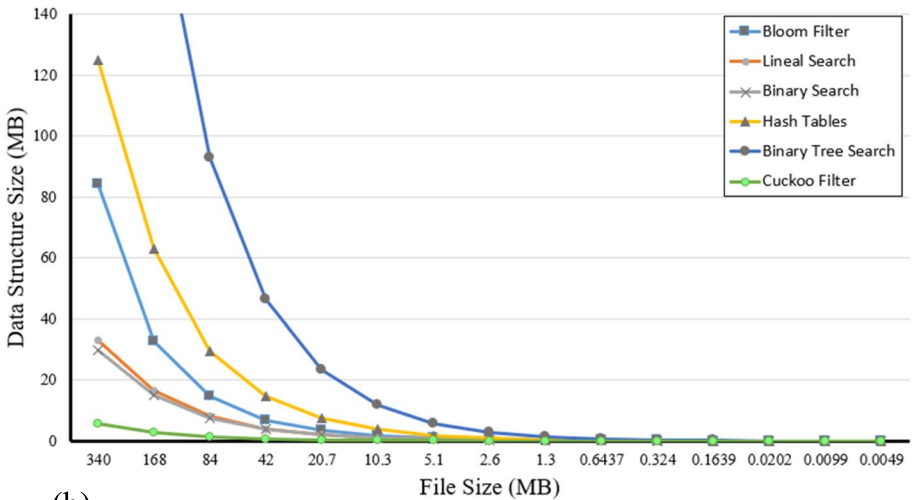
Finally, the performance metrics that we focus on are searching time (speed), data structure size, and the rate of false positives. The experiments were carried out using a Windows 11 OS, with a 13th Gen Intel(R) Core(TM) i7-13700H processor, 8GB RAM, and NVIDIA GEFORCE RTX 4070 Ti GPU.

5 Results

The results are illustrated in Fig. 3 and Fig. 4. As shown in Fig. 3a, we juxtapose various search strategies alongside the Bloom filter for evaluation in terms of speed. We can observe that the best results are obtained with hash tables. They exhibit the shortest duration needed to identify correspondences between fd_i and the hash targets od_i. The Bloom filter ranks as the second quickest approach, followed by cuckoo filters as the third in terms of speed. The results of the hash tables and the Bloom filter are within the same order of magnitude, though, and the smaller the sub-dataset the faster the convergence among methods.

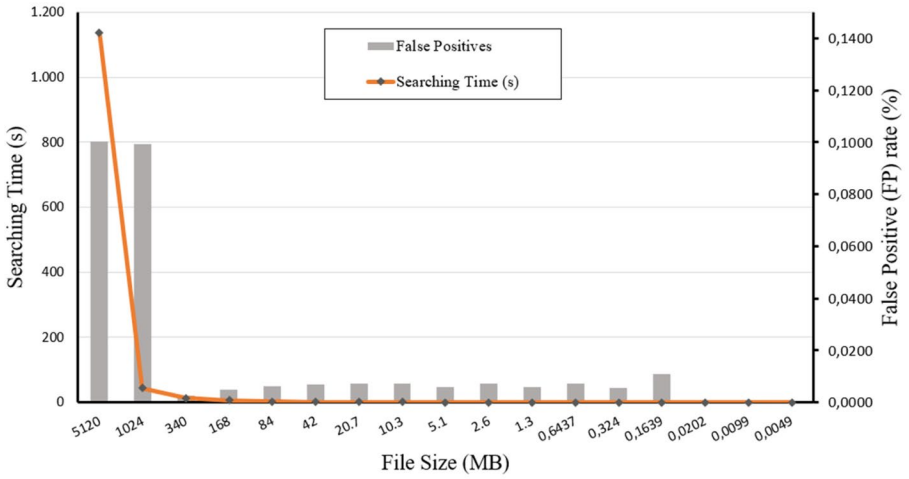


(a) Duration needed to pair up the generated hashes (derived from the guess passwords) with the intended hash targets across various search algorithms.

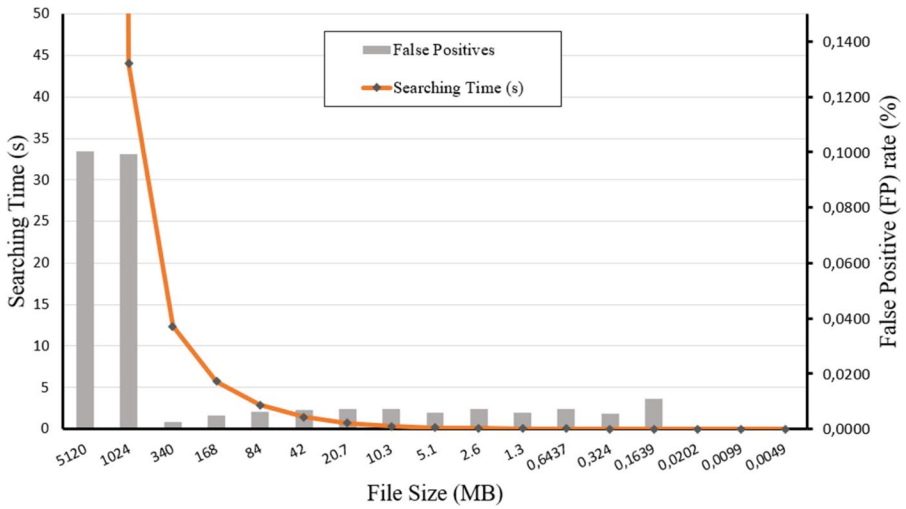


(b) Memory footprint of the data construct employed for pairing. This measurement stems from the output of the *asizeof()* function (sourced from the *pympler* library), which calculates an object's memory size in bytes. It is important to note that this calculation accounts solely for the memory directly associated with the object itself, excluding the memory used by any objects it may link to.

Fig. 3 Comparative analysis of the performance of searching strategies in the pairing process of password cracking



(a) Variation of Bloom filter efficiency, illustrating the searching time (left vertical axis) and the rate of false positives (right vertical axis) for a wide range of file sizes.



(b) A zoomed-in version of Figure 4(a), focusing on smaller file sizes to provide more clarity on the searching time and false positive rate for datasets below 1 GB.

Fig. 4 Variation of Bloom filter efficiency using as metrics the searching time (left vertical axis) and the rate of false positives (right vertical axis)

In Fig. 3b, we analyze the amount of memory required to carry out the matchmaking with the different search algorithms. From examining this graph, it becomes clear that the sizes of the data structures play a significant role and should not be overlooked. Cuckoo filters are the lightest, requiring the least amount of memory. This is due to their more efficient use of space, as they employ smaller hash tables and reallocation strategies to handle collisions, which minimizes the memory footprint. Following the Cuckoo filters, we have binary search and linear search, which also have relatively small memory requirements. While binary search and linear search consume less memory, i.e., are well positioned in this ranking, they come with significant drawbacks. Binary search requires a previously sorted list, which Bloom filters and Cuckoo filters do not, and it is much slower. Linear search, although simple and memory-efficient, scales poorly as the dataset size increases. Next in rank and close to the liner search are Bloom filters. The lack of need for prior sorting is a key advantage of Bloom filters, as it facilitates their use in applications where search speed is critical and prior sorting would be costly or inefficient. Hash tables, on the other hand, are one of the most consuming algorithms. When comparing Bloom filters against all these methods, we can see that Bloom filters occupy more memory than Cuckoo filters but are still more space-efficient than hash tables.

Linear search stands out for its simplicity but falls short of efficiency with growing datasets. Its straightforward

implementation comes at the cost of search times that escalate directly in proportion to the dataset's size, rendering it impractical for handling large volumes of data. Yet, for diminutive datasets, it presents an acceptable choice. Contrastingly, binary search marks a substantial leap in efficiency for sorted data collections. It shifts away from linear search's scalability issues by offering search times that increase logarithmically, thereby making it far more suited for expansive datasets. This method is distinguished by its precision, eliminating the risk of false outcomes. However, it demands that data be sorted ahead of time and can introduce added complexity in its setup. Binary search trees strike a balance, providing relatively swift search capabilities and precision. While they surpass linear search in terms of efficiency, they do not quite match up to the standard binary search in performance. Their memory footprint hinges on the tree's structure—specifically, its node count and depth, which could lead to considerable memory use. They emerge as a favored option when organizing data in a structured manner is paramount and accuracy cannot be compromised.

Standing out for their remarkable efficiency in search operations we have hash tables, leveraging hash functions to achieve immediate access to specific items. This mechanism is particularly advantageous for handling voluminous datasets, as it significantly reduces search times. The principle behind hash tables involves mapping each item to a unique location in the table through a hashing process, facilitating direct retrieval. In our implementation, we used Python's internal hash() function, which is based on the SipHash algorithm to prevent collision attacks. However, this efficiency comes with a caveat regarding scalability. As the volume of stored elements escalates, the physical dimensions of the hash table expand correspondingly, which could pose challenges in terms of increased memory requirements. This growth necessitates careful consideration in the selection of an optimal hash function to minimize the occurrence of hash collisions—situations where different items yield the same hash value, leading to potential retrieval issues. Consequently, Bloom filters present themselves as an appealing compromise among the critical factors of memory efficiency, accuracy, and speed.

Unlike hash tables, Bloom filters employ a probabilistic approach, allowing for rapid query responses with a compact representation of the dataset. While they excel in minimizing space consumption, Bloom filters achieve this efficiency at the expense of introducing a controlled rate of false positives, where a search might incorrectly indicate the presence of an element. This trade-off makes Bloom filters particularly suited for applications where speed and space efficiency outweigh the drawbacks of occasional inaccuracies. If we look at Fig. 4a, we can see the false positive rate we obtained during our study for all file sizes (from files smaller than 10 MB to larger ones of 1 GB and 5 GB). Figure 4b provides a zoomed-in view of smaller file sizes to better observe the detailed behavior of the Bloom filter. As expected, the search times increased as the file size grew. For instance, the 1 GB file had a search time of 44 s, while the 5 GB file reached 1138 s. However, the false positive rate remained well controlled, even with these larger datasets. For the 5 GB file (818 million hashes), the false positive rate was only 0.1004%, and for the 1 GB file (10 million hashes), it was even lower at 0.0993%. In smaller files, such as those of 340 MB or less, the search times were noticeably faster, with approximately 12 s for the 340 MB file and times under one second for the smaller files below 1 MB. Additionally, in these smaller datasets, the false positive rate was almost negligible. This may be because a filter has been used that is large enough for all items to fit without collisions, the hash functions used to generate the item fingerprints may have worked exceptionally well for this specific data, or the evaluated data may have properties that reduce the probability of collisions in the filter. Overall, these results demonstrate that Bloom filters can handle both small datasets with high speed and much larger files without significantly compromising accuracy or performance.

6 Conclusion

This study has shed light on the efficiency and suitability of Bloom filters for password cracking compared to other search algorithms working on different scenarios. We have observed that they offer a trade-off between speed, memory consumption, and the probability of false positives. They can be a solid choice for applications where a small probability of false positives can be tolerated, especially on larger datasets. We have found that as the size of items evaluated increases, the false positive rate in Bloom filters tends to decrease, making them especially effective in scenarios involving large datasets. Compared to Cuckoo filters, the latter are advantageous for dynamic environments due to their ability to delete elements, making them more adaptable than Bloom filters, which lack this feature. However, Cuckoo filters can be slower than Bloom filters because of the element reallocation during insertions. The choice between them depends on the application: Cuckoo filters are better for memory efficiency, while Bloom filters excel in speed when occasional false positives are acceptable. Additionally, the false positive rate in Bloom filters is influenced by the array size and the number of elements, which can be optimized by adjusting parameters like the number of hashes and the error rate. It is important to note that while Bloom filters are not infallible and there is always the possibility of false

positives due to their probabilistic nature, their flexibility and efficiency make them a valuable tool in a variety of applications. The ability to not require prior sorting of the data is a significant advantage in situations where search performance is critical and prior sorting would be costly or inefficient. In summary, Bloom filters have proven to be a valuable alternative not only in data search and retrieval but also in password cracking, and their use is particularly beneficial when efficiency in processing large data sets is a priority. Regarding future research directions, it would be valuable to explore the inclusion of this data structure in open-source password cracking tools, such as JtR, and to address the optimization of the false positive rate, especially in scenarios involving iterative hashing or salting techniques.

Authors' contributions

I.T.: Conceptualization, validation, investigation, methodology, visualization, writing—review and editing. A.V.: Data curation, formal analysis, investigation, software, visualization, writing—original draft. M.-D.C.: Conceptualization, data curation, formal analysis, funding acquisition, investigation, methodology, project administration, resources, supervision, validation, visualization, writing—original draft, writing—review and editing.

Funding

This research is being carried out within the framework of the Recovery, Transformation, and Resilience Plan funds, financed by the European Union (Next Generation).

Data availability

The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request.

Declarations

Competing interests

The authors declare no competing interests.

Received: 9 May 2024 Accepted: 28 October 2024
Published online: 04 November 2024

References

1. Abraham C, Chatterjee D, & Sims, R. R. (2019). Muddling through cybersecurity: insights from the U.S. healthcare industry. *Business Horizons, 62*(4), 539–548. https://doi.org/10.1016/j.bushor.2019.03.010
2. Ackley, D., & Yang, H. (2020). Exploration of smart grid device cybersecurity vulnerability using Shodan. In *In Proc. IEEE Power & Energy Society General Meeting (PESGM)* (pp. 1–5). Montreal, QC, Canada. https://doi.org/10.1109/PESGM41954.2020.9281544
3. Advanced Password Recovery. (2022). Hashcat. Retrieved February 9, 2022, from https://hashcat.net/hashcat/
4. B.H. Bloom, Space/time trade-offs in hash coding with al lowable errors. Commun. ACM **13**(7), 422–426 (1970). https://doi.org/10.1145/362686.362692
5. P.E. Boudreau, W.C. Bergman, D.R. Irvin, Performance of a cyclic redundancy check and its interaction with a data scrambler. IBM J. Res. Dev. **38**(6), 651–658 (1994). https://doi.org/10.1147/rd.386.0651
6. F.P. Cao, J. Almeida, A.Z. Broder, Summary cache: a scalable wide-area Web cache sharing protocol. IEEE/ACM Trans. Networking **8**(3), 281–293 (2000). https://doi.org/10.1109/90.851975
7. Center for Strategic & International Studies. (2024). Significant cyber incidents. Retrieved January 29, 2024, from https://www.csis.org/programs/strategic-technologies-program/significant-cyber-incidents
8. Das, S., & Namasudra, S. (2022). A lightweight and anonymous mutual authentication scheme for medical big data in distributed smart healthcare systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1–12. https://doi.org/10.1109/TCBB.2022.3230053
9. Florin, I. L., & Bălan, T. (2020). Vulnerability remediation in ICS infrastructure based on source code analysis. In *In Proc. 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)* (pp. 1–6). Bucharest, Romania. https://doi.org/10.1109/RoEduNet51892.2020.9324845
10. Fowler, G., Noll, L. C., Vo, K.-P., Eastlake, D., & Hansen, T. (2019). The FNV non-cryptographic hash algorithm. https://datatracker.ietf.org/doc/draft-eastlake-fnv/
11. H. George, A. Arnett, Implementing cybersecurity best practices for electrical infrastructure in a refinery: a case study. IEEE Ind. Appl. Mag. **27**(4), 18–24 (2021). https://doi.org/10.1109/MIAS.2021.3063095
12. E.P. Kechagias, G. Chatzistelios, G.A. Papadopoulos, P. Apostolou, Digital transformation of the maritime industry: a cybersecurity systemic approach. Int. J. Crit. Infrastruct. Prot. **37**(100526), 1–14 (2022). https://doi.org/10.1016/j.ijcip.2022.100526
13. Z. Li, Z. Yang, P. Szalachowski, J. Zhou, Building low-interactivity multifactor authenticated key exchange for industrial internet of things. IEEE Internet Things J. **8**(2), 844–859 (2021). https://doi.org/10.1109/JIOT.2020.3008773
14. H. Lim, J. Lee, C. Yim, Complement Bloom filter for identifying true positiveness of a Bloom Filter. IEEE Commun. Lett. **19**(11), 1905–1908 (2017). https://doi.org/10.1109/LCOMM.2015.2478462
15. National Institute of Standards and Technology (NIST). (n.d.). Software bill of materials (SBOM). Retrieved January 29, 2024, from https://www.cisa.gov/sbom
16. Open Wall. (2022). John the Ripper. Retrieved February 9, 2022, from https://www.openwall.com/john/
17. Open Wall. (2023). Password security: past, present, future. https://www.openwall.com/presentations/Passwords12-The-Future-Of-Hashing/
18. pyBloom-live Library. (2024). Retrieved from https://pypi.org/project/pybloom-live
19. Reviriego, P., Martínez, J., Larrabeiti, D., & Pontarelli, S. (2020). Cuckoo filters and bloom filters: comparison and application to packet classification. *IEEE Transactions on Network and Service Management*, *17*(4). https://doi.org/10.1109/TNSM.2020.3024680
20. Song, H., Dharmapurikar, S., Turner, J., & Lockwood, J. (2005). Fast hash table lookup using extended bloom filter: An aid to network processing. In *In Proc. Applications, technologies, architectures, and protocols for computer communications* (pp. 181–192). Philadelphia, PA. https://doi.org/10.1145/1080091.1080114
21. C.-W. Ten, C.-C. Liu, G. Manimaran, Vulnerability assessment of cybersecurity for SCADA systems. IEEE Trans. Power Syst. **23**(4), 1836–1846 (2008). https://doi.org/10.1109/TPWRS.2008.2002298
22. C.-W. Ten, G. Manimaran, C.-C. Liu, Cybersecurity for critical infrastructures: attack and defense modeling. IEEE Trans. Syst. Man Cybern. **40**(4), 853–865 (2010). https://doi.org/10.1109/TSMCA.2010.2048028
23. The Economist. (2021). To stop the ransomware pandemic, start with the basics. Retrieved February 17, 2022, from https://www.economist.com/leaders/2021/06/19/to-stop-the-ransomware-pandemic-start-with-the-basics
24. Verizon. (2021). *Verizon 2021 Data Breach Investigations Report*. Retrieved from http://verizon.com/dbir/
25. VMWare, Kroll, & RedCanary. (2022). *The state of incident response 2021: It's time for a confidence boost*. https://www.kroll.com/en/insights/publications/cyber/state-of-incident-response
26. World Economic Forum, & Long-term center for cybersecurity UC Berkeley. (2023). *Cybersecurity Futures 2030 New Foundations*. https://www.weforum.org/publications/cybersecurity-futures-2030-new-foundations/
27. Wu, Y., He, J., Yan, S., Wu, J., Yang, T., Ruas, O., … Cui, B. (2021). Elastic Bloom filter: deletable and expandable filter using elastic fingerprints. *IEEE Transactions on Computers*, *PP*. https://doi.org/10.1109/TC.2021.3067713

Publisher's Note