MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

A NOVEL HONEYWORD-BASED AUTHENTICATION SCHEME FOR PASSWORD DATABASE BREACHES

NEVZAT ÖZCANDAN

MASTER THESIS
Department of Computer Engineering

Thesis Supervisor
Prof. Dr. Ali Fuat ALKAYA

Thesis CO- Supervisor
Dr. Muhammed Ali BİNGÖL

ISTANBUL, 2022

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

A NOVEL HONEYWORD-BASED AUTHENTICATION SCHEME FOR PASSWORD DATABASE BREACHES

NEVZAT ÖZCANDAN

(524119008)

MASTER THESIS
Department of Computer Engineering

Thesis Supervisor
Prof. Dr. Ali Fuat ALKAYA

Thesis CO- Supervisor
Dr. Muhammed Ali BİNGÖL

ISTANBUL, 2022

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Dr. Ali Fuat ALKAYA and Dr. Muhammed Ali BİNGÖL for their continuous support and guidance throughout the journey on my master thesis. It was a great experience for me to have a chance to study with them. They always tried to encourage me when I felt hopeless and helped me a lot when I struggled with my study.

Besides my advisor, I also would like to thank my family for their understanding and encouragement during my study.

May, 2022 **Nevzat Özcandan**

TABLE OF CONTENTS

ÖZET

PAROLA VERİTABANI İHLALLERİ İÇİN BALLI KELİME TABANLI KİMLİK DOĞRULAMA YÖNTEMİ

Ballı kelimeler tekniği, hem parola güvenliğinde iyileştirmeler hem de her bir kullanıcının hesabıyla birlikte ballı kelimeler adlı tuzak parolaların kasıtlı olarak eklenmesiyle parola veritabanı ihlallerinin saptanmasını sağlar. Ancak, hedefli parola tahmini ve parola yeniden kullanımının neden olduğu çoklu sistem kesişim saldırıları gibi ballı kelime mekanizmasıyla ilişkili önemli güvenlik sorunları vardır. Bu tez çalışmasında ilk olarak, mevcut ballı kelime şemaları Superword, Append Secret Modeli ve Erguler'in şemasındaki tuzakların kısa bir analizini sunuyoruz. Özellikle, Erguler'in ballı kelime şemasına bir saldırı modeli önerilmiş ve saldırgan şifre dosyasını bir kez ele geçirse bile, Ergüler'in şemasının çevrimiçi kimlik doğrulama girişimleriyle tamamen kırılabileceği gösterilmiştir. İkinci olarak, parola dosyalarının sızdırılması ve çevrimdışı olarak bunların sistemlere izinsiz girilmesi için tersine çevrilmesi şeklindeki yaygın tehdidi ele alan Panacea adlı yeni bir ballı kelime tabanlı kimlik doğrulama şeması önerilmiştir. Sonuçları incelediğimizde şemamızın, parola dosyasındaki tehdidi algılamadan önce, kullanıcıların parola seçimlerinden bağımsız olarak sistemdeki her kullanıcı için verimli bir şekilde çalıştığı görülmüştür. Ek olarak, yakın zamanda önerilmiş olan ballı kelime şemalarının dağıtık parola depolama şemaları ile daha çok ortak noktası olduğu tartışılmıştır ve bu nedenle güvenlik ve karmaşıklık yönlerini tatmin etmede oldukça iyi olan Panacea' nın basitleştirilmesi olan alternatif bir şema daha önerilmiştir. Kapsamlı analiz, önerilen her iki şemamızın da daha önce önerilen ballı kelime şemalarının sınırlamalarının ve güvenlik açıklarının çoğunun üstesinden geldiğini göstermektedir. İddiaları doğrulamak için önerilen şemalarımızı çeşitli ataklar altında mevcut çalışmalarla karşılaştırılmıştır. Bu tez, bilgi güvenliği ile ilgilenenler başta olmak üzere parola veritabanının saldırgan tarafından çalınmasıyla birçok olası siber saldırının hedefi haline gelen sistemlerin daha güvenli hale getirilmesine yardımcı olacaktır.

Mayıs, 2022 **Nevzat Özcandan**

ABSTRACT

A NOVEL HONEYWORD-BASED AUTHENTICATION SCHEME FOR PASSWORD DATABASE BREACHES

“Honeywords" provide both enhancements to hashed password security and detection of password database breaches by intentional insertion of trap passwords–named as honeywords– along with each user’s account. Nevertheless, there are several primary security problems associated with the honeyword mechanism such as targeted password guessing and multiple system intersection attacks caused by password reuse. In this thesis, we first present a brief analysis of pitfalls in the existing honeyword schemes Superword, Append Secret Model and Erguler’s scheme. In particular, we propose an attack on Erguler’s honeyword scheme and show that it can be fully broken by online login attempts even if an adversary just compromises the password file once. We propose a novel honeyword scheme, named *Panacea*, that mitigates the threat of password file breaches and offline password inversion attacks. Our scheme works efficiently and for every user in the system regardless of their password choices before detecting the threat on a password file. Additionally, we discuss that recent honeyword schemes have more in common with distributed password storage schemes and thus suggest an alternative scheme, the simplification of the Panacea, that does reasonably well in satisfying security and complexity aspects. The extensive analysis demonstrates that both our proposed schemes overcome most of the limitations and vulnerabilities of previously proposed honeyword schemes. We compare our proposed schemes with the existing ones under various attacks to validate our claims. This thesis will help to make systems more secure, which have become the target of many possible cyberattacks, especially those who are interested in information security, by stealing the password database by the adversary.

May, 2022 **Nevzat Özcandan**

Claim of Originality

Although honeyword mechanism have been gained a great attention in the research community, it is inherently vulnerable to username/pair correlations, targeted password guessing, and multiple system intersection attack caused by password reuse. To fill this gap, we mainly concentrate on these three types of attacks for the authentication infrastructure in this thesis. Firstly, we present a brief analysis of pitfalls in the existing state-of-the-art schemes and show that the adversary can exploit the own vulnerabilities of these schemes or the inherent weaknesses of the use of honeywords. We report that some of the schemes can be fully broken under online login attacks. Further, a novel honeyword-based authentication scheme, called the Panacea, is proposed to both increase the total effort in obtaining plaintext forms of hashed passwords by the adversary and to detect the password compromise at the same time. We also simplify our scheme based on recent developments so that it is more efficient in different use cases. The thesis gives an accurate analysis on honeyword-based authentication systems and answers the question of "How can a honeyword system best be designed to mitigate the offline inversion attacks for all users regardless of the password choices and to detect whether the system is under attack?". To validate the claims, we compare our proposed schemes with existing state-of-the-art honeyword-based storage schemes on various security and complexity parameters. We summarize results of these comparisons in an accurate manner.

SYMBOLS

U_i	**:** Username of the i-th user
P_i	**:** Password of the i-th user
$H()$	**:** Cryptographic hash function
N	**:** Total number of users in each system
$\vec{\boldsymbol{\sigma}}_i$	**:** Secured sweetword vector of user i
$\|\|$	**:** Concatenation operator
$\oplus$	**:** Bitwise XOR operator
S_i	**:** Salt of user i
σ_{ij}	**:** j-th Secured Password of user i (i.e., $H(P\|\|S_i) \oplus r_{ij}$)
S_i	**:** Salt of user i
ℓ	**:** Size of a salt
λ	**:** Size of the security parameter
$\mathscr{A}$	**:** The adversary
$\mathscr{O}_H$	**:** An oracle for a hash function
B_i	**:** Bloom filter of user i
$\vec{\boldsymbol{R}}_i$	**:** Registration Vector of user i
$\vec{\boldsymbol{R}}'_i$	**:** Login Vector of user i
$< . >$	**:** Sorted set
$\{.\}$	**:** Unsorted set

ABBREVIATIONS

MS	**:** Main Server
HC	**:** HoneyChecker Server
HGA	**:** Honeyword Generation Algorithm
HBA	**:** Honeyword-based Authentication
PII	**:** Personally Identifiable Information
MSP	**:** Multiple System Protection
TPGP	**:** Targeted Password Guessing Protection
DoS	**:** Denial-of-Service
UI	**:** User Interface
PPT	**:** Probabilistic Polynomial Time
EPM	**:** Evolving Password Model
ASM	**:** Append Secret Model
PDP	**:** Paired Distance Protocol
SHA	**:** Secure Hash Algorithm
MD	**:** Message Digest Algorithm
SSV	**:** Secured Sweetword Vector

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Password based authentication systems remain the most popular and widely used in user authentication [1, 2, 3] since they are simple, cost effective, and user friendly even though various authentication techniques are put forward [4, 5, 6, 7, 8]. Instead of storing passwords in the raw form (i.e., plaintext), they are enciphered using a cryptographic hash function and then stored along with the other user related information to ensure confidentiality. Human chosen passwords are almost entirely foreseeable since users tend to select short, non-random and easily-guessed passwords to facilitate usage [9, 10, 11], so when the hashed password file is leaked, passwords concealed in hash value can be reversed with overwhelming percentage by brute-force search.

One of the most notorious attack models for inverting hashes to obtain the plaintext of passwords is *dictionary attack* [12]. In this attack model, owing to improvements of massively parallelized computing hardware, such as Graphics Processing Units (GPUs), adversary can cheaply gain strong computation ability and efficiently create dictionaries with commonly used passwords. Any adversary with this precomputed dictionary, only needs to get access to the server database. She can then simply compare the entries and discover user passwords. In general practise, an auxiliary random value called salt [12] that is stored in the database along with the password hash is used to randomize hash output. It introduces more cost to brute-force search and can be used to resist specialized attacks like rainbow table attack [13]. Also, many cryptographic protocols, such as sophisticated hash functions (e.g., bcrypt) have been designed for storing passwords more securely at the server side. However, although using salt and/or sophisticated password hash function makes brute-force search more complex and costly, this does not constitute a real difficulty for an adversary to retrieve them by a high percentage by employing modern machine learning-based cracking algorithms [14, 15] and password-cracking hardware like GPU [16]. An adversary equipped with dedicated password-cracking hardware [17] quickly inverts a hashed password of length not more than 15 character within several hours. Thus, when an adversary has access to the password hash file, it is reasonable to presume that most of them can be inverted offline. Besides, increasing the adversary's offline inversion workload does not facilitate the detection of password

file breaches since the adversary only needs to perform brute-forcing locally. In other words, building a defense procedure considering only making the password cracking more difficult such as in [18] is not sufficient to deal with and detect such an extraordinary threat scenario.

Having a responsive defense procedure to be counteractive against the threat on the password file is crucial because this will inform the system administrator about the database leakage immediately after the adversary tries to perform an unauthorized action by using the stolen password file. In 2013, Juels and Rivest proposed the concept of adding a set of decoy passwords named honeywords [19], along with the correct password to detect the threat on a password file. In this technique, instead of a password, each username is associated with a number of sweetwords consisting of a true password and decoy ones created by the server in random order. Even if the password file is leaked and an adversary recovers the passwords from their hash values, he has to further recognize the actual one from a number of sweetwords. If the adversary tries to log into the system using a honeyword instead of the real password, the system triggers an alarm, detecting the disclosure of the password file. Consequently, any login attempt with one of the honeywords notifies the administrator about a password file leakage by triggering an alarm. The method was inspired by other following cyber-defense techniques. Use of decoys for building a theft-resistant password manager was proposed in [20] called as Kamouflage. This model generates multiple decoy password lists along with the real user password list. Honeypot mechanism [21, 22] deliberately sets up some fake users in the system to affect the attacker to interact with the incorrect target and can raise an alarm for password file disclosure if the adversary attempts to login with fake accounts. Nonetheless, honeypot provides low detection success rate, since fake users are generally much less than real users, thus these users might be distinguished by the adversary [23].

Password hash file is sensitive and an attractive target for the attackers. The breach of hash files is a serious security issue that has affected millions of users and numerous reputable web services like Yahoo [24], Dropbox [25], LinkedIn [26] because stolen passwords can also trigger many other probable cyber-security issues for both users and companies. Also some of those services (such as Yahoo, Phpbb and Anthem) have leaked their users' passwords more than once [24, 27]. Disclosures of password files are

generally not detected for a long time even months or years after they have initially been compromised. For instance, the leakage of almost 3 billion Yahoo passwords was occurred in 2013, but users are requested to change their passwords four years later [24]. Similarly, in case of Dropbox, the breach of almost 68 million passwords was declared in 2016, while the damage had happened four years earlier [25]. Hence, this is plenty of time for the adversary to successfully crack the passwords and then to share or sell them online or to impersonate the users. These password database leakages indicate the existence of security problems in existing password-based authentication techniques which can be unsuccessful to guarantee user privacy.

Within the research on decoy passwords, there exists one more proposal other than honeyword based authentication schemes. Almeshekah et. al [28] utilize a machine-dependent function (e.g., physically unclonable function) in the password hash at the server-side to prevent offline cracking of the list of hashed passwords if breached. More relevantly, an adversary who is not aware of this defensive protocol and tries to crack the database offline will generate plausible fake passwords (what they called "ersatzpasswords") that, when entered, notify the site administrator about the breach. Note that ersatzpasswords are useful in notifying the administrator about the breach only if the adversary is not aware of the use of this scheme, otherwise, the adversary will be aware that passwords produced through offline cracking without access to the machine-dependent function are ersatzpasswords. Besides, it requires substantial changes to the server-side authentication system and suffers from its poor scalability [29].

While potentially effective, honeyword mechanisms are inherently vulnerable to various attacks. One of the crucial threats is targeted password guessing. Wang et al. [15, 29] discuss that the usage of personally identifiable information (PII) while building a password is very common. They observe that some previously stolen password databases contain PII-associated passwords up to 51.43%. They also find that the real password can be distinguished from honeywords with a success rate between 56.8%-67.9% by making only *one* guess. Another important vulnerability that dramatically limits the overall effort of distributed honeyword security architecture is multiple system intersection attacks caused by password reuse. In [30], authors observe that 52% of the users reuse the same password (or with only slight modifications) and a recent survey [31] by Google, in

partnership with Harris Poll greatly confirms the password reuse problem. Other issues related to recognizable patterns in password and correlational hazards (i.e., flatness) [32, 33] between the username and the password also weaken the security of the honeywords mechanisms. The honeyword-based authentication technique basically aims to enhance the security of hashed password of *every* user account [19]. These all show that if the adversary obtains password hashes, for many users, the success rate of hashed password protection and intrusion detection can be much lower than the expected one considering these issues. We emphasize that having high quality honeywords is closely related to the user's password choice behavior and thus honeywords may not provide adequate protection for *every* user in the system. We also observe that recent proposals fail to satisfy failover case, defined in [19], which is one of the important criterion of the honeyword-based authentication (HBA) context. This scrutiny motivates us to re-examine the security aspects of the state-of-the-art schemes (mainly because a flaw in the working principle may often cause many security threats to attaining the intended security goals).

1.1. Contributions of the Thesis

In this thesis, we aim to give an accurate analysis on honeyword-based authentication systems and answer the question of "How can a honeyword system best be designed to mitigate the offline inversion attacks for all users regardless of the password choices and to detect whether the system is under attack?". We get a clearer view of honeyword schemes' security and other aspects to protect organization's authentication infrastructure. Main contributions of this paper are summarized below.

(i) We analyze the security and complexity of existing honeyword schemes and show that Guo et al. [34], Akshima et al. [35], Tian et al. [36] have serious security issues. We also propose an effective attack on Erguler's scheme [33] due to its periodical honeyindex update. In our attack, even if the password file is just compromised once, and find that an adversary can obtain almost all correct username/real password pairs from the targeted system.

(ii) We develop a novel scheme, what we called *Panacea*, to provide both two security aspects of the honeyword scheme– increasing the total effort in obtaining plaintext forms of hashed passwords and detecting the password compromise– at the same

time. We also simplify our scheme based on recent developments so that it is more efficient in different use cases.

(iii) We revisit the flatness property regarding the indistinguishability of the user-chosen password from the decoys that Juels and Rivest gave in [19].

(iv) To justify our claims, we compare our proposed schemes with existing state-of-the-art honeyword-based storage schemes on various parameters. We summarize results of these comparisons in Table 6.1 and Table 6.2.

As a contribution to the literature, a journal paper titled "Panacea: A Novel Honeyword-based Authentication Scheme" that summarizes a part of this study is submitted to the Computers&Security Journal.

1.2. Outline of the Thesis

The rest of the thesis is organized as follows. In Section 2, we review the working principle of the honeyword mechanism and the Bloom filter data structure. In Section 3, we present our improved scheme, the Panacea, followed by the security analysis about the proposed scheme in Section 4. In Section 5, we point out the soundness and security problems of existing honeyword-based schemes and propose an effective attack on Erguler's scheme. In Section 6, We compare both of our scheme with the previously proposed schemes in terms of security and complexity. In Section 7, we conclude this thesis.

2. BACKGROUND

In this section, we briefly explain the working principle of Honeywords and Bloom filter structure to help in better understanding the details of the proposed idea.

2.1. Review of Honeywords

Jules and Rivest introduce the seminal work “Honeywords" as an important contribution that adds a useful defensive layer to potentially deter or detect the thefts of a sensitive file comprised of the registered users’ hashed passwords [19]. The idea provides both enhancements to password security and detection to password database breaches by intentional insertion of trap passwords–named as honeywords– along with each user’s account. They store each username associated with a set consisting of hash values of the user’s real password and honeywords. Even if an adversary obtains the hashed password file and inverts password hashes, he only gets a list of probable plain-text passwords for each user and cannot be sure about which password is real. The probable password list of each user, involving honeywords and a real password unifiedly, are briefly named as the user’s sweetwords. Any login attempt with a honeyword from the user’s sweetwords to the system triggers an alarm to inform the system administrator to detect the leakage of the password file. A HBA technique depends on the dual-server structure that consists of the main server-auxiliary server pair. In general, HBA with N users $< U_1, U_2, \cdots, U_N >$ (where U_i is the username of the i-th user and p_i denotes the password of i-th user) operates as follows. At the registration phase, the users send their login credentials (i.e., U_i and P_i) to the main server (MS). After receiving the login credentials $< U_i, P_i >$, for each user, MS creates the sweetword list, $X_i = (x_{i1}, x_{i2}, \cdots, x_{ik})$, employing $\mathsf{HGA}(k, P_i)$. X_i contains $k-1$ honeywords and a real password P_i where k denotes the number sweetwords of each user and precisely one of the elements in the list X_i is the real password. Let a random index c_i denotes the position of U_i’s real password (sugarword) P_i in the list X_i. Here, $x_{i,c_i} = P_i$, and each x_{ij} ($1 \leq j \leq k$ and $j \neq c_i$) is U_i’s honeyword. The database of MS stores[1] the username u_i and hashed sweetwords as $< u_i, (w_{i1}, w_{i2}, \cdots, w_{ik}) >$ where $w_{ij} = H(x_{ij})$ and $H()$ represents a one-way cryptographic hash function, such as SHA-256

[1]Note that in a traditional password based authentication system $< U_i, H(P_i) >$ pair is kept for each account, whereas for this technique $< U_i, W_i >$ tuple is stored in the database, where $W_i = (w_{i1}, w_{i2}, \cdots, w_{ik})$.

or MD-5, while the correct index c_i of each user u_i is maintained in another physical location (i.e., an auxiliary secure server) called as HoneyChecker (HC). We highlight the following statement from Juels and Rivest's paper: "*There is no place on computing environment (so in* MS*) where anyone can securely store additional secret information with which to prevent the adversary*" [19, p. 146]. Thus, HC is a part of authentication responsible for the storage of the correct hashed password index for each account and assisting with the detection of the compromised password files. In addition, it has a minimalist design that does not permit any secure computation inside and is a separate hardened computer system where MS accesses for two following cases. (i) At registration phase, setting the index of the sugarword for U_i at position j in the list X_i, which means $c_i = j$. (ii) Checking if the index of sugarword for U_i is at position j. It reports the result of this check to MS and also triggers an alarm if check fails at authentication phase. In the right part of Fig. 2.1, we show the items stored at MS and HC. Secret information stored in HC is useless, unless paired with the database of MS. Therefore, even if both MS and HC are compromised, the security of the system is solely reduced to the level of security before honeywords and HC are included. As mentioned in [19], an encrypted and authenticated communication channel for secure data transmission connects MS with HC. Therefore, storing password hashes in one server and c_i in the other server (i.e., in HC) provides a simple method of distributed security and makes it more difficult to compromise the system as a whole.

As shown in Fig. 2.1, participation of both MS and HC is needed to successfully authenticate the user into the system. Assume a user submits his/her login credentials $< U_i, P_i' >$ to the system. MS first checks whether the corresponding entry for that username in the credentials database or not. If U_i exists, MS computes the hash of the user-supplied password $H(P_i')$ and tries to find a match in hashes of sweetwords, W_i. If $H(P_i')$ is not equal to any w_{ij} in U_i's W_i, the login request is denied due to wrong password submission. If $H(P_i')$ matches any element of W_i (Assume $H(P_i')$=w_{ij}), then MS sends U_i and the matching index $< U_i, j >$ to HC. Eventually, HC checks whether the index of the sugarword for U_i is at position c_i. After checking if c_i =j, it returns the result of this comparison to MS. If the check fails, it returns a reject message and additionally triggers either an alarm, notifying the system operator that the password database may

have been compromised, or other actions depending on the security strategy of the system. Otherwise, it returns an accept message, completing the login procedure successfully. It is significant to notice that the submission of the correct username/password pair will always match with the stored index value at HC, and thus, the legal login attempt will always be allowed.

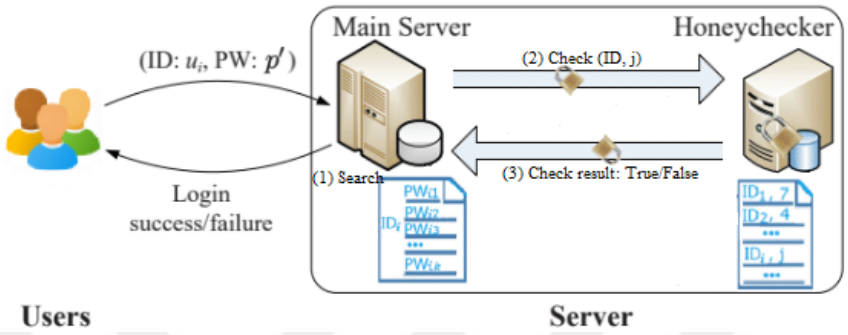

Figure 2.1. Working principle of honeyword-based authentication system

HGAs are divided into following two categories according to whether the password choices of users are influenced by the UI. Preferably, it is better to employ a HGA that belongs to legacy-UI category to boost usability.

- Legacy UI: The user only submits his/her login credentials (U_i, P_i) as the input to the system such as chaffing-by-tweaking in [19]. Users know nothing about the use of honeywords, thus this category imposes no additional memory responsibility on the user.
- Modified-UI: In addition to login credentials (U_i, P_i), Modified-UI requires the users to provide an additional entry that helps the generation of honeywords or the users are influenced by the UI for the password choice. However, this category imposes more memory responsibility on the user.

HBA techniques depend on the assumption where an attacker obtains MS files, inverts hashed passwords, but HC is not compromised and the communication between MS and HC is secured. When the password file has plundered from MS and obtained plaintext passwords from their hash values, the security of the technique immediately depends on the adversary's ability to guess the user's real password from a list of sweetwords. The adversary should recognize the real passwords of the users among

k sweetwords with a probability, not more than $1/k$ since each sweetword should be equivalently likely as a probable password from the adversary's perspective. The recommended k value is 20 to force the adversary to use honeywords at $(k-1)/k = 95\%$ in an ideal honeyword system design during login attempts.

2.2. Review of Bloom filters

The Bloom filter is a widely used and extremely space-efficient probabilistic data structure that is used to test whether an element is a member of a set and designed by Burton H. Bloom in 1970 [37]. It is a bit vector and does not store the actual element in its internal structure, that will consume huge space. Instead, it offers a compact probabilistic way to represent a set that may incur false positives (wrongly reporting that a non-member element to be part of the set even if it was not added), but never result in false negatives (exhibiting an added element to be absent from the set). The query response time of Bloom filter is also incredibly fast, and it is in $O(1)$ time complexity using the space savings. These make Bloom filters useful for many distinct sorts of tasks that cover lists and sets. The fundamental operations cover inserting elements to the set and querying for element membership in the probabilistic set representation. The removal of elements is not supported by the classic Bloom filter; but, several variants have been built that additionally support removals such as [38, 39]. The accuracy of the Bloom filter depends on the filter-size, the number of hash functions used and the elements inserted into the filter. Figure 2.2 demonstrates the false positive probability of Bloom filter with respect to the number of hash functions and filter-size (in terms of number bits per set item). Horizontal axis represents the number of hash functions and the vertical axis represents the probability of false positive (in log scale). Starting from the top to bottom, Fig. 2.2 depicts a Bloom filter with distinct size ranging from 10 to 40 bits per item. For instance, B40 represents a Bloom filter with 40 bits per item. Increasing the amount of allowed space per element (going from top to bottom in Fig. 2.2) reduces the false positive rate for the same number of hash functions. Besides, false positive rate can be decreased to a certain limit by adding more hash functions. The curves also demonstrate the trend that increasing the number of hash functions up until some point, for a fixed bits-per-element, decreases the false positive rate, but after some point, increasing the number of hash functions increases the false positive rate. For instance, in the case of a Bloom filter with

20 bits per item (curve third from the top), the false positive rate is at the minimum using 13 hash functions. Using more hash functions does not importantly reduce the rate of false positive.

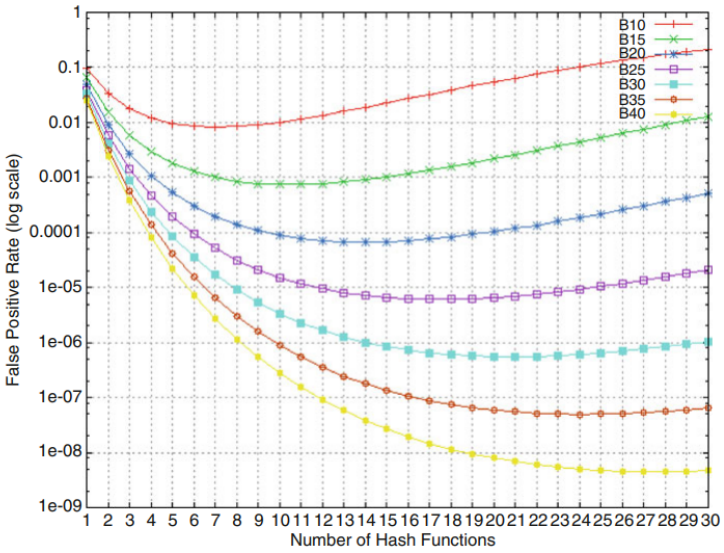

Figure 2.2. The plot relating the number of hash functions used and the false positive rate in a Bloom filter.

3. OUR PROPOSED SCHEME

In this section, we introduce our scheme that is simple, but efficacious and practical to enhance the security of hashed passwords. Cracking cannot be launched even if the users have registered with weak or PII-based passwords. Our scheme masks each hashed sweetword, thus an adversary who steals the files of MS cannot obtain any information about sweetwords. Our scheme is also robust against failover cases by utilizing Bloom filters. Overall, we suggest a novel honeyword strategy that can be easily integrated with legacy systems while preserving HC's interface extremely simple.

3.1. Registration

Algorithm 1 presents the registration procedure of our proposed scheme. First, a username/password pair $< U_i, P_i >$ is submitted by the user who wants to register. MS first checks whether U_i already exists in secured sweetwords table T_1 (see Table 3.1) and if exists, it returns a *Reject* message that the given username is already in the system and requests a new username/password pair. Otherwise, MS generates $k-1$ number of honeywords using $\mathsf{HGA}(k, P_i)$, a unique salt and k random values r_{ij} where $|r_{ij}| = \lambda$ and $j = 0, \ldots, k-1$. The random r_{ij} values denote freshly generated masking values for each sweetword. After the generation of sweetwords, the secured sweetword values σ_{ij} are computed by $H(P_i||S_i) \oplus r_{ij}$. The hash values of the masks are first computed and then stored in Bloom filter (see Table 3.2).

MS sends a message $< U_i, \vec{\boldsymbol{R}}_i >$ to HC in which the first element of $\vec{\boldsymbol{R}}_i$ is the *sugarmask* R_{i0} value (see Table 3.3). The remaining masks are the honeymasks that only appear when a *honeyword* is submitted for that username. MS completes the registration procedure by randomizing the order of elements of secure password vector $\vec{\boldsymbol{\sigma}}_i$ and then storing $< U_i, \vec{\boldsymbol{\sigma}}_i, S_i >$ tuple in T_1 and $< U_i, B_i >$ in T_2 . Note that as suggested by [19, p.152] a special server can be employed to store Bloom filters, or they can be kept in encrypted form as they are only needed in failover mode (see Section 4.5 for security discussions). In such a case, the keys can be kept in a secure module (such as HSM [40] or any other mechanism of equivalent general functionality) and the needed filter can be decrypted in failover mode.

Algorithm 1: Registration Procedure of the *Panacea* Protocol

Input: U_i, P_i

Output: $S_i, B_i, \vec{\sigma}_i, \vec{R}_i$, $Accept/Reject$

if U_i *exists* **then**
    return $Reject$

Pick $S_i \leftarrow_R \{0,1\}^{\ell}$ and $r_{i0} \leftarrow_R \{0,1\}^{\lambda}$

Compute $\sigma_{i0} \leftarrow H(P_i||S_i) \oplus r_{i0}$

for *j=1 to k-1* **do**
    Generate P_{ij} using $\mathsf{HGA}(k, P_i)$ Compute $\sigma_{ij} \leftarrow H(P_{ij}||S_i) \oplus r_{ij}$

for *j=0 to k-1* **do**
    Compute $R_{ij} \leftarrow H(r_{ij})$
    Feed Bloom B_i with R_{ij}

$\vec{\sigma}_{temp} \leftarrow < H(P_i||S_i) \oplus r_{i0}, H(P_{i1}||S_i) \oplus r_{i1}, \cdots, H(P_{ik-1}||S_i) \oplus r_{ik-1} >$

$\vec{R}_i \leftarrow < R_{i0}, R_{i1}, \cdots, R_{ik-1} >$

Send $U_i, \vec{R}_i$ to T_3 // T_3 is located in HC

Obtain $\vec{\sigma}_i$ from $\vec{\sigma}_{temp}$ // by permuting the order of the elements of $\vec{\sigma}_{temp}$

Store $U_i, \vec{\sigma}_i, S_i$ in T_1

Store U_i, B_i in T_2 // T_1 and T_2 are located in MS

return $Accept$

3.2. Login

Algorithm 2 presents the login procedure of our scheme. First, a username/password pair $< U_i, P_i' >$ is submitted[2] then, MS creates an authentication session for the requested user and launches the standard login routine by first checking whether the submitted username exists in T_1 or not. If not, a username or password entry error is returned and the login request is denied. If U_i exists, the hash value of the entered password P_i' is computed by $H(P_i'||S_i)$. Then, this value is XORed with each element of $\vec{\sigma}_i$ to retrieve the login vector $\vec{R}_i'$ i.e., $H(P_i'||S_i) \oplus \sigma_{ij}$ where $j = 1, \ldots, k$. Note that the elements of $\vec{R}_i'$ are in random order. Regardless of whether the given password P_i' is a sweetword or not, MS forwards the computed $\vec{R}_i'$ in each login attempt to HC. This variant that notifies HC for each login attempt is also suggested by Juels and Rivest in [19, p. 148] and helps the

[2] P_i' denotes that the submitted password is either the user's real password, or a honeyword or just a guess.

observation of whether a password guessing attack is in progress.

HC gets $< U_i, \vec{\boldsymbol{R}}_i >$ pair from T_3. In login procedure there are three possible cases: (i) HC takes the first element of $\vec{\boldsymbol{R}}_i$ i.e., R_{i0} and compare with the elements of the received $\vec{\boldsymbol{R}}'_i$. If a match is found then it returns Accept. (ii) If a match is not found for the first element then HC starts comparing each of the remaining elements of $\vec{\boldsymbol{R}}_i$ with $\vec{\boldsymbol{R}}'_i$. If a match is found, an Alarm is triggered meaning that a honeyword is submitted and necessary actions are taken depending on the system's defence policy. (iii) If no match is found, then it returns Reject message meaning that P'_i is neither the user's password nor one of the user's honeywords.

Algorithm 2: Login Procedure for the *Panacea* Protocol

Input: U_i, P'_i
Output: Accept/Reject/Alarm

if U_i *exists* **then**
  Get $\vec{\boldsymbol{\sigma}}_i$ and S_i corresponds to U_i from T_1
  Compute $h \leftarrow H(P'_i||S_i)$
  for *j=0 to k-1* **do**
    Compute $r'_{ij} \leftarrow h \oplus \sigma_{ij}$
    Compute $R_{ij'} \leftarrow H(r'_{ij})$
  $\vec{\boldsymbol{R}}'_i \leftarrow < R'_{i0}, R_{i1'}, \cdots, R'_{ik-1} >$
  Send $< U_i, \vec{\boldsymbol{R}}'_i >$ to HC
  /* the following procedure will be performed by HC */
  for *j=0 to k-1* **do**
    if $R_{i0} == R'_{ij}$ **then**
      return Accept
  for *t=1 to k-1* **do**
    for *j=0 to k-1* **do**
      if $R_{it} == R'_{ij}$ **then**
        return Alarm
      else
        return Reject
else
  return Reject

3.3. Password Update

When a user U_i changes her password from P_{old} to P_{new}, MS takes the following procedures: (i) Generate $k-1$ honeywords by using $\mathsf{HGA}(\mathsf{k}, \mathsf{p_{new}})$ and new masks r_{ij} for each sweetword. (ii) Compute new secured sweetword vector $\vec{\boldsymbol{\sigma}}_{new}$ and feed B_{new} with new $H(r_{ij})$s. (iii) Replace $\vec{\boldsymbol{\sigma}}_{new}$ with $\vec{\boldsymbol{\sigma}}_{old}$ in $\mathsf{T_1}$ and replace B_{old} with B_{new} in $\mathsf{T_2}$. (iv) Notify and send new $\vec{\boldsymbol{R}}_{new}$ to HC to update user's entry in $\mathsf{T_3}$.

3.4. Procedure of failover mode

As honeyword schemes depend on the dual-server structure that consists of the MS-HC pair, the communication between the servers might fail or HC might simply be unreachable for a short period due to technical hitches or attacks. Although such cases might rarely occur, the system is expected to be still in operational in "failover" cases. Our scheme continues to work in a possible failover case by employing Bloom filter structure.

Algorithm 3 depicts the steps of the failover procedure. In general, the first seven steps of the failover mode is the same as the login procedure that is executed by MS. After that, as HC is unreachable, the authentication request is done using the user's Bloom filter. Note that in the registration procedure for each user a Bloom filter is fed with hashed masked values i.e., $\vec{\boldsymbol{R}}_i \leftarrow < R_{i0}, R_{i1}, \cdots, R_{ik-1} >$. MS searches each elements of $\vec{\boldsymbol{R}}_i'$ on B_i. If B_i returns TRUE for any element of $\vec{\boldsymbol{R}}_i'$, then the login request is accepted and the found mask with corresponding username are buffered to be sent later for further evaluation when HC is available. If no element of $\vec{\boldsymbol{R}}_i'$ is found then the login request is rejected which means P_i' is neither the user's password nor one of the user's honeywords. When HC receives the buffered masks and usernames, it checks the correctness of them and take necessary security actions similar to the login procedure.

Algorithm 3: Failover Mode for the *Panacea* Protocol

Input: U_i, P_i'

Output: U_i $\vec{R}_i'$, Accept/Reject

if U_i *exists* **then**
  Get $\vec{\sigma}_i$ and S_i corresponds to U_i from T_1
  Compute $h \leftarrow H(P_i'||S_i)$
  for *j=0 to k-1* **do**
    Compute $r_{ij}' \leftarrow h \oplus \sigma_{ij}$
    Compute $R_{ij}' \leftarrow H(r_{ij}')$
  $\vec{R}_i' \leftarrow < R_{i0}', R_{i1}', \cdots, R_{ik-1}' >$
  Get B_i corresponds to U_i from T_2
  for *j=0 to k-1* **do**
    if $B_i(R_{ij}') == \mathsf{TRUE}$ **then**
      Buffer U_i and $\vec{R}_i'$ `// to send HoneyChecker when available`
      return Accept
  return Reject
else
  return Reject

3.5. A Simplified Scheme

HBA schemes use multi-server structure that consists of $\mathsf{MS}-\mathsf{HC}$ pair. There are other commercialized multi-server password-based authentication approaches [41, 42] that use distributed cryptography which leads to a huge amount of computing and communication overhead. They also require major changes to the server-side systems. In contrast, only a simple communication is needed for $\mathsf{MS}-\mathsf{HC}$ pair of the honeyword scheme. If the failover is not considered as an issue then we simplify *the Panacea* not to cover a failover scenario and the honeyword generation. The architecture suggested here is simple-to-deploy and splits password-related secrets across servers to hide passwords fully in the case of a server breach. The main structural difference from the *Panacea* is that, in this scheme the honeyword generation and storing T_2 are not needed but participation of both MS and HC is required to log the user into the system. Thus if

Username	Secured Sweetword	Salt
U_1	$\vec{\boldsymbol{\sigma}}_1$	S_1
U_2	$\vec{\boldsymbol{\sigma}}_2$	S_2
...	...	...
U_N	$\vec{\boldsymbol{\sigma}}_N$	S_N

Table 3.1. T_1 - Secured Sweetwords Table

Username	Bloom filter
U_1	B_1
U_2	B_2
...	...
U_N	B_N

Table 3.2. T_2 - Bloom Filter Table

Username	Hashed Masks
U_1	$\vec{\boldsymbol{R}}_1$
U_2	$\vec{\boldsymbol{R}}_2$
...	...
U_N	$\vec{\boldsymbol{R}}_N$

Table 3.3. T_3 - Hashed Masks Table

HC becomes somehow unreachable (i.e., failover case), all login requests are temporarily disallowed until reaching HC again. Below, we present the working principle of the simplified *Panacea* that provides absolute flatness, and resistance to offline password inversion, targeted password guessing and multiple system intersection attacks.

After $< U_i, P_i >$ is submitted by the user at the registration phase, MS first checks whether U_i already exists in its database and if exists, it returns a Reject message and requests a new username/password pair. Otherwise, MS stores $< U_i, \sigma_i, S_i >$ tuple where $\sigma_i \leftarrow H(P_i||S_i) \oplus r_i$ and r_i is the freshly generated mask value of P_i and $|r_i| = \lambda$. Accordingly, HC maintains only $< U_i, H(r_i) >$ pairs in its database. Hence, hashes of real passwords are never stored neither in MS nor in HC.

At login phase, after $< U_i, P_i' >$ is submitted, MS computes $r_i' \leftarrow H(P_i'||S_i) \oplus \sigma_i$. Then, it sends $< U_i, H(r_i') >$ pair to HC for eventual decision. Finally, HC checks whether $H(r_i')$ is equal to $H(r_i)$. If $H(r_i')$ equals to $H(r_i)$, HC returns an Accept message, otherwise the login request is denied and HC returns a Reject message, meaning that P_i' does not belong to U_i.

In case of a password update, σ_i^{new} is computed by $\leftarrow H(P_i^{new}||S_i) \oplus r_i^{new}$ and σ_i^{old} is replaced with σ_i^{new} in the database of MS. Accordingly, MS sends $< U_i, H(r_i^{new} >)$ to HC to update user's entry in its database.

4. SECURITY ANALYSIS

In this section, we start with describing each essential security property and then prove the security of our proposed model. In what follows, we analyse the security of our scheme and compare with the existing honeyword schemes. We follow the general assumption that the security of HC and its communication with MS are ensured [19, 33, 34, 29, 35, 43, 36]. Let $\mathscr{O}_H$ be an oracle such that given a hash function H and a hash output $y = H(P)$, it inverts the pre-image for some inputs P in probabilistic polynomial time (PPT). Let $\mathscr{A}$ be an adversary and $\mathscr{A}^{\mathscr{O}_H}$ who accesses $\mathscr{O}_H$ for cracking hashed passwords[3] . In addition, unlike many other literature works such as [19, 33, 44, 45] we did not limit the adversary's strength in performing trawling guessing attack[4] only and kept targeted guessing attack within the scope of this study.

Lemma 4.1. *Given some secured sweetwords (i.e., $\sigma = H(P|S) \oplus r$) and corresponding salt values (i.e., S), an attacker $\mathscr{A}^{\mathscr{O}_H}$ who has access to $\mathscr{O}_H$ cannot obtain the plaintext forms of the sweetwords (i.e., P).*

Proof. Let h be an hash output $h = H(P|S)$ for a given P and $\mathbf{H}$ be the set of hash outputs, $\mathbf{r}$ be the set of random masks freshly generated for each sweetword and Σ be the set of secured sweetwords. Considering the scheme given in Algorithm 1 (see steps 4 and 7) similar to the one-time-pad setting:

$$Pr[\Sigma = \sigma \mid \mathbf{H} = h] = Pr[\mathbf{H} \oplus \mathbf{r} = \sigma \mid \mathbf{H} = h] = Pr[h \oplus \mathbf{r} = \sigma] = Pr[\mathbf{r} = h \oplus \sigma] = \frac{1}{2^{\lambda}} \quad (4.1)$$

As Equation 4.1 holds for all distribution and all h, we have that for every probability distribution over $\mathbf{H}$. For all $h_0, h_1 \in \mathbf{H}$ and every $\sigma \in \Sigma$ we have:

$$Pr[\Sigma = \sigma \mid \mathbf{H} = h_0] = Pr[\Sigma = \sigma \mid \mathbf{H} = h_1] = \frac{1}{2^{\lambda}}$$

Therefore, the attacker can only obtain h with a probability of $\frac{1}{2^{\lambda}}$. As this probability is negligible for a large λ, we can conclude that $\mathscr{A}^{\mathscr{O}_H}$ cannot obtain hash output h and make use of $\mathscr{O}_H$ to revert the hash function to attain plaintext form of the password. □

[3] We also follow the general assumption [19] that adversary has the capability of inverting most or many of the password hashes by an offline cracking method such as brute-force, dictionary and time-memory-trade-off attacks.

[4] This type represents the adversary that does not have access to the user's PII.

4.1. Offline Password Inversion

In conventional password-based systems, the authentication server stores the passwords in salted-hash form. This poses no great difficulty for the attacker to recover them with an overwhelming percentage using advanced cracking algorithms [14, 46]. Therefore, an adversary who has access to hash file can employ cracking attacks to obtain password P given $H(P)$.

Theorem 4.1. *An adversary $\mathscr{A}^{\mathscr{O}_H}$ who has access to* MS *can perform an offline inversion attack with only a negligible probability.*

Proof. An $\mathscr{A}^{\mathscr{O}_H}$ compromises MS files can have access to T_1 and T_2 tables. Note that as T_2 is only needed in failover case, it can be kept in a secure form (e.g. can be encrypted and the keys stored in a HSM device). From T_1, $\mathscr{A}^{\mathscr{O}_H}$ can access secured sweetwords $\sigma_{ij} = H(P_{ij}||S_i) \oplus r_{ij}$ value for a user U_i. By Lemma 4.1, $\mathscr{A}^{\mathscr{O}_H}$ cannot obtain the sweetwords. To this end, $\mathscr{A}^{\mathscr{O}_H}$ can either have a guess on r_i (with probability of $\frac{1}{2^\lambda}$) or on the password itself (among the whole password space) without having any additional advantage from $\vec{\sigma}$. It is equivalent to say that compromising T_1 does not give any advantage to $\mathscr{A}^{\mathscr{O}_H}$ to reveal the plaintext form of the passwords. □

4.2. Flatness

Flatness property is first defined in [19] based on only honeyword generation technique and ensures that after compromising MS of a honeyword scheme, each sweetword should be equivalently likely as a probable password from the adversary's point of view.

Definition 4.1 (Perfect and Approximate Flatness)**.** *Let z be the probability of recognising the real password for an adversary $\mathscr{A}^{O_H}$. A honeyword scheme is said to be "perfectly flat" if $z = 1/k$ and "approximate flat" if z is very close to $1/k$ (for a given uniform distribution).*

In [19], each sweetword is stored in hashed form. Therefore, by Definition 4.1, an adversary having $\mathscr{O}_H$ can obtain the sweetword list X_i in plaintext form that contains $k-1$ honeywords and a real password (sugarword). Thus, the probability of finding the real password among sweetwords by random-selection is $1/k$.

If the algorithm is not flat enough, $\mathscr{A}^{O_H}$ can easily identify the correct password by eliminating fake ones. We emphasize that flatness is strongly connected to the users'

password choice. Thus, none of the existing legacy-UI techniques achieves perfect flatness due to several issues such as related to correlational hazards and recognizable patterns in passwords. For example, Jules and Rivest's legacy-UI methods cannot achieve perfect flatness, and Wang et al. [29] shows correct passwords can be picked out with a success rate of up to 49% using advanced trawling-guessing attackers such as Markov [14] and PCFG [46]. Therefore, a more generalised definition of flatness is required to cover different types of evolving solutions. We extend the flatness definition as follows.

Definition 4.2 (Absolute Flatness)**.** *A honeyword scheme is "absolute flat" if z is negligibly small $z \approx \varepsilon \ll 1/k$, despite the presence of $\mathscr{A}^{\mathscr{O}_H}$.*

Theorem 4.2. *Given Lemma 4.1, our scheme achieves absolute flatness.*

Proof. In registration procedure of the Panacea protocol, each sweetword is masked (see Algorithm 1's step 4 and 7). For all PPT adversaries

$$\left| adv[\mathscr{A}^{\mathscr{O}_H}(1^\lambda)] - adv[\mathscr{A}(1^\lambda)] \right| < \varepsilon$$

then the scheme satisfies absolute flatness property, where ε is negligible. In other words, the adversary $\mathscr{A}^{\mathscr{O}_H}$ has only negligible advantage over any PPT adversary $\mathscr{A}$ who has not necessarily access to $\mathscr{O}_H$. Thus, by Lemma 4.1, $\mathscr{A}^{\mathscr{O}_H}$ can only obtain a sweetword with a probability of $\frac{1}{2^\lambda}$ to compare and distinguish the sugarword, which is negligible and much smaller than $1/k$. Therefore, our scheme provides absolute flatness. □

Note that our scheme's security is at worst reduced[5] to $1/k$-flatness when the adversary compromises T_1 and somehow able to crack T_2. We highlight again that this is very unlikely as T_2 is secured (as defined in Sect. 3).

4.3. Targeted Password Guessing

A targeted guessing attacker exploits user's Personally Identifiable Information (PII) like name and birthday to find username-real password pairs. This attitude of users in choosing a password is continuously more vulnerable, as their PII can be easily collected through social engineering attacks and unending data leaks [48, 24, 49, 27]. For instance, in October 2017, a data breach revealed the personal information of 3 million Yahoo users, including their name, phone number, and date of birth [24]. Similarly, in July 2017,

[5]This is the best security level achieved by the existing works such as [19, 33] which depends on the generation of honeywords and flatness (i.e., $1/k$-flatness). Besides, in any case, Bloom filters do not reveal passwords and masks directly [47].

Equifax –the largest credit reporting agency in America– leaked PII data approximately half of the US population, with names, date of births, SSN [49]. Figure 4.1 shows that the majority of users form passwords using their own PII and some of these real-word password databases contain more than 50% of users' personal information [15, 29]. This can make users and organizations the target of many cyber attacks, and therefore sound authentication strategies should take this user behaviour into account. The authors in [29] also show that the real password can be distinguished from honeywords generated by legacy-UI HGAs in [19] with a success rate between $56.8\% - 67.9\%$ by making only *one* guess (when $k = 20$), which is significantly higher than $1/k$. These all show the importance of designing an enhanced version of the practical legacy-UI-based honeyword scheme to prevent such elaborated attack models.

Mitigating targeted password guessing attacks using any HGA that belongs to modified-UI category is also a challenging task. While honeywords produced by HGAs may not contain user's PIIs, the original user-selected password is more likely to be associated with the user's personal information [15, 29]. The honeywords produced by this category depends on the user's original password, which causes a specific connection between the honeywords and the original password. Overall, the usage of PII-based passwords greatly helps an attacker in recognizing the real password from honeywords generated from any category.

Theorem 4.3. *Our scheme resists targeted password guessing attack.*

Proof. Considering our scheme, by Lemma 4.1, a successful password break can be performed with only a negligible probability. Therefore, the adversary $\mathscr{A}^{\mathcal{O}_H}$ cannot obtain any information about users' password choices even if $\mathscr{A}^{\mathcal{O}_H}$ has all types (name, birthday, etc.) of personally identifiable information of all users. □

4.4. Multiple System Vulnerability

According to a recent survey [31], 65% of users tend to reuse the same password (or with only slight modifications) for accessing distinct web services/accounts. Statistics greatly confirm the importance of the password reuse problem [50, 30, 51], and organizations need to take action to reduce the risk of multiple system vulnerability. This creates different opportunities to the adversaries when attacking various web services

Typical usages of PII	PII-Tianya (430,966)	PII-Dodonew (161,517)	PII-12306 (129,303)	PII-CSDN (77,216)	PII-Rootkit (69,330)	PII-000web-host(153,390)	PII-ClixSense (2,222,045)	PII-Yahoo (16,307)	PII-QNB (77,799)
Name	9.13%	23.82%	23.83%	16.71%	4.32%	12.66%	9.14%	6.95%	8.52%
Birthday	20.80%	16.37%	18.75%	19.16%	1.57%	7.36%	7.07%	5.20%	11.85%
Email prefix	6.31%	8.60%	6.61%	8.65%	3.75%	7.89%	5.14%	3.86%	4.91%
Username	3.09%	10.53%	10.12%	7.46%	3.45%	5.74%	5.29%	3.82%	7.61%
Phone #	1.18%	1.00%	0.89%	1.43%	–	–	–	–	1.92%
NationalID	0.81%	0.39%	0.71%	0.12%	–	–	–	–	0.13%
Total PII usages (all above)	36.95%	51.43%	50.71%	46.87%	12.76%	29.94%	24.81%	18.76%	27.16%

Table 4.1. The existence of PII-based passwords (with PII types and cumulative usage) in *nine* previously leaked real world password databases

assuming the collision probability of honeywords corresponding to the same password chosen for various web services or organizations is zero since HGAs characteristically generate a distinct set of honeywords at each run. Thus, if an adversary exposes distinct password databases that have been used the legacy-UI category algorithm for honeyword generation belonging to different services/organizations and gets the intersection of sweetwords, then the intersection high likely reveals the user's real passwords when the user reuses the same password (or its slight variant). Hence, even if a honeyword-based system is perfectly flat, the intersection attack completely defeats the detection ability and dramatically limits the overall effort of distributed honeyword security architecture.

Note that HGA that belongs to modified-UI category either modifies the user's password or requires the user to modify her original password to prevent intersection attack in multiple systems. Well-known modified-UI procedures are take-a-tail [19], random-pick [19], modified tail [52], caps key based [52], paired distance protocol (PDP) [44] and append secret model [35]. They all bring the burdensome on the user (e.g., usability overhead) by requiring the user to remember the modified password. Some of these procedures need the user to modify her original password. The modified password may have a relationship or correlation to the original password. Such cases can help the adversary to explore the characteristics of original user passwords, making the system vulnerable multiple system intersection attack. Similarly, a loophole in the design principle of a scheme can open new attack surfaces. One of the most important examples showing this case is Akshima et al.'s attack [35] which demonstrates that the modified-UI PDP is not secure against the multiple system intersection attacks.

Theorem 4.4. *Based on Lemma 4.1, our scheme is secure against multiple system intersection attack.*

Proof. When an adversary compromises two (or more) password files belonging to different organisations, he has no advantage in finding the real passwords on either system.

Let consider two different organisations have distinct databases A and B. Assume U_i uses the same password in both databases. Also, assume that the attacker captures both databases and the secured sugarwords of U_i. In our scheme, each database generates different masks and salt values resulting different secured sweetword values (i.e., $\sigma_A \leftarrow H(P_i||S_A) \oplus r_A$ from A, and $\sigma_B \leftarrow H(P_i||S_B) \oplus r_B$ from B). Therefore, by Lemma 4.1, $\mathscr{A}^{\mathscr{O}_H}$ cannot launch a successful attack on σ_A and σ_B to obtain the password and *the Panacea* resists the intersection attack in multiple systems. □

4.5. Handling the Failover

A failover refers to a case when HC is somehow not reachable. Logins should continue more-or-less as usual even if HC is unavailable [19]. During a failover situation, honeywords can be used as acceptable passwords temporarily. Denial-of-service attacks (DoS) is prevented resulting from attack on the HC or the communications between MS and HC [19]. Some recent proposals such as [34, 35, 43, 36, 44] have not focused on this important criterion. They temporarily disallow login requests until HC becomes available again. Besides, they require critical secrets that must be returned by HC to MS for hash computation to complete the login request. This leads three main problems: (i) extra communication overhead (more queries needed to HC for a normal login) (ii) fail to work without the dependency of HC (iii) security issues related to returning critical secrets. Storing critical secrets in HC cannot guarantee security against any threat, as the secret is returned to MS for computing or recovering the hash value of the passwords [53]. Therefore, having only one-round transmission (from MS to HC) provides safer solution with lower latency.

Theorem 4.5. *The Panacea is resistant to failovers.*

Proof. Considering Algorithm 3, we utilise Bloom filters to store all r_{ij}s of each user. Here, we assume that Bloom filters are kept in encrypted form as they are only needed in failover mode. During a failover, after accessing T_2 and decrypting the requested user's Bloom filter, MS queries the Bloom filter after computing $\vec{\boldsymbol{R}}_i$ vector i.e., $r_{ij} \leftarrow H(P_i||S_i) \oplus \sigma_{ij}$ to confirm the login request. The *Panacea* reverts to the current practice even if anyone uses a honeyword to login after a failover case by buffering messages on MS for later forwarding to and processing by HC. □

4.6. Security of the Simplified Scheme

Theorem 4.6. *Our simplified scheme provides absolute flatness and security against offline password inversion, targeted password guessing and multiple system intersection attacks without having honeyword generation and the failover resistance.*

Proof. We prove that *the Panacea* successfully satisfies Theorem 4.1, Theorem 4.2, Theorem 4.3, and Theorem 4.4 respectively. The simplified scheme employs similar procedure for the storage of passwords (i.e., $\sigma_i \leftarrow H(P_i||S_i) \oplus r_i$) and login requests (i.e., $r_i \leftarrow H(P_i'||S_i) \oplus \sigma_i$). Therefore, it also proves these theorems. Rather than taking the risk of the adversary's intrusion into the system, we remove the honeyword generation when the failover is not considered as an issue since the adversary $\mathscr{A}^{\mathscr{O}_H}$ cannot make use of any σ. □

5. EXISTING HONEYWORD SCHEMES AND THEIR LIMITATIONS

In this section, considering the security parameters explained in Section 4, we argue the limitations and vulnerabilities of existing honeyword schemes in chronological order while recommending probable enhancements for the schemes and give references if the provided analysis is from existing publications.

5.1. Juels and Rivest's Scheme (2013)

The use of honeywords has many benefits in practice and there is a huge effort for generating a set of k equally probable sweetwords in the existing literature [19, 33, 35, 52, 44, 45, 54, 55, 29]. However, it is still an active research topic and unresolved problem to the research community since passwords are selected by the users, and modeling human behavior is not possible using a set of standardized rules. The scarcity of theoretical evaluation methods and empirical standards for HGAs also limits the assessment of HGAs as well as their comparison with each other. After obtaining plaintext of passwords by inverting password hashes, the security depends on the adversary's ability to guess the user's real password among k sweetwords with a probability, not more than $1/k$. However, the use of honeywords may fail to work for *every* user account and to satisfy its security objectives. Despite having many benefits, primary difficulties regarding flatness property are abridged below in an accurate manner.

- In [32], the authors indicate that user-selected passwords follow the Zipf's law distribution. Thus, although random replacement-based legacy-UI HGA provides adequate protection against DoS attack such as simple model and modeling syntax in [19], they all fail to meet *flatness* property [29].
- Because of Zipf-law distribution of users' passwords in the password file, none of the legacy-UI technique can satisfy $1/k$-flatness. Essentially, trawling guessing attackers can also recognize the real passwords of the users from the sweetword lists with high probability [29, 32, 56].
- Recently, Wang et al. in [29] demonstrates that any legacy-UI HGA, regardless of its design principle, is always prone to the threat of targeted guessing attack.
- Several HGAs such as simple model in [19] and EPM in [35] require a list (or a

file) provided as input to them. However, these papers have not focused that how the computer system securely stores these inputs that are continuously used by the system during registration and password update processes. A secure storage for these inputs is needed, otherwise, it gives a clue to the adversary to in guessing the real password of the users.

Table 5.1 compares existing HGAs regarding other various properties. DoS attack is briefly discussed under the scenario that an adversary or a malicious user know HGA and submit one (or more) of the user's honeywords to the system intentionally to trigger false alarms without compromising password-file. The Denial-of-Service (DoS) attack protection is described as 'weak' or 'strong' where 'weak' shows that the adversary can guess honeywords and deliberately trigger false alarms to make the system dysfunctional and 'strong' signifies that the adversary cannot guess any honeyword belonging to a user to lure the system. Failure to meet typo-safety property reduces the usability because false alarms unintentionally or accidentally may be triggered by a legitimate user who submits a honeyword due to a typing error instead of the real password. Table 5.1 also shows that modified-UI approaches are considerably vulnerable to typos and DoS attacks as well as they bring the memory overhead.

Table 5.1. Comparison of HGAs

Algorithm Name	**DoS-protection**	**Typo-Safe**	**Legacy-UI**	**User-friendly**
Tweaking [19]	Weak	No	Yes	Yes
Modeling-syntax [19]	Strong	Yes	Yes	Yes
Take-a-tail [19]	Weak	No	No	No
EPM [35]	Strong	Yes	Yes	Yes
User-profile model [35]	Moderate	Yes	Yes	Yes
Close-number formation [52]	Weak	No	No	Yes
Modified-tail approach [52]	Weak	No	No	Yes
Caps-key [52]	Weak	No	No	No

5.2. Erguler's Scheme (2016)

In [33], Erguler proposes benefiting from existing user passwords in the system to simulate honeywords to improve the honeyword generation challenge without reducing usability. In Erguler's scheme, each username is associate with k indexes which all

represent a real user password available in the system. Thus, the scheme is expected to achieve notably flatness (if the username and the password are not correlated) since all sweetwords of users are user-supplied passwords. MS of the Erguler's scheme stores two tables T_1 and T_2. T_2 consists of hashed password, random index of each password, and salt. In T_1, each username is associated with a honeyindex set that contains k indexes of the user's real password and $k-1$ randomly-selected other users' passwords. Then, the passwords of $k-1$ selected users behave as honeywords. Accordingly, HC maintains the correct index of each user. Since each user at registration phase can just select existing random indexes, the password of any user cannot be selected as the honeywords of preceding users, which will lead to insecurity. Therefore, as stated in Section 4.2. in [33], MS will periodically update honeyindex set of each user such that any user's password could be selected as honeyword of any other user, including her preceding users. *Our Attack:* An adversary can make use of periodical regeneration which will alter the honeywords of all users. When the adversary gains only one version of files at MS, the adversary can delay his attack time such that the attack is one or more update periods later than the disclosure. After the delay, of Erguler's scheme would update T_1, and then each user's sweetwords will alter with high probability. This means any honeyword of a user will no longer be her honeyword with high probability after update. Thus, so long as waiting for one or more update periods before launching the attempts, the adversary can utilize online login attempts to seek out real passwords. As a result, our attack fully breaks Erguler's scheme, and the scheme of selecting honeywords from existing passwords provides no security beyond traditional password-based system.

5.3. Append Secret Model (2018)

The Append Secret Model or ASM proposed in [35] requires an additional user-provided entry (any character string of length 2-4) ℓ_i in addition to the user's password. MS of this technique also produces a random string r_i of default length of 3 where r_i is stored on HC to be returned to MS for use in the hash computation at authentication time. Then, MS computes $x{=}f(p_i|\ell_i|r_i)$ where f is a collision-resistant one-way function and stores H($p_i|x$) in MS's database. For instance (from [35]), let the user chosen password be "abcde", the user chosen extra entry ℓ be "1998", and r_i=\$8d. MS first computes $f(abcde|1998|\$8d)$ = $4e7j@$, then stores H=$(abcde|4e7j@)$. As the

designers of ASM consider, the dictionary attack reveals the user's *abcde4e7j@* and the attacker can get the value x. Then, brute-forcing on x is possible to get the actual input p_i, l_i [35, p. 766]. We list our analysis on the ASM below.

- Authors claim that their model provides multiple system protection (MSP) based on the assumption that offline brute-forcing requires extra computational effort (see Section 6.1 in [35]). However, their claim contradicts with the general assumption of the existing literature that after compromising the password file, the attacker can perform brute-force locally which is done offline to obtain plaintext forms of inputs. A realistic assumption is that most of them can be inverted offline as designing a defense protocol that can make password cracking more difficult is insufficient. Hence, once the adversary $\mathscr{A}^{\mathcal{O}_H}$ gets hashed password file, he also can obtain plaintext forms of sweetwords with or without the extra effort. He then finds intersection of both (or more) systems, the result will reveal the user's actual inputs, making the targeted system system protected by the ASM vulnerable against multiple system intersection attack. Besides, returning system generated random string (e.g., r_i) from HC to MS makes the ASM unable to handle the failover scenario and other issues explained in Section 4.5.
- Authors of the ASM have not focused on the generation and flatness of user-chosen ℓ. In contrary to what the authors state in their paper [35], the ASM cannot satisfy the flatness criterion very well and can suffer from targeted guessing attack because of following possibilities: (i) ℓ may be a PII-based string (ii) there may be a correlation between u_i and ℓ_i or a correlation between p_i and ℓ_i (iii) the adversary can easily recognize user-supplied ℓ due to semantics or content integrity such as *S@f3* or *T0Y*. By assuming individual characters are replaced by any other character/characters, the content integrity of such entries would be broken and the real entry becomes completely obvious. Besides, ASM does not sense any risk if a wrong ℓ (but true user password) is entered for a username and this may facilitate for attackers to launch online guessing attacks (especially targeted online password guessing [15]).

5.4. Superword (2019)

Superword [34] is presented as an improvement of honeywords. One of its design principles is to reduce the success probability of the adversary from $1/k$ to $1/N$ by isolating the direct relationship between a username and the corresponding hashed password in the password file. It deviates from the general path of honeyword techniques by depending on huge number of system generated fake accounts or honeypots instead of honeywords to detect password-file leakages. Below, we present a brief analysis of pitfalls in the Superword. We list our analysis on the Superword below.

- As excessive use of honeypots to support detection makes the Superword system vulnerable to DoS attacks [53], it also makes the use of the Superword system in web or social media sites infeasible, where usernames are often considered public or semi-public. Therefore, violating the main idea behind the use of honeywords that it works for *every user account* both limits the Superword's applicability and its detection ability depending on the fact that attackers can set off an alarm solely by inputting honeypot usernames.
- The adversary $\mathscr{A}^{\mathcal{O}_H}$ can perform offline password inversion attack as mentioned in Section 6.1 in [34], this makes the Superword system vulnerable to (i) username/password pair correlations, (ii) targeted password guessing and (iii) multiple system intersection attacks. For example, assume that the user account "Bob123" has chosen the password "Bob.1998". Here are sample password list derived from leaked password databases [26, 57]: Carbonade123, StrAng3E, bluefalcon@325, L1nk3dINh@2u, bond007, $\cdots$, **Bob.1998**. For cases (i) and (ii), the adversary can easily recognize the password belonging to the username even if the direct relationship between them is isolated. The adversary $\mathscr{A}^{\mathcal{O}_H}$ can also launch online login attacks on reused usernames and passwords until the correct matches are found.
- Following the assumption of Juels and Rivest "the adversary can obtain the password hash files on one system at various times" [19, p. 146], the Superword completely loses its ability to detect the password-file compromise when the password hash file on one system is disclosed in different moments since the Superword generates the honeypot accounts and places them into the password

file in its initialization phase (see Section 4.3 in [34]). In contrast to the authors' claim in [34], the honeypots may not help detection of the presence of online attacks. Further, when the adversary leaks at least two or more different versions of password files, he also can find newly added legitimate usernames and passwords by finding the intersection of password files. As well as the intersection eliminates the honeypots, it also opens a new attack surface for the adversary to match the cracked user passwords with these freshly added legitimate usernames via online login attempts. This attack scenario increases the overall flatness from $1/N$ to 1 since the Superword does not sense any risk if a wrong (but not fake) password is entered against a real username.

5.5. Other Works

In 2017, Chakraborty et al. [44] proposed a modified-UI scheme, named the *Paired Distance Protocol (PDP)*, that uses circular list to help the generation of honeywords. The list is composed of default 36 characters consisting of alphabets and strings in a random order. The PDP computes the paired distance between the elements of user-selected password-tail. The protocol aims to confuse the adversary among a list of sweetwords even if the adversary retrieves the paired distance of a user. The authors of the PDP [52] limits the presence of targeted password guessing attackers and has not focused on this security criterion. We emphasize that the design of the protocol fails to resist targeted password guessing attacks identified in Sect. 4.3 although it requires usability overhead to users. Besides, the authors in [35] attack the paired distance protocol and show that the protocol is not secure against multiple system intersection attacks.

In 2021, Tian et al. [36] proposed a method for protecting hashed passwords by using topological graphic sequences. A series of operations of topological graphic sequences and user passwords are used to produce honeywords. The adversary who has access to the password file cannot recognize the real password among sweetwords, as each sweetword is converted into an unrealistic form. Our analysis on Tian et al.'s scheme [36] is as follows. First, it needs more drastic alterations than honeywords as it violates the design principles of honeywords, explained in [19], by putting too much stress on HC. Two major problems exist in HC design of [36]. First, HC becomes a

single point of failure and also adopts other problems explained in Section 4.5. Second, HC of the proposal in [36] is responsible for both generation of the topological graphic matrix and the management of the topological graphic sequence. This leads to huge computational operations in HC and thus, HC maintains much more secret states breaking the common intention of designing its interface to be extremely simple. Authors in [36] aim to eliminate the semantic meaning of passwords and, thus have less focused on the detection property. Even if an adversary who somehow obtains the password file cannot distinguish real passwords among decoy ones since each password seems unrealistic. Therefore, the proposal in [36] has more in common with distributed password storage schemes as the adversary cannot invert a single password without obtaining the data stored in HC.

6. COMPARISON WITH EXISTING WORKS

We now compare our proposed models with the other works with respect to security and complexity parameters respectively.

6.1. Security Comparison

We compare the state-of-the-art honeyword schemes with our proposed models with respect to (i) offline password inversion, (ii) flatness, (iii) targeted password guessing, (iv) multiple system protection and (v) the existence of failover mode of each scheme. We consider the adversary $\mathscr{A}^{\mathcal{O}_H}$ defined in Sect. 4 has the ability to perform inversion attack on hashed passwords. Thus, the adversary can launch a successful offline password inversion attack on the schemes of Juels and Rivest [19], Erguler [33], Append Secret [35], and Superword [34]. For Juels and Rivest's scheme, the adversary $\mathscr{A}^{\mathcal{O}_H}$ requires kN cryptographic effort in inverting a password from the corresponding hash value stored in password database as each user is associated with a set consisting of hash values of the user's real password and honeywords. In the case of Erguler's scheme and the Superword, the adversary $\mathscr{A}^{\mathcal{O}_H}$ needs N operations to obtain all plaintext words from the respective hash value in the system. In the case of the ASM, $\mathscr{A}^{\mathcal{O}_H}$ requires extra computational effort as explained in [35] in addition to kN operations to obtain actual user inputs. In contrast, our simplified scheme and *the Panacea* provides very strong hash whose (e.g., secured sweetword) password the adversary is unable to invert. Hence, they are both important alternatives to address the offline inversion attack.

Regarding flatness property, research studies [15, 29, 32, 53] show that all generation models may leave traces to an attacker in recognising the real password from the honeywords and none of the legacy-UI generation algorithms achieves perfect flatness. Therefore, the flatness of Juels and Rivest's scheme is affected by many factors such as the composition of the user password and unequivocal pattern incompatibility of a specific website. We show that an adversary can make use of the periodical honeyword update requirement of Erguler's scheme. Therefore, the adversary can obtain all username/real password pairs via online login attack, which increases the overall flatness from $1/k$ (if there is not a correlation between username and real password) to 1. Although the Superword targets $1/N$-flatness where N is number of total users in a system, we show that

their scheme can suffer from username/password correlations. Besides, it is vulnerable to online login attacks under an allowed range if the adversary steals two (or more) distinct versions of password files. This attack case also increases the overall flatness to 1 from $1/N$. Besides, other convincing analysis results regarding flatness property in [53] support that the technique in [34] does not satisfy $1/N$-flatness. We discuss that the authors of the ASM have not focused the user chosen additional entry ℓ. After the adversary $\mathscr{A}^{\mathscr{O}_H}$ obtains plaintext forms of sweetwords, he can easily recognize the real password from the honeywords if the content integrity of ℓ is broken or there is a correlation between the username and ℓ, or the user password P and ℓ. For our proposed schemes as described in Sect. 3 freshly generated masks for each password make our schemes achieve absolute flatness ($\frac{1}{2^\lambda}$). The adversary $\mathscr{A}^{\mathscr{O}_H}$ cannot distinguish the sugarwords from the sweetwords despite all inversion efforts.

Table 6.1. Comparison of schemes after evaluating their security standards.

* denotes that the system belongs to the modified-UI category. MSP means that even if accounts of the same user on distinct systems are leaked, it will not reveal the password. Targeted Password Guessing Protection (TPGP) means that if an adversary somehow obtains PII of any user, he cannot distinguish the sugarword among sweetwords under no circumstances.

Method Name	**Offline Pwd. Inv.**	**Flatness**	**TPGP**	**MSP**	**Failover Mode**
Juels and Rivest [19]	No	$\sim 1/k$	No	No	Yes
Erguler's scheme [33]	No	1	No	No	Yes
Append Secret Model* [35]	No	$\sim 1/k$	No	No	No
Superword [34]	No	$> 1/N$	No	No	No
Simplified *Panacea* (This work)	Yes	$1/2^\lambda$	Yes	Yes	No
The Panacea (This work)	Yes	$1/2^\lambda$	Yes	Yes	Yes

Using PII while building a password helps an attacker recognizes the user's real password from her honeywords. Juels and Rivest's scheme, Erguler's scheme, the ASM, and the Superword are always susceptible to targeted password guessing attacks. Authors in [29] shows that none of the developed algorithms until now for Juels and Rivest's scheme satisfies the expected security level by a large margin. Erguler's scheme cannot resist other correlational threats, nor does it resist attackers who make targeted guessing.

We also show that the adversary can find username/password pairs if PII-based password is used in the Superword system even if the direct relationship between the username and the password is isolated. In case of the ASM, the adversary can perform targeted guessing attacks if ℓ is related to personal information of a user. On the other hand, even if a PII-based password is used in the authentication system that is protected by one of our schemes, the adversary $\mathscr{A}^{\mathcal{O}_H}$ has no advantages to recognize the real passwords from the honeywords. Since masks are stored in HC, when an adversary obtains password files from MS, she cannot reverse a secured sweetword using PII. Similarly, all of these approaches have inherent weakness against multiple system and cannot provide resistance against the attack scenarios such that a user reuse passwords (or highly correlated passwords) on two different systems. Conversely, even if user U_i reuses password P_i in two different systems, an attacker $\mathscr{A}^{\mathcal{O}_H}$ only knows the secured sweetwords of U_i in one system and can not infer the password hash of U_i in another system. Hence, both of our schemes will provide resistance against multiple system intersection attack.

The ASM and the Superword are not designed to have a failover mode and they both return user-related critical secrets from HC for each login request, which will lead to insecurity [53]. HC of the ASM stores system-generated random string (r) to be used in authentication requests. Similarly, HC of the Superword system stores system-generated random salt value to be returned to MS at the time of the login. They both temporarily disallow all login requests when a failover case occurs. The simplified *Panacea* may be seen as a promising and effective alternative to such approaches, where handling the failover is not the issue as they have somehow detection issues ranging from activation of false alarms to flatness vulnerability. On the other hand, *the Panacea* continues to allow login requests by employing Bloom filters even if a failover occurs. Table 6.1 compares the state-of-the-art techniques and shows that *the Panacea* overcomes most of risks and limitations associated with previously proposed honeyword schemes.

6.2. Complexity Comparison

In this part, we compare the storage requirement of our system and the number of queries to HC needed for a normal login with other techniques. Let N be total number of users in each system and M be total number of stored hashed passwords.

Table 6.2. Comparison of the Schemes in Terms of Storage Cost (in bytes) and the Number of Queries to HC.

Method Name	**Total Storage Cost**	**# Queries to** HC
Juels and Rivest [19]	$2NS_U + 32kN + NS_k$	1
Erguler's scheme [33]	$2(N+T)S_U + (k(N+T)+M)S_I + 32M + (N+T)S_k$	2
Append Secret Model* [35]	$2NS_U + 32kN + NS_k + Nr$	2
Superword [34]	$(N+T)S_U + 2(N+T+M)S_I + 32M$	2
Simplified *Panacea* (This work)	$2NS_U + 2(32N)$	1
The Panacea (This work)	$2NS_U + 2(32kN) + NS_B$	1

S_B, S_U, S_I, S_k denote the size of a Bloom filter, the length of username, the length of an index and the size of k in bytes respectively. Let used hash function be SHA-256 that produces a 32 bytes (256-bit) hash value. In this case, 32 bytes of memory space to store a password by converting it into corresponding hashed form is required. Both Juels and Rivest [19], and the ASM [35] require the storage cost of $NS_U + 32Nk$ for maintaining sweetwords' information of all users in MS. Required memory spaces in HC are $NS_U + NS_k$ and $NS_U + NS_k + Nr$ for Juels and Rivest, and the ASM respectively. *The Panacea* maintains Bloom filters to store hashed mask values. The size of Bloom filters depends on the administrator's acceptable false positive rate. For each user in the password file, *the Panacea* stores the passwords in hashed and masked format. Thus, *the Panacea* requires $NS_U + 32Nk + NS_B$ bytes of storage space in MS and $NS_U + 32Nk$ in HC. Note that Erguler's scheme [33]and the Superword [34] also use T honeypots. Therefore, honeyindex technique in [33] requires memory space $(N+T)S_U + k(N+T)S_I + 32M + MS_I$ bytes for MS and $(N+T)S_U + (N+T)S_k$ for HC. Similarly, the Superword requires $(N+T)S_U + NS_I + 32M + MS_I$ bytes of memory space in MS and $(N+T)S_I + MS_I$ in HC. We also assume that each index requires 4 bytes [33].

Erguler's scheme, the Superword and the ASM bring extra communication overhead compared to both Juels and Rivest's scheme and *the Panacea*. They all require two queries to HC for a legitimate login. MS of Erguler's scheme checks whether the account is a honeypot. We assume that HC stores the information about whether an account is a honeypot or not. If HC returns that the account is not a honeypot, MS sends another

honeyindex verification message to HC to complete the login request. On the other hand, MS of both the Superword and the ASM asks HC to return the user secret for the hash value calculation. The Superword returns the salt of users from HC for every login request after the username is found in the database of the MS. After returning the salt value of user, it is concatenated with the user password and the password hash is computed. After the computation operation, MS sends another message to check whether the login is legitimate or not. Similarly, the ASM requires the system-generated random string (r) to start the login routine of each user. After returning r value of a user from HC, the hash of inputs are computed $H(p||\ell||r)$ for the user. MS then sends another message to check whether the sent index matches with the stored index in HC. Therefore, they both require two sequential queries to HC. These will cause the insecurity as well as cause an additional delay in the systems. After finding a match hashed form of the password in the user's sweetwords, Juels and Rivest's scheme sends only one check message to HC for the completion of a legitimate login request. Similarly, after a password is submitted, *the Panacea* require only one query to HC to allow a normal login request.

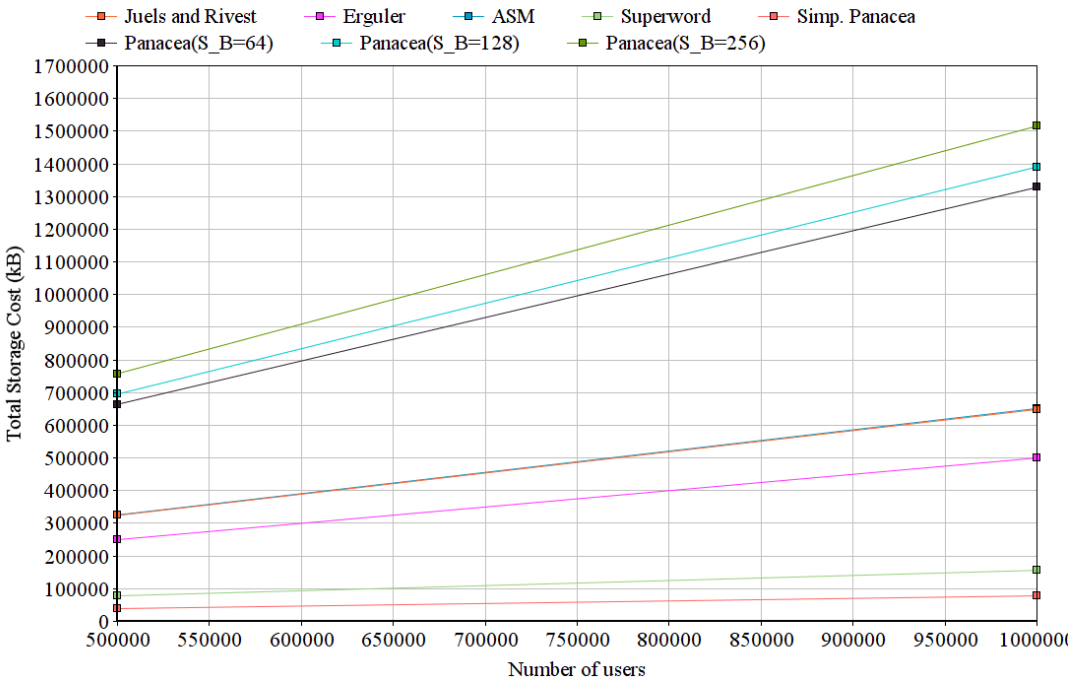

Figure 6.1. Comparison of total storage costs of the schemes

Fig. 6.1 shows the required memory space with respect to the number of users in the system. As stated in Superword [34], we select average lengths of S_U, S_k, S_I are $8, 8$ and 10 bytes. We choose the total number of users in the system as amounts ranging from $500,000$ to $1,000,000$. For Erguler's scheme and the Superword, we use the same assumption in [34] that T honeypots (they suggest the value of N is equal to T) are added in the system. For the *Panacea*, we use three different size in bytes, 64, 128, 256, for Bloom filters. As the size of the filter increases, total storage cost required for *Panacea* also increases but the rate of false positives will be smaller. We summarize the comparison results of total storage costs in *kB* and the number of required queries for a normal login in Table 6.2.

7. CONCLUSION

An adversary who has access to a hash file can brute-force to find plaintext forms of passwords. Thus, after cracking passwords, the adversary can log in successfully and undetected under various attack models even if honeywords are used. In this thesis, we address crucial security and complexity issues of existing schemes and show that the attacker can exploit the own vulnerabilities of these schemes or the inherent weaknesses of the use of honeywords. We show that the requirement of periodically updating honeywords of users of Erguler's scheme leads to a severe security problem. We present that even if the password file is leaked just once, our proposed attack can fully break it and attain all real passwords by online attempts. On the other hand, *the Panacea* provides a complete solution that makes the defensive protocol more robust before making password cracking detectable. The big difference when *the Panacea* is used is that the adversary who obtains MS files cannot brute-force to search for passwords. The use of *the Panacea* thus forces an adversary to attempt compromising either Bloom filters or HC as well. The fact that it equally protects every user account regardless of their password choices without reducing usability before detecting a honeyword submission is its major advantage over the related techniques of honeywords. Besides, analysis results show *the Panacea* balances the drawbacks associated with existing schemes and does reasonably well in satisfying crucial security aspects. Last, we discuss that most of schemes temporarily disallow login requests during a failover and have multiple detection issues. One open question is that should MS and HC pair be used as distributed password storage system if the failover is not considered an issue? Therefore, in order to fulfill both cases we also introduce a simpler variant of *the Panacea* that provides a more efficient solution. Both schemes require no modification to the client-side authentication systems, therefore, they preserve the current usability.

REFERENCES

[1] Joseph Bonneau et al. “The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes”. In: *IEEE Symp. on Security and Privacy* (May 2012), pp. 553–567.

[2] Joseph Bonneau et al. “Passwords and the Evolution of Imperfect Authentication”. In: *Communications of the ACM* 58 (Oct. 2014).

[3] Blase Eric Ur. “Supporting password-security decisions with data”. PhD dissertation. Carnegie Mellon University, 2016.

[4] Hasini Gunasinghe and Elisa Bertino. “PrivBioMTAuth: Privacy Preserving Biometrics-Based and User Centric Protocol for User Authentication From Mobile Phones”. In: *IEEE Transactions on Information Forensics and Security* 13.4 (2018), pp. 1042–1057.

[5] Liam M. Mayron. “Biometric Authentication on Mobile Devices”. In: *IEEE Security Privacy* 13.3 (2015), pp. 70–73.

[6] Wenyuan Xu et al. “Challenge-Response Authentication Using In-Air Handwriting Style Verification”. In: *IEEE Transactions on Dependable and Secure Computing* 17.1 (2020), pp. 51–64.

[7] Chao Shen et al. “Pattern-Growth Based Mining Mouse-Interaction Behavior for an Active User Authentication System”. In: *IEEE Transactions on Dependable and Secure Computing* 17.2 (2020), pp. 335–349.

[8] Diego Valsesia et al. “User Authentication via PRNU-Based Physical Unclonable Functions”. In: *IEEE Transactions on Information Forensics and Security* 12.8 (2017), pp. 1941–1956.

[9] Jerome H Saltzer. “Protection and the control of information sharing in Multics”. In: *Communications of the ACM* 17.7 (1974), pp. 388–402.

[10] Blase Ur et al. “Do users’ perceptions of password security match reality?” In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 2016, pp. 3748–3760.

[11] Zhigong Li, Weili Han, and Wenyuan Xu. “A Large-Scale Empirical Analysis of Chinese Web Passwords”. In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. SEC’14. San Diego, CA: USENIX Association, 2014, pp. 559–574. ISBN: 9781931971157.

[12] Robert Morris and Ken Thompson. “Password security: A case history”. In: *Communications of the ACM* 22.11 (1979), pp. 594–597.

[13] Philippe Oechslin. “Making a Faster Cryptanalytic Time-Memory Trade-Off”. In: *Advances in Cryptology - CRYPTO 2003*. Ed. by Dan Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 617–630. ISBN: 978-3-540-45146-4.

[14] Jerry Ma et al. “A Study of Probabilistic Password Models”. In: *2014 IEEE Symposium on Security and Privacy*. 2014, pp. 689–704.

[15] Ding Wang et al. “Targeted online password guessing: An underestimated threat”. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 1242–1254.

[16] Markus Dürmuth and Thorsten Kranz. “On password guessing with GPUs and FPGAs”. In: *International Conference on Passwords*. Springer. 2014, pp. 19–38.

[17] Dan Goodin. *Anatomy of a hack: How crackers ransack passwords like “qeadzcwrsfxv1331”*. https://arstechnica.com/information-technology/2013/05/how-crackers-make-minced-meat-out-of-your-passwords/. Accessed: 2022-06-29.

[18] Jan Camenisch, Anja Lehmann, and Gregory Neven. “Optimal distributed password verification”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 182–194.

[19] Ari Juels and Ronald Rivest. “Honeywords: Making password-cracking detectable”. In: *Proceedings of the ACM Conference on Computer and Communications Security* (Nov. 2013), pp. 145–160.

[20] Hristo Bojinov et al. “Kamouflage: Loss-Resistant Password Management”. In: *Proceedings of the 15th European Conference on Research in Computer Security*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 286–302. ISBN: 3642154964.

[21] Fred Cohen. “The use of deception techniques: Honeypots and decoys”. In: *Handbook of Information Security* 3.1 (2006), pp. 646–655.

[22] Mohammed H Almeshekah, Eugene H Spafford, and Mikhail J Atallah. “Improving security using deception”. In: *Center for Education and Research Information Assurance and Security, Purdue University, Tech. Rep. CERIAS Tech Report* 13 (2013), p. 2013.

[23] Neal Krawetz. “Anti-honeypot technology”. In: *IEEE Security & Privacy* 2.1 (2004), pp. 76–79.

[24] Robert Hackett. *Yahoo Raises Breach Estimate to Full 3 Billion Accounts, By Far Biggest Known*. https://fortune.com/2017/10/03/yahoo-breach-mail/. Accessed: 2022-06-29.

[25] Patrick Heim. *Resetting passwords to keep your files safe*. https://blog.dropbox.com/topics/company/resetting-passwords-to-keep-your-files-safe. Accessed: 2022-06-29.

[26] Poul-Henning Kamp et al. "LinkedIn password leak: salt their hide." In: *ACM Queue* 10.6 (2012), p. 20.

[27] Thu T. Pham. *Four Years Later, Anthem Breached Again: Hackers Stole Credentials*. https://duo.com/blog/four-years-later-anthem-breached-again-hackers-stole-employee-credentials. Accessed: 2022-06-29.

[28] Mohammed Almeshekah et al. "ErsatzPasswords: Ending Password Cracking and Detecting Password Leakage". In: Dec. 2015, pp. 311–320.

[29] Ding Wang et al. "A Security Analysis of Honeywords". In: Oct. 2017.

[30] Chun Wang et al. "The next domino to fall: Empirical analysis of user passwords across online services". In: *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*. 2018, pp. 196–203.

[31] *Google Online Security Survey - Harris Poll*. https://services.google.com/fh/files/blogs/google_security_infographic.pdf. Accessed: 2022-03-29.

[32] Ding Wang et al. "Zipf's Law in Passwords". In: *IEEE Transactions on Information Forensics and Security* 12 (June 2017), pp. 2776–2791.

[33] Imran Erguler. "Achieving Flatness: Selecting the Honeywords from Existing User Passwords". In: *IEEE Transactions on Dependable and Secure Computing* 13 (Feb. 2015), pp. 1–1.

[34] Yimin Guo, Zhenfeng Zhang, and Yajun Guo. "Superword: A honeyword system for achieving higher security goals". In: *Computers & Security* 103 (2021), p. 101689.

[35] Donghoon Chang et al. "Generation of secure and reliable honeywords, preventing false detection". In: *IEEE Transactions on Dependable and Secure Computing* 16.5 (2018), pp. 757–769.

[36] Yanzhao Tian et al. "Achieving flatness: Graph labeling can generate graphical honeywords". In: *Computers & Security* 104 (2021), p. 102212. ISSN: 0167-4048.

[37] Burton H Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[38] Bin Fan et al. “Cuckoo filter: Practically better than bloom”. In: *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 2014, pp. 75–88.

[39] Christian Esteve Rothenberg et al. “The deletable Bloom filter: a new member of the Bloom family”. In: *IEEE Communications Letters* 14.6 (2010), pp. 557–559.

[40] Payment Card Industry. *Hardware Security Module (**HSM**) Security Requirements*. https://www.pcisecuritystandards.org/documents/PCI%20HSM%20Security%20Requirements%20v1.0%20final.pdf. Accessed: 2022-06-29.

[41] Jan Camenisch, Anna Lysyanskaya, and Gregory Neven. “Practical yet Universally Composable Two-Server Password-Authenticated Secret Sharing”. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS ’12. Raleigh, North Carolina, USA, 2012, pp. 525–536. ISBN: 9781450316514.

[42] Xun Yi et al. “ID2S Password-Authenticated Key Exchange Protocols”. In: *IEEE Transactions on Computers* 65 (Dec. 2016), pp. 1–1.

[43] Canyang Shi and Huiping Sun. “HoneyHash: Honeyword Generation Based on Transformed Hashes”. In: *Secure IT Systems*. 2021.

[44] Nilesh Chakraborty and Samrat Mondal. “On designing a modified-UI based honeyword generation approach for overcoming the existing limitations”. In: *Computers & Security* 66 (2017), pp. 155–168.

[45] Antreas Dionysiou, Vassilis Vassiliades, and Elias Athanasopoulos. “HoneyGen: Generating Honeywords Using Representation Learning”. In: *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 2021, pp. 265–279.

[46] Matt Weir et al. “Password Cracking Using Probabilistic Context-Free Grammars”. In: May 2009, pp. 391–405.

[47] Eugene H Spafford. “Observing reusable password choices”. In: (1992).

[48] Arvind Narayanan and Vitaly Shmatikov. “De-anonymizing Social Networks”. In: *2009 30th IEEE Symposium on Security and Privacy*. 2009, pp. 173–187.

[49] Lee Mathews. *Equifax Data Breach Impacts 143 Million Americans*. https://www.forbes.com/sites/leemathews/2017/09/07/equifax-data-breach-impacts-143-million-americans/?sh=297a78a5356f. Accessed: 2022-03-29.

[50] Anupam Das et al. “The tangled web of password reuse.” In: *NDSS*. Vol. 14. 2014. 2014, pp. 23–26.

[51] Sarah Pearman et al. “Let’s go in for a closer look: Observing passwords in their natural habitat”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 295–310.

[52] Nilesh Chakraborty and Samrat Mondal. “Few notes towards making honeyword system more secure and usable”. In: *Proceedings of the 8th International Conference on Security of Information and Networks*. 2015, pp. 237–245.

[53] Nilesh Chakraborty et al. “Cryptanalysis of a Honeyword System in the IoT Platform”. In: *IEEE Internet of Things Journal* (May 2021), pp. 1–12.

[54] Muhammad Ali Fauzi, Bian Yang, and Edlira Martiri. “PassGAN-Based Honeywords System”. In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2019, pp. 179–184.

[55] Omar Z Akif et al. “Achieving Flatness: Honeywords Generation Method for Passwords based on user behaviours”. In: *International Journal of Advanced Computer Science and Applications* 10.3 (2019).

[56] Ding Wang and Ping Wang. “Two Birds with One Stone: Two-Factor Authentication with Security Beyond Conventional Bound”. In: *IEEE Transactions on Dependable and Secure Computing* 15.4 (2018), pp. 708–722.

[57] William J. Burns. *RockYou Password Leak*. `https://www.kaggle.com/datasets/wjburns/common-password-list-rockyoutxt`. Accessed: 2022-06-29.