DOI: 10.53608/estudambilisim.1318760

(Geliş Tarihi / Received Date: 22.06.2023, Kabul Tarihi/ Accepted Date: 13.10.2023)

(**Derleme Makalesi**)

Parola Saklama Tekniklerinin Evrimi ve Güncel En İyi Uygulamaları

Tuğberk KOCATEKİN[*1]

[1]İstanbul Arel Üniversitesi, Mühendislik-Mimarlık Fakültesi, Yazılım Mühendisliği Bölümü, Büyükçekmece, İstanbul, ORCID No : http://orcid.org/0000-0001-6171-0135

Anahtar Kelimeler:
Özetleme,
Tuzlama,
Anahtar Türetme Fonksiyonları,
Şifreler

Özet: Parolalar tarihsel olarak erişim kontrolü ve kimlik doğrulama için kilit bir öneme sahip olmuşlarsa da, güvenlikleri bugünün dijital dünyasında tekrar eden bir endişe olarak kalmaktadır. Yüksek profilli veri ihlalleri ve güvenlik açıklarıyla kanıtlandığı gibi, güvenli parola saklama her zaman en üst düzeyde önemli olmasına rağmen, genellikle başarılı olunamamıştır. Kullanıcılar güçlü ve akılda kalıcı parolalar oluşturma konusunda uğraşırken, parolaların güvenli bir şekilde saklama sorumluluğu da hizmet sağlayıcılara düşmektedir. Alternatif kimlik doğrulama mekanizmaları ortaya çıkmış olmasına rağmen, parola tabanlı kimlik doğrulama yaygın olarak kullanılmaya devam etmektedir. Araştırmalar, yazılım geliştiricilerin parola saklama güvenliği konusunda yanılgılara ya da ihmalkarlığa düştüğünü göstermektedir. Bu makale, Crypt ile başlayıp Parola Özetleme Yarışması'nın kazananı Argon2d'de son bulan parola saklama yöntemlerinin ilerleyişini izlemektedir. Dört adet modern parola saklama sistemleri hakkında bilgi vererek bu bilgi boşluğunu kapatmayı, daha iyi uygulamalar için savunma yapmayı ve güvenliği işlevsellikle birlikte önceliklendirme önemini aydınlatmaya çalışmaktadır.

(**Review Article**)

Evolution and State of the Art in Password Storage

Keywords:
Hashing,
Salting,
Key Derivation Functions,
Passwords

Abstract: Passwords have historically been pivotal for access control and authentication, yet their security remains a recurring concern in today's digital world. As evidenced by high-profile data breaches, secure password storage has always been paramount, but often not achieved. While users grapple with the creation of strong, memorable passwords, the burden also falls on service providers to store these passwords securely. Even though alternative authentication mechanisms have emerged, password-based authentication remains pervasive. Surprisingly, studies highlight that developers frequently exhibit misconceptions or negligence towards password storage security. This paper traces the progression of password storage methods by explaining four password hashing methods. By informing of four modern password storage systems, this work seeks to bridge the knowledge gap, advocating for better practices and illuminating the significance of prioritizing security alongside functionality.

1. INTRODUCTION

Passwords are widely used for authenticating ourselves to reach sensitive information. It is known that poor password practices end up with being exploited and expose private information about users. Although password security is just a component of a system's security, it is essential [1]. In the recent years, both Sony and LinkedIn were hacked and user information was published [2], [3]. In addition, there are local breaches that are known such as Yemeksepeti [4]. It should be noted that it is also possible there are some attacks which are not even known to public. These data breaches have both monetary and non-monetary effects on companies. For example, it was calculated that data breach on Sony had a direct cost exceeding 150M US dollars and combined with the brand damage, it exceeded 1B US dollars [5], [6]. Choosing a right password is very important for users. If the user lacks a strong password, attackers can easily guess the user password regardless of the security of services. They may have to use and memorize multiple

*Sorumlu Yazar/Corresponding Author: tugberk@gmail.com passwords. Password managers are such software which aims to solve this problem. However, it doesn't end there. Service providers and applications also needs to keep their security in check. It is also known that several vulnerabilities are found in the service providers [7].

There are difference schemes to use for authentication other than using passwords, but it is still one of the most common ways for authentication. There are several studies showing that developers are still asking a lot of questions about passwords and password storage online and password storage is one of the most popular areas in security [8]. Apart from these, there are a lot of studies showing that developers are having a hard time to store passwords in a secure way [9]. Although many developers are working in a group setting and it would be expected that it would be more secure, number of developers working alone cannot be dismissed [10]. It is also safe to say that developers are not prioritize security [10] and put functionality above security [11].

Hallett et al. [9] conducted an experiment with 138 developers and asked them to write code to store password in whatever language they want. Half of them were asked to write a specification before writing the code and the other half were asked to write code immediately. It turned out that although they were confident in themselves, they failed to store passwords in a secure manner. Only 38% used hashing algorithms and only 14% used salting while storing passwords. Naiakshina et al. have several studies on how developers approach security [11], [12]. First, he did a study on 20 computer science students and gave the task of password storage. The outcome of the study was that students consider functionality before security. Unless the students are primed to consider security, they didn't. Even those who were primed, they could not meet the standards of the time [11]. Examples above show that there is a lack of knowledge on password storage. This study aims to inform users by giving knowledge about modern password storage systems and their comparisons.

2. MATERIALS AND METHODS

2.1. Types of attacks

Password guessing attacks can be classified into three categories: brute-force attacks, dictionary attacks and rainbow tables [13]. Although all of them are explained, throughout the paper; password cracking refers specifically to brute force and dictionary attacks.

2.1.1. Brute-force attacks

Brute-force attacks are generally applied to the hashed values of the passwords to get the original plaintext. These attacks are CPU-intensive and therefore time-consuming [14]. In brute-force attacks, the attacker is trying every possible combination of the available characters. For example, if a password of eight digits is being attacked, the attack will start by typing "aaaaaaaa", "aaaaaaab" and increment to "zzzzzzzz" [15]. There are also several configurations to adjust the character set based on the data set. Attacker can choose the character set as numeric-only, alphanumeric and alike.

2.1.2. Dictionary attacks

Studies show that users tend to select easy to remember and predictable passwords [16]. Password leaks show us that they use common names and surnames, pet names, band names, sports team names, etc. [17]. These phrases are combined together alongside with the password obtained from recent-leaks to create a list of passwords, also called a dictionary. Instead of trying all possible combinations of characters, words from these dictionaries are used to crack passwords. There are four types of dictionary attacks [13]: i) pure, ii) Probabilistic Context Free Grammar (PCFG) based [18], iii) Markov model based [19] and iv) mangling rules [20]. In pure type, attacker is just using a simple dictionary. In second, dictionary is constructed by using PCFG theories and it contains modified passwords with assigned probabilities. In the third type, Markov based models are applied based on the probability distribution over sequences of characters to create new passwords [19]. Finally in the last type, rules are applied to words in the dictionary and generate new highly likely passwords such as combination of multiple words, mixed letter cases and leet speak [21].

2.1.3. Rainbow tables

Cryptographic hash functions are not reversable. Therefore, attackers cannot reverse the hash to get the original password. However, attackers may use precalculated tables for hash lookup. These tables are called rainbow tables. When the attacker has access to the database where the passwords are hashed instead of plaintext, attacker can use rainbow tables to search for hash values to find the original password. However, it should be noted that rainbow tables are not usable for salted passwords, multiple hashes, or combination of several hash functions [22].

2.2. Cryptographic hash functions

Generally, a hash function is used to compress (index) arbitrary-length strings into shorter strings, in order to achieve O(1) insertion and lookup times for a set of elements. However, as the amount of data increases, possibility of having a collision also increases. That is why regular hash functions are not good candidates to use in cryptography.

A good hash function has to supply a unique output for every possible input to minimize the possibility of collision. Those kind of hash functions are called collision-resistant hash functions. Designing those are not as easy as creating a regular hash function, where the only purpose is to index files as a data structure. However, in order to use hash functions in cryptography, collision-resistance is one of the key goals and therefore they have a more advanced design [23]. It should be noted that although every cryptographic hash function is a hash function, not every hash function can be called as cryptographic hash function.

According to Katz and Lindell [23], there are three levels of security when considering a cryptographic hash function: collision resistance, second pre-image resistance and preimage resistance. • Collision resistance: It should be computationally infeasible to find a pair of different input values (m, m′) to have the same digest. • Second pre-image resistance: It should be computationally infeasible to find a message m′, to hash to the same output as message m. • Pre-image resistance: It should be computationally infeasible to find a message m′, which hashes to a specific output, y = H(m). Here, if a hash function is collision resistant, it is also preimage resistant; because if there is a second preimage, then that means there is a colliding pair. A pre-image resistant function is called a one-way function, since it is difficult to inverse it.

One-way function means that there is no inverse on that function, meaning that one cannot find message m by looking at the digest y, where H(m) = y. That is why cryptographic hash functions are used in several information security areas such as digital signatures, message authentication codes, fingerprinting, checksums, and many more [24], [25], [26].

2.2.1. Salting

It is possible that multiple users can have the same password. If this is the case, the hashed values are going to be the same. This can easily be a security problem. It is also possible that these passwords can be used in other services. In order to solve this problem, a random value (salt) is concatenated with the password in the registration process and the instead of just giving the password to the hash function, the system now gives salt+password to it [27]. Therefore, even if multiple users are using the same password, the hash value would be different because the password has a unique salt concatenated to it (see Figure 1). In addition to this, since one or more systems are going to be using different salts, salting makes it impossible for an attacker to find out whether a person is using the same password on two or more systems [1]. Salting also provides security against the usage of rainbow tables [28] and dictionary attacks [29], [30].

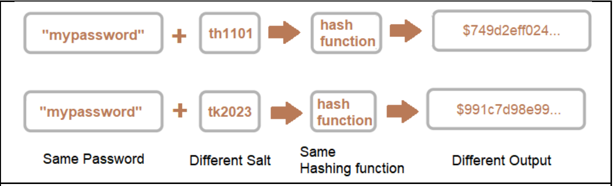

Figure 1. How salt effects the resulting hash

2.2.2. Key derivation functions

Key derivation functions (KDF) aim to produce one or more secret keys from a secret value such as passwords. They derive cryptographic keys of desired length from passwords. They are usually implemented by secure cryptographic hash functions or HMACs [27]. PBKDF2 [31], scrypt [32] and argon2 [33] are examples to KDFs.

Password-based key derivation functions are mainly used for these two purposes: hashing passwords and creating cryptographic keys [34]. Key derivation functions use secure hashes to provide password hashing and therefore is strong against key derivation attacks. However, it is possible for attackers to apply a brute-force attack to get the output.

2.2.3. Key stretching

Ordinarily, passwords are chosen to be 8 characters (8 bytes). Since it is low-entropy, it is susceptible to exhaustive search attacks where the attacker is trying every possible combination until they find the password [27]. Key stretching is the typical method for protection of such attacks. The term *key stretching* was first used by Kelsey et al. in 1997 [29]. The aim is to increase the entropy of a low-entropy key by adding bits to it and thus increase the time required for a brute-force attack. With key stretching, low-entropy s-bit keys are converted into a longer s + t bit keys and the difficulty of a brute-force attack is increased to 2 s+t operations instead of 2 s . It also makes exhaustive searching more expensive for the attacker [28]. However, this process also slows down the user login process, therefore key stretching should be limited to the user's tolerance [27].

2.3. Password hashing schemes

The easiest way to store passwords in a database is by storing them as plaintext. Wilkes [35] observed that storing passwords as plaintext is insecure. If the database is compromised, the attacker can see the passwords easily. In addition, even there is no attacker, people who has access to database can also see passwords. However, one was able to find systems storing passwords as plaintext until the recent years.

2.3.1. Password hashing

The passwords were stored as plaintext in the filesystem in the early versions of Unix operating systems. This made it very hard for system administrators to adjust read and write permissions in order to provide security for password files. After an incident where the password file became visible to everyone due to a software design error, Morris and Thompson decided to solve the password storage problem. They are credited as the first ones with the idea of storing passwords by using a one-way function and storing the output instead of the original password [1]. This algorithm was called Crypt and it was based on Data Encryption Standard (DES) algorithm.

Crypt was a very important breakthrough for password storage. Before that, the standard was storing the passwords as plaintext and it was known to be insecure [35]. With Crypt, a minimum-security standard for password storage emerged. After that, storing hash values instead of plaintext became a standard [28]. Other important contributions of Crypt are adopting key stretching and usage of salts. It is based on DES and works on encrypting the password instead of hashing it. However, it is used as a hash function [28].

Password hashing is applying a moderately-hard function to a password or a password concatenated with salt. Resulting output is the password hash and is stored in the database instead of storing the password as plaintext. This way, even if the database or file containing the password is compromised, a brute-force attack will be costly for the attacker [36]. Additional approach can be applying the hash function multiple times. This does not present any problem for the user, but it creates a time-consuming task for the attacker. However, evolution of Moore's law led to production of faster gates and units. Nowadays, GPUs can compute billions of instructions in a second, it has become easier for attackers to run cryptographic hash functions such as SHA1, MD5, etc. Mishra and Janarthanan [37] showed that GPUs accelerate the cracking process compared to CPU implementation by launching comprehensive search attacks on password hashing schemes. Therefore, using cryptographic hash functions by themselves are not considered as secure anymore [11].

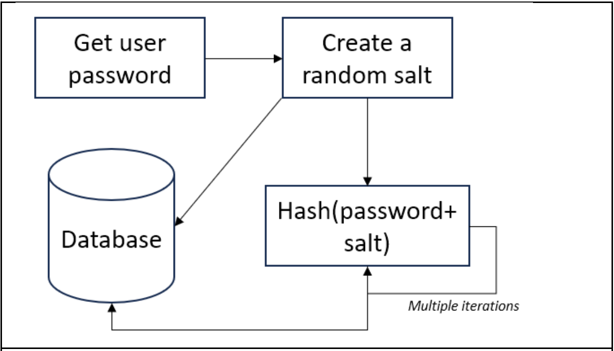

Figure 2. A very basic example of password storage

2.4. Standards

Since GPU's are improved a lot and can run billions of instructions per second [38], one-way hash functions can be calculated very quickly. For these reasons, new and more computationally intensive password hashing schemes are proposed such as PBKDF2 [39], bcrypt [40] and scrypt [32].

In 2013, because of limited set of solutions, cryptographic community announced Password Hashing Competition (PHC) with the purpose of providing a better solution to password hashing problem [27]. Out of the initial 22 candidates, Argon2 ended up being the winner. NIST (National Institute of Standards and Technology) published Digital Identity Guidelines in 2017, with a recent update in 2020 [41]. In addition to many recommendations about password creation and authentication, it also provides guidelines on secure password hashing. They recommended storing passwords in such form that they are resilient to offline attacks. For this reason, they stated that passwords must be salted and hashed by using a suitable one-way key derivation function such as PBKDF2 and Balloon [42] using an approved one-way function such as HMAC [25], and approved hash function such as SHA3 [43].

2.4.1 bcrypt

Bcrypt was designed by Provos and Mazieres in 1999 as an improvement to the present password schemes [40]. In their study, they presented two algorithms, eksblowfish and bcrypt. Eksblowfish is a block cipher with a purposefully expensive key schedule, and bcrypt is the password hash function related to it.

Bcrypt algorithm gets three parameters as input: cost, salt and key. Cost parameter is used to change the cost of computation. Salt is a random 128-bit value. It is chosen randomly in order to create a different output even if the key is the same. Key is the user-chosen password which is used to encrypt a specific 192-bit plaintext (OrpheanBeholderScryDoubt) and it can be up to 72 bytes. It produces a 192-bit hash as output.

Eksblowfish is a variable cost and salted block cipher based on Blowfish algorithm [40]. Blowfish is a 64-bit block cipher [44]. It is free and resides in public domain, where everyone can use freely. Authors replaced the key setup to be able to control the speed of the function. Therefore, by adjusting the cost, it can render one of the most common offline attacks (dictionary attack) unfeasible. The user can increase the cost as much as they want as long as it is tolerable by the users.

In short, *Bcrypt* is a password-hashing function based on Blowfish cipher. It is not a key derivation function because the output is fixed. It is the output of encrypting OrpheanBeholderScryDoubt 64 times using Blowfish cipher and presents the end result as the hash. It is also used as the default password hashing scheme for the BSD operating system.

2.4.2. PBKDF2: Password based key derivation function 2

First and foremost, PBKDF2 is a key derivation function. The aim is to provide cryptographic keys for encryption algorithms by using the user-chosen passwords. PBKDF2 was chosen as a good key derivation function and was mentioned in NIST documents [45]. It was also used in Blackberry and IOS systems [46]. It is also used in TrueCrypt, WPA2, WinRAR [47] and many more.

PBKDF2 introduces CPU-intensive operations which are intentionally designed to take more time to compute by applying key stretching [31]. This helps by providing better resistance for brute-force attacks by increasing the computation time by iterating the hash of the salted passwords multiple times [48], [29]. Random salts are used to provide different keys from the same password and the iteration count is used to call the pseudorandom function. PBKDF2 requires four inputs: a user password *pwd*, a salt *S*, desired output length k_{len} and an iteration counter *c*.

NIST recommends the iteration count should be as high as possible unless there are performance problems, and stated that 10.000 iterations are the minimum number of iterations [41]. However, Blocki et al. [49] stated that PBKDF2-SHA256 with 100.000 iterations is not enough to provide secure user password storage. They also stated that memory hard functions like scrypt [32] and argon2 [33] would provide better protection compared to PBKDF2. In addition, since PBKDF2 does not take memory into account, it is vulnerable to parallel attacks [27]. This vulnerability is important to note as processors are getting faster and cheaper, the number of parallel attacks is expected to rise.

2.5. Memory hard functions

With constant improvement in technology, we expect that processors are going to be faster and smaller. Although using iteration count in key derivation functions helps us to cope with the increasing speed of processors, attackers would be able to store more processing power in a given space. This would allow attackers to have more parallelism with the same cost. It means that in time, the iteration count must be increased to keep the information secure from the attackers because their computation power will increase. However, not only that, attackers are going to be able to store more processors in a given space with the same cost and will have highly parallel circuits. That means that in time, CPU intensive security will be disadvantageous to brute-force attacks [34].

Although Kelsey et al. [29] stated that using moderately large amounts of RAM would make hardware attacks more expensive, before Percival's work [34] introducing a sequential memory-hard algorithm **scrypt**, such functions used only constant memory.

2.5.1. scrypt

Scrypt is a password-based key derivation function created by Colin Percival in 2009 for the Tarsnap online backup service [32]. He introduced the concept of memory-hard algorithm and a sequential memory-hard function.

It was specifically designed to require large amounts of memory so that it would be costly to perform custom hardware attacks to it. It is widely used in proof-of-work schemes for well-known cryptocurrencies such as Litecoin and Dogecoin [36]. Since it is memory-intensive, it reduces the risk of brute-force attacks by making them more expensive computationally. Therefore, it is a good option for password hashing and therefore used in password storage.

As input, scrypt takes a passphrase *P*, a salt *S*, and parameters *N, p*, and *dkLen*. These are CPU/Memory cost parameters, parallelization parameter and the output length in octet respectively. It then generates a derived key *DK* with output of length *dkLen* octets to be used as a cryptographic key.

First, it uses PBKDF2 with SHA256 and generates p blocks of octets from the provided password and salt. Generated blocks are mixed by a mixing function in order to make the computation expensive. Lastly, the result is used as salt for another PBKDF2 computation to get the final result. However, it is vulnerable to attacks such as cache-timing and garbage-collector attacks [50], [51].

2.5.2. Argon2

Argon2 is the next generation of memory hard hash function Argon [33]. It became the winner of the Password Hashing Competition (PHC) out of 24 submitted projects and became the standard for password storage. It is the state of the art in memory-hard functions for password storage. It is optimized for x86 architecture and exploits the cache and memory organization of the recent Intel and AMD processors [52].

It has two flavors: Argon2d and Argon2i. Argon2i is more suitable to use for password storage because it uses data-independent memory access and therefore preferred for password hashing and password-based key derivation [53]. Being slower compared to Argon2d helps provide better security from tradeoff attacks. It is slower because it uses the memory at a speed of two processor cycles per byte [54]. Argon2d on the other hand uses data-dependent memory access and is better suited for applications such as backend servers and cryptocurrencies, where side-channel attacks are not a threat [55].

Argon2 has two types of inputs: primary and secondary inputs. Primary inputs are the message *P* as password with length between 0 and $2^{32} - 1$ bytes; and a nonce *S* as a salt for password hashing with length between 8 and $2^{32}-1$ bytes. Secondary inputs are not mandatory, but they are as follows: a degree of parallelism *p*, tag length ρ, memory size *m*, number of iterations *t*, version number *v*, a key *K*, associated data *X* and type(mode) as *y* [52].

3. DISCUSSION AND CONCLUSION

Although there are a lot of downsides to it, passwords are still the primary way to authenticate users with various systems. They are usually stored in the database in hashed form. When attackers compromise the database, they apply dictionary and brute-force attacks to crack these hashes. They are often successful because passwords have low-entropy. With the emergence of Crypt, developers started to add salt values and use one-way hash functions to passwords to make them secure against these types of attacks.

Computers are rapidly advancing, leading to quicker computation of hash algorithms. To combat this, hash functions are iterated numerous times. But this isn't without its issues. As GPU technology advances, CPU-intensive calculations aren't as secure. Attackers migrated to new architectures such as FPGAs, GPUs, dedicated

ASIC modules [33]. A solution to this problem was memory-hard functions [34], like scrypt which require a large amount of memory and therefore making it costlier for attackers [49].

Table 1. Comparison of password storage algorithms

Algorithm	Iterations	Key Length	Speed	Security	Highlights
Bcrypt	Configurable (>= 4,000)	192	Moderate	Strong	Resistant to rainbow tables (salt)
Scrypt	Configurable	Varied Length	Slow	Very strong	Memory hard
Argon2	Configurable	Varied Length	Slow	Very strong	Memory hard
PBKDF2	Configurable (> 10,000)	Varied Length	Moderate	Strong	Very popular and widely used

This research delves into the criticality of safe password storage and various methods, ranging from less secure to highly secure techniques. Given NIST's recommendation of key derivation functions, we shed light on their purpose and application in password storage.

Recent studies advocate for the use of memory-hard functions [11]. However, it's essential to note that not all systems are updated regularly. This paper serves as a guide for those still using older security measures to protect user credentials. It's very important for system admins to regularly upgrade their security, considering the evolving threats.

To conclude, considering past studies that indicate developers' limited understanding of secure password storage [9, 10, 11], our research aims to educate readers on the current best practices for password storage.

Ethical Considerations

Compliance with ethical guidelines

Funding

No funding.

Conflict of interest

There is no conflict of interest.

REFERENCES

[1] Morris, R., Thompson, K. 1979. Password Security: A Case History, Communications of the ACM, 22(11), 594-597.

[2] Goode, S., Hoehle, H., Venkatesh, V., Brown, SA. 2017. User Compensation as a Data Breach Recovery Action, MIS Quarterly, 41, 703–A16.

[3] Gibson, B., Townes, S., Lewis, D., Bhunia, S. 2021. Vulnerability in Massive Api Scraping: 2021 linkedin data breach, 2021 International Conference on Computational Science and Computational Intelligence (CSCI).

[4] webteknohaber. 2021. Yemeksepeti Hacklendi: Kullanıcıların Hesap Bilgileri Ele Geçirildi, 27 Mart 2021. Available: https://www.webtekno.com/yemeksepeti-kullanici-veri-tabani-siber-saldiri-h108027.html.

[5] Hachman, M. 2011. PlayStation Hack to Cost Sony $171M; Quake Costs Far Higher, 23 May 2011. [Çevrimiçi]. Available: https://news.yahoo.com/playstation-hack-cost-sony-171m-quake-costs-far-163824525.html?guccounter=1.

[6] Sherr, I., Wingfield, N. 2011. Play by Play: Sony's Struggles on Breach, 7 May 2011. [Çevrimiçi]. Available: https://www.wsj.com/articles/SB10001424052748704810504576307322759299038.

[7] Hatzivasilis, G. 2020. Password Management: How Secure Is Your Login Process?, International Workshop on Model-Driven Simulation and Training Environments for Cybersecurity.

[8] Yang, X.-L., Lo, D., Xia, X., Wan, Z.-Y., Sun, J.-L. 2016. What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts, Journal of Computer Science and Technology, 31, 910–924.

[9] Hallett, J. , Patnaik, N., Shreeve, B., Rashid, A. 2021. "Do this! Do that!, And Nothing Will Happen" Do Specifications Lead to Securely Stored Passwords?, 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE).

[10] Van Der Linden, D., Anthonysamy, P., Nuseibeh, B., Tun, T. T., Petre, M., Levine, M., Towse, J., Rashid, A. 2020. Schrödinger's Security: Opening the Box on App Developers' Security Rationale, Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, 2020.

[11] Naiakshina, A., Danilova, A., Tiefenau, C. , Herzog, M., Dechand, M., Smith, M. 2017. Why do Developers Get Password Storage Wrong? A Qualitative Usability Study, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017.

[12] Naiakshina, A., Danilova, A., Gerlitz, E., Von Zezschwitz, E., Smith, M. 2019. If You Want, I Can Store The Encrypted Password a Password-Storage Field Study with Freelance Developers, Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, 2019.

[13] Christoforos Ntantogian, S. M. C. X. 2019. Evaluation of Password Hashing Schemes in Open Source Web Platforms, Computers & Security, 206-24.

[14] Raza, M., Iqbal, M., Sharif, M., Haider, W. 2012. A survey Of Password Attacks and Comparative Analysis on Methods for Secure Authentication, World Applied Sciences Journal ,19(4), 439-444.

[15] Kyaw, A. K., Sioquim, F., Joseph, J. 2015. Dictionary attack on Wordpress: Security and Forensic Analysis, 2015 Second International Conference on Information Security and Cyber Forensics (InfoSec), Cape Town, 2015.

[16] Bošnjak, L., Sreš, J., Brumen, B. 2018. Brute-force and Dictionary Attack On Hashed Real-World Passwords, 2018 41st İnternational Convention on Information and Communication Technology, Electronics And Microelectronics (Mipro), Opatija, 2018.

[17] Zviran, M., Haga, W. J. 1999. Password Security: an Empirical Study, Journal of Management Information Systems, 15(4), 161-185.

[18] Matt Weir, S. A. B. d. M. B. G. 2009. Password Cracking Using Probabilistic Context-Free Grammars, 30th IEEE Symposium on Security and Privacy, 2009.

[19] Arvin Narayanan, V. S. 2005. Fast Dictionary Attacks on Passwords Using TimeSpace Tradeoff, Proceedings of the 12th ACM Conference on Computer and Communications Security, Virginia, 2005.

[20] Marechal, S. 2012. Automatic Mangling Rules Generation, December 2012. [Çevrimiçi]. Available: https://www.openwall.com/presentations/Passwords12-Mangling-Rules-Generation/Passwords12-Mangling-Rules-Generation.pdf. [Erişildi: 25 09 2023].

[21] Briland, H., Paolo, G., Giuseppe, A., Fernando, P.-C.2017. PassGAN: A Deep Learning Approach for Password Guessing, *CoRR*, 2017.

[22] Josef Horálek, F. H. O. H. L. P. V. S. 2017. Analysis of the Use of Rainbow Tables to Break Hash, Journal of Intelligent & Fuzzy Systems, 1523-1534.

[23] Katz, J., Lindell, Y. 2020. Introduction to Modern Cryptography, CRC press, 2020.

[24] Merkle, R. C. 1987. A Digital Signature Based on a Conventional Encryption Function, Conference on the theory and application of cryptographic techniques.

[25] Krawczyk, H., Bellare, M., Canetti, R. 1997. HMAC: Keyed-hashing for Message Authentication.

[26] Oostveen, J., Kalker, T., Haitsma, J. 2002. Feature Extraction and A Database Strategy for Video Fingerprinting, Recent Advances in Visual Information Systems: 5th International Conference, VISUAL 2002 Hsin Chu, Taiwan, March 11–13, 2002 Proceedings 5, 2002.

[27] Hatzivasilis, G., Papaefstathiou, I., Manifavas, C. 2015. Password Hashing Competition-Survey and Benchmark, Cryptology ePrint Archive.

[28] Forler, C., Lucks, S., Wenzel, J. 2013. Catena: A Memory-Consuming Password-Scrambling Framework, Cryptology ePrint Archive.

[29] Kelsey, J., Schneier, B., Hall, C., Wagner, D. 1997. Secure Applications of Low-Entropy Keys, International Workshop on Information Security.

[30] Abadi, M., Lomas, T. M., Needham, R. 1997. Strengthening passwords, Digital Equipment Corporation Systems Research Center [SRC].

[31] Ertaul, L., Kaur, M., Gudise, V. A. K. R. 2016. Implementation and Performance Analysis of Pbkdf2, Bcrypt, Scrypt Algorithms, Proceedings of the international conference on wireless networks (ICWN).

[32] Percival, C., Josefsson, S. 2016. The Scrypt Password-Based Key Derivation Function.

[33] Biryukov, A., Dinu, D., Khovratovich, D. 2015. Argon 2 : The Memory-Hard Function For Password Hashing and Other Applications.

[34] Percival, C. 2009. Stronger Key Derivation Via Sequential Memory-Hard Functions, BSDCan.

[35] Wilkes, M.V. 1968. Time-Sharing Computer Systems. MacDonald Computer Monographs, American Elsevier Publishing Company.

[36] Alwen, J., Chen, B., Pietrzak, K., Reyzin, L., Tessaro, S. 2017. Scrypt is Maximally Memory-Hard, Annual International Conference on the Theory and Applications of Cryptographic Techniques.

[37] Mishra, J. K., Janarthanan, M. 2022. GPU-based Security Of Password Hashing İn Cloud Computing, Materials Today: Proceedings, 60, 939–944.

[38] Orman, H. 2013. Twelve Random Characters: Passwords İn The Era Of Massive Parallelism, IEEE Internet Computing, 17, 91–94.

[39] Kaliski, B. 2000. PKCS# 5: Password-based cryptography Specification version 2.0.

[40] Provos, N., Mazieres, D. 1999. A future-Adaptable Password Scheme., USENIX Annual Technical Conference, FREENIX Track.

[41] Grassi, P., Garcia, M., Fenton, J. 2020. Digital İdentity Guidelines.

[42] Boneh, D., Corrigan-Gibbs, H., Schechter, S. 2016. Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks, Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22.

[43] Kelsey, J., Chang, S.-j., Perlner, R. 2016. SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash, NIST special publication, 800, 185.

[44] Schneier, B. 1993. Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish), International Workshop on Fast Software Encryption.

[45] Turan, M. S., Barker, E., Burr, W., Chen, L. 2010. Recommendation for Password-Based Key Derivation, NIST special publication, 800, 132.

[46] Smartphone forensics: cracking blackberry.

[47] Chen, J., Zhou, J., Pan, K., Lin, S., Zhao, C., Li, X. 2013. The Security of Key Derivation Functions in WINRAR., J. Comput., 8, 2262–2268.

[48] Visconti, A., Mosnáček, O., Brož, M., Matyáš, V. 2019. Examining PBKDF2 security margin—Case study of

LUKS, Journal of Information Security and Applications, 46, 296–306.

[49] Blocki, J., Harsha, B., Zhou, S. 2018. On the economics Of Offline Password Cracking, 2018 IEEE Symposium on Security and Privacy (SP).

[50] Forler, C., Lucks, S., Wenzel, J. 2014. Memory-Demanding Password Scrambling, Advances in Cryptology – ASIACRYPT 2014, Berlin.

[51] Forler, C., List, E., Lucks, S., Wenzel, J. 2015. Overview of the Candidates for the Password Hashing Competition, Technology and Practice of Passwords, Cham.

[52] Polasek, V. 2019. Argon2 Security Margin for Disk Encryption Passwords.

[53] Biryukov, A., Dinu, D., Khovratovich, D. 2016. Argon2: New Generation Of Memory-Hard Functions For Password Hashing and Other Applications, 2016 IEEE European Symposium on Security and Privacy (EuroS&P).

[54] Duka, M. 2020. Elliptic-curve Cryptography (ECC) And Argon2 Algorıthm in Php Using Openssl and Sodium Libraries, Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska, 10, 91-94.

[55] Wetzels, J. 2016. Open Sesame: The Password Hashing Competition And Argon2.