Enhancing the resistance of password hashing using binary randomization through logical gates

Muhamad Zaki Anbari, Bambang Sugiantoro
Master of Informatics Department, Faculty of Science and Technology, Sunan Kalijaga Islamic State University, Yogyakarta, Indonesia

Article Info

Article history:

Received Feb 20, 2024
Revised Jun 14, 2024
Accepted Jun 17, 2024

Keywords:

Binary
Hash
Information security
Logical gates
Salt

ABSTRACT

Digitalization in various sectors makes information security issues very crucial. Information security follows the authentication, authorization, and accounting (AAA) principle, where one of the most important parts is authentication. The most widely used authentication method is username-password. The best method to secure a user-pass is to convert the plaintext using a hash so that the converted plaintext cannot be recovered. However, with higher technology, hackers can crack the ciphertext using brute force. This research proposes a username-password scrambling algorithm before it is fed into the hash function to improve resilience from attacks. This algorithm is named logical gates (LG). It works by converting the user pass into binary form, adding salt, and scrambling it with certain logical gates before inserting it into the hash function. Testing is divided into two: time of execution and attack resistance. Time of execution results show that LG takes 0.0443432033 s, while without LG takes 0.01403197646 s. The resistance of attack results show that the plaintext of the hash amplified by LG cannot be cracked at all and increases the attack time by 321.3% at prefix and 161.3% at postfix, while without LG, the plain text can be found for a certain duration of time.

Corresponding Author:

Muhamad Zaki Anbari
Master of Informatics Department, Faculty of Science and Technology, Sunan Kalijaga Islamic State University
Caturtunggal, Depok, Sleman, Yogyakarta, 55281, Indonesia
Email: mzakianbari@gmail.com

1. INTRODUCTION

Today most people have utilized technology to support their various activities such as communication, business, study, entertainment, public services, and so on. Each application of the technology is closely related to data or information so that information security becomes very important for everyone. The rapid digitization on the one hand does facilitate various activities but also creates risks such as data leakage. Confidentiality, integrity, and availability are three parameters that must be referenced in building information security [1]. Confidentiality means ensuring that the use and storage of data can only be done by those who have the right. Integrity means ensuring that the data is intact and not deformed or contaminated by malware or viruses. Availability means ensuring that users who have data access rights can access the data when they need it [2].

Every application must have an authentication system. Authentication is very important because it serves to ensure that the data or information in the application can only be accessed by those entitled to access it [3]. The most commonly used authentication is a combination of username and password, so the issue of securing password data is very important [4]. Some methods of securing passwords are by encrypting passwords, entering passwords into a hash function before storing them in the database [5], and creating strong passwords [6]. Strong password criteria include a minimum length of 6 characters, not using the same password for all accounts, using a combination of letters, numbers, symbols, and avoiding dictionary words as much as possible [7].

The method of securing passwords by storing hashes is the most secure method. Hackers will have difficulty finding the plaintext password due to the one-way nature of the hash function [8]. The use of hashes to secure data or passwords can be improved by adding salt [9]. Salt is random text that is combined with the plaintext password. Salt in addition to increasing the number of password characters will also make it more difficult to read and guess, making it more immune to various attacks such as dictionary attack, brute force attack, or rainbow table attack [10], [11]. One of the modules used to generate salt is Secrets. The Secrets module exists in the python programming language and is used to generate cryptographically strong random strings. Secrets has incorporated various parameters in cryptography theory to be able to generate random strings that are more immune to cryptographic attacks [12]. Apart from using salt, hash security is also affected by the combination scheme between password and salt. Research conducted by [13] clearly shows the effect of salt placement when used as prefix, postfix, and alternating. This means that the security of the hash function can be improved by increasing the complexity of the password before it is entered into the hash function.

Another method introduced to improve hash security is the SXR algorithm. SXR stands for split, XOR, and replace. This algorithm consists of 4 steps: the password is hashed, then the ratio and number of iterations of the secret key are calculated. Next, the hashed password is split into 2 based on the result of the ratio calculation. Finally, the obtained data is combined and stored in the database [14]. Another method introduced is by swapping the elements in the array proposed by Karrar *et al.* [15]. This algorithm works by randomizing the password and salt elements, then separating and hashing the salt when stored in the database, making it difficult for hackers to guess the stored password. Research conducted by De Guzman *et al.* [16] tried to strengthen the hash function by adding a hill chipper. Hill chipper is used to increase complexity when generating salt. This research focuses on overcoming hash-collision that may occur in large databases. The method used is that when the system detects hash-collision, the algorithm will take the first and last characters of the username, then input them into the matrix encryption key (MEK) which then generates a salt. The salt is combined with the original password to produce a new hash value so that hash-collision can be avoided.

Based on previous research, there are several methods to improve hash security. Starting from changing the hash function itself, adding salt, using complex passwords, or adding other algorithms before the hash function. In summary, this research aims to develop alternative algorithms in randomizing usernames, passwords, and salt before being entered into the hash function. The resulting algorithm is expected to have better complexity and robustness than other existing algorithms or methods. The proposed algorithm is named the logical gates (LG) algorithm. The way it works is that the username and password that have been combined with salt into binary form, then scramble them with a series of certain logic gates, the result is a binary circuit that is completely different from the initial binary circuit. Furthermore, the binary series is converted back to string form and entered into a hash function. In the final stage, testing is done using time of execution test and resistance of attack hash to compare the performance of LG algorithm with other algorithms.

2. PROPOSED ALGORITHM

At this stage the proposed algorithm is designed and developed. The proposed algorithm uses a combination of logic gates to scramble the username, password, and salt before being entered into the hash function. The flowchart of the proposed algorithm can be seen in Figure 1. Logic gates have many types with different functions. According to Garg and Kaur [17] there are 7 types of logic gates, namely AND, OR, NOT, NAND, NOR, XOR, and XNOR.

The LG algorithm uses some of these logic gates to randomize the binary username, password, and salt before entering the hash function. The first step is for the user to enter a username and password like most authentication methods. Second, the system will automatically generate a salt using the secrets module. The third salt and password will be combined using two different schemes in turn, namely the prefix and postfix schemes. The goal is to find out which scheme is better for strengthening the hash function.

The fourth step of the combined password and salt will be converted in binary form which only consists of the numbers 1 and 0. Then the binary will be entered sequentially through a series of certain logic gates. Determining the order of the logic gate circuit is very considering the function of the function of each logic gate. Errors in making sequences can cause functions between logic gates to be ineffective and eliminate each other. For example, when the NAND gate is followed by the NOT gate, it is the same as directly using the AND gate. The logic gate circuit used in the LG algorithm is as follows.

$$A\ AND\ B = AB \tag{1}$$

$$A\ OR\ B = A + B \tag{2}$$

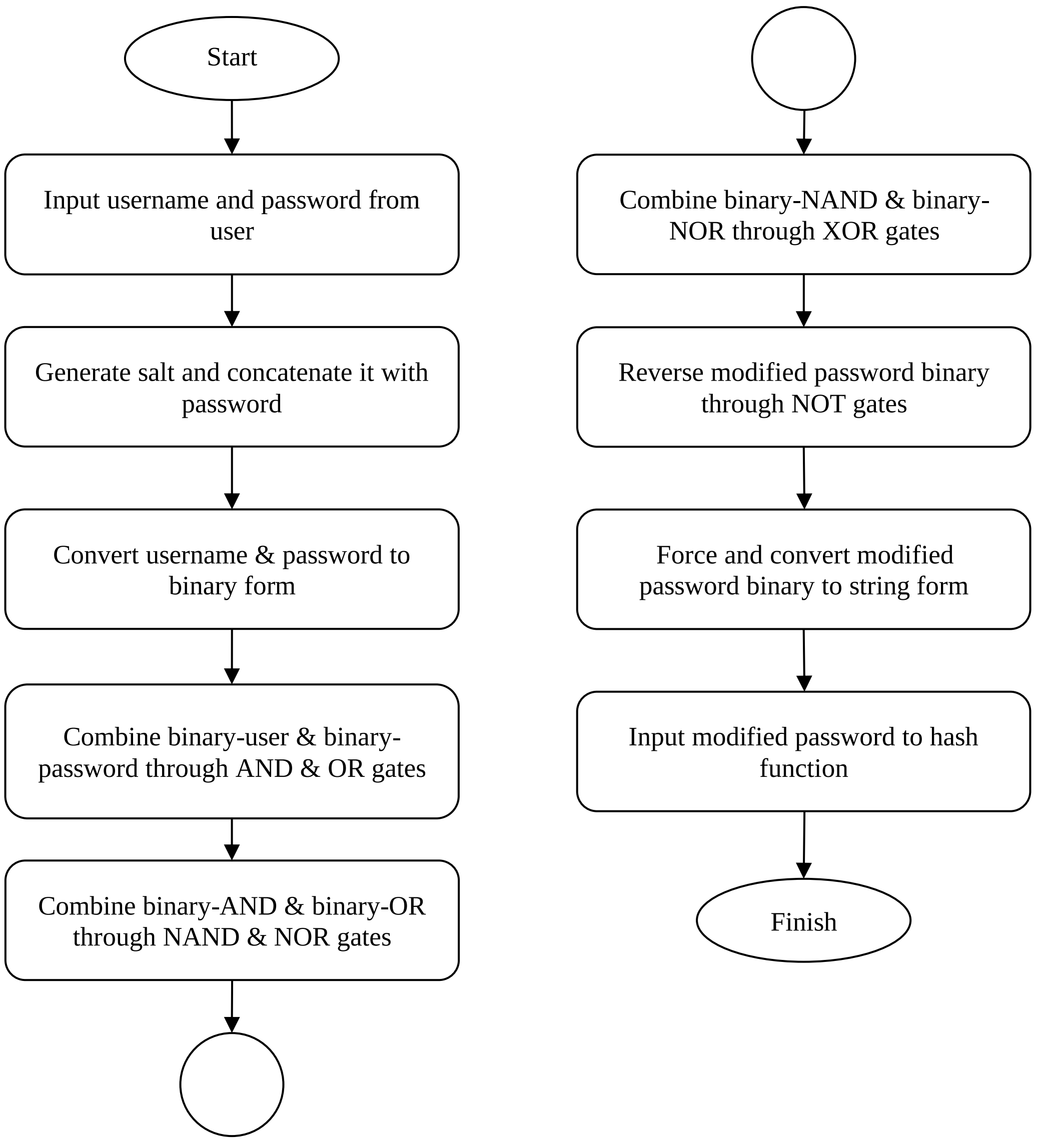
$$AB\ NAND\ (A + B) = \overline{AB(A + B)} \tag{3}$$

$$AB\ NOR\ (A + B) = \overline{AB + (A + B)} \tag{4}$$

$$\overline{AB(A + B)}\ XOR\ \overline{AB + (A + B)} = \overline{AB(A + B)} \oplus \overline{AB + (A + B)} \tag{5}$$

$$\overline{AB(A + B)} \oplus \overline{AB + (A + B)}\ NOT = AB\ (A + B) \oplus AB + (A + B) \tag{6}$$

After passing through a series of logic gates, the resulting binary will be converted back into string form. But sometimes the resulting binary cannot be converted immediately into string form because the arrangement of the resulting binary does not meet ASCII standards. So, the step before converting back into a string is to force the binary to meet the ASCII standard. Finally, the resulting string is entered into the hash function. The flowchart of LG algorithm is shown in Figure 1.



Start
Input username and password from user
Generate salt and concatenate it with password
Convert username & password to binary form
Combine binary-user & binary-password through AND & OR gates
Combine binary-AND & binary-OR through NAND & NOR gates
Combine binary-NAND & binary-NOR through XOR gates
Reverse modified password binary through NOT gates
Force and convert modified password binary to string form
Input modified password to hash function
Finish

Figure 1. Flowchart of LG algorithm

3. METHOD

3.1. Research Setup

The research begins with preparing research equipment in the form of a computer with Intel Core i3-10100T @3.00 GHz processor specifications, 8 GB RAM, and Windows 10 operating system. Computer specifications will greatly affect the testing process. Especially in testing time of attack using the hashcat application, where the application is highly dependent on computer computing capabilities [18].

3.2. Experimental design

This research uses 8 different passwords and salts with the same username. Each password and salt will pass through the LG algorithm with variations in salt placement postfix and prefix. The following is the pseudo-code of the LG algorithm when using the prefix variation, while the postfix variation only needs to change the pseudo-code on line number 4 to $ps \leftarrow salt + p$. Algorithm 1 are showing the prefix variations of the LG algorithm:

Algorithm 1. LG algorithm

```
Input:  a username U
        a password P
Output: a hash value with modified password MP
Start
01: U ← char(00000000)
02: P ← char(000000)
03: salt ← generate_random char (00)
04: ps ← p + salt
05: bin_U ← conv2bin(U)
06: bin_PS ← conv2bin(PS)
07: bin_AND ← empty array
08: bin_OR ← empty array
09: bin_NAND ← empty array
10: bin_NOR ← empty array
11: bin_XOR ← empty array
12: bin_NOT ← empty array
13: ascii_char ← empty array
14: for i to len(bin_U) do :
15:     bin_AND[i] ← bin_U[i] AND bin_PS[i]
16:     bin_OR[i] ← bin_U[i] OR bin_PS [i]
17: end loop
18: for I to len(bin_U) do :
19:     bin_NAND[i] ← bin_AND[i] NAND bin_OR[i]
20:     bin_NOR[i] ← bin_AND[i] NOR bin_OR[i]
21: end loop
22: for I to len(bin_U) do :
23:     bin_XOR ← bin_NAND[i] XOR bin_NOR[i]
24: end loop
25: for I to len(bin_U) do :
26:     bin_NOT ← bin_XOR[i] NOT
27: end loop
28: ascii_char ← "01"+Bin_NOT[3:len(bin_U)]
29: hash_value ← md5(ascii_char)
```

Testing is done with the aim of knowing how the hash resistance that has been strengthened with the LG algorithm when compared to hash without the LG algorithm. Testing is divided into 2, namely time of execution and resistance of attack. Time of execution test is a test of algorithm execution time, which is the time required for a computer to run an algorithm, generally the lower the execution time, the better the algorithm [19]. While the resistance of attack test is a test of hash resistance when attacked or hacked. In this test, the brute force attack method will be used, which is the easiest attack and is widely used by hackers because the method is simple [18]. The resistance of attack test uses hashcat v6.2.6 software to intentionally penetrate or brute force attacks. The more complex or secure a hash used, the longer it takes to hack it [20].

4. RESULTS AND DISCUSSION

The LG algorithm is tested through two types of testing, namely time of execution and resistance of attack. Some examples of passwords and salts used for testing purposes are taken based on [13] which are then limited to 6 characters, while the salt is generated using the secrets python module and is limited to 2 characters. Limiting the number of characters in passwords and salts aims to speed up testing due to limited computer specifications and computing power [21].

4.1. Test dataset

The username used in the research is made uniform, namely "uinjogja", while the list of passwords, salt, and hash values used can be seen in Table 1. The LG algorithm is implemented using the Python language by utilizing Google Colab. Here's an example of how the LG algorithm works. For example, the password inputted is "qwerty" and the salt generated from the secret's module is "b9". In the LG algorithm, the prefix variations of the two variables will be combined into "b9qwerty" then the string and username are converted into binary form which only consists of the numbers 0 and 1. After that the binary username and binary password will be input into a series of logic gates as described in (1) to (6). The result is a modified binary string that is different from the initial username and password binary. However, the modified binary is often not recognized by the ASCII standard. In ASCII the binary first byte of a string is "0" so when the modified binary has a first byte of "1" it needs to be converted back to a value of "0" in order to be recognized by ASCII. After the modified binary is in accordance with ASCII, it will then be returned in the form of a string, so that from the input username "uinjogja", password "qwerty", and salt "b9" a new string will be generated, namely "h/`bujag". The new string is then inputted into a hash function which in this study uses the MD5 hash function. Finally, the resulting hash value is stored in the database.

Table 1. Password, salt, and hash value data

No.	Password	Salt	Hash value prefix (Non-LG)	Hash value prefix (LG)	Hash value postfix (Non-LG)	Hash value postfix (LG)
1	qwerty	b9	a57cffb9f48b37302279cb65c28774c6	3ca9d9ceab8fb2b639c49a73590b519e	b47ed88cae441f0e23fc12722fb98478	9fc747ca76a15894c647d34441cea7fe
2	123456	d8	2b10bb383ccffa1cff33af8187367143	6ac43c69655c04b37d706b2a766e9ed1	38dffc940cc4f21c3cdef30a249576da	c82d53102081487769f77374f9da69cf
3	monyet	ya	bcb00de27df5c0b5c5a7c4d1b1a18dd4	c8e9d120ee747ff54c5841b5d6dc51fa	c5ad972283684079c891c780d3b66342	c8c0f9b7db316890b4e3ad955fd87d45
4	qazwsx	60	773fc2c5b4c3d4549e0b804d80b65330	416cb4084e98a3351209bdb50a131e07	8f82ee16388f7dd17fda7b018b5c7940	ec8f2e10507ef3309eee1f7879c48767
5	qwaszx	e2	aac8ec9aa26314d72eaf11270a62e1cc	f34eab820a2241d9bf6cbc8cf8595026	c8301921bccc19b4e249d94d4a443d5b	7a58fb0be8d7fa40c33581ae35163467
6	harkey	e8	c0c3cbb45de7d1d6e00d3794f1415aed	74cd4547873b1c77456a16a750598338	012fed830ef9e9f9d863616a8bdd2eb4	b9ed9b01df5d7d65ddbb7b5ea9c7f60a
7	654321	ef	77cbe5a08f41ac2baef6d0a67591d45a	4c461860053b441bce17dd596d00833c	4e981513bf3f127d0358fb7b4f0bdff9	a3be7d5cc1ce86ceb346bd36931273c4
8	abc123	67	3a13787a28981863ea00a9a099e9ea1a	b897de258ed1bfd6a8febeac6ff5c3e3	8459e2de27ab485d40a8f8813f03b74f	6e7be7789b71bb171f54f0fb554c05ad

4.2. Time of execution test

Time of execution test aims to determine the execution time of the LG algorithm and the algorithm without LG. The execution time is calculated from the time the username and password are inputted until the hash value is generated. This test uses software. The test results show that the average execution time of the algorithm without LG takes 0.01403197646 seconds while the LG algorithm takes 0.0443432033 seconds. The LG algorithm takes 4 times longer than the algorithm without LG. This data is obtained by testing each password from Table 1 using prefix and postfix variations for 10 repetitions and then taking the average. Thus, the algorithm without LG has better performance than the LG algorithm in the execution time section. However, with an execution time of less than 0.1 seconds, humans will not really notice it when the algorithm is implemented in the real world [22]. The time of execution test data is shown in Table 2.

Table 2. Execution time data

No	Time execution proposed algorithm (s)		Time execution without proposed algorithm (s)	
	Prefix	Postfix	Prefix	Postfix
1	0.0141196250	0.0131189823	0.0410013198	0.0520370006
2	0.0111825466	0.0125463008	0.0456936359	0.0502171516
3	0.0131368637	0.0158793926	0.0417432785	0.0403289794
4	0.0123655796	0.0108537674	0.0394756794	0.0459666252
5	0.0161843299	0.0133821964	0.0389025211	0.0469002723
6	0.0199582576	0.0157468318	0.0447325706	0.0459043979
7	0.0125904083	0.0116624832	0.0402889251	0.0458452701
8	0.0178318023	0.0139522552	0.04235863686	0.04809498787

4.3. Resistance of attack test

The resistance of attack test aims to compare the resistance of the hash value generated without the LG algorithm with the hash that has been reinforced with the LG algorithm. The test data is obtained by testing each password from Table 1 using prefix and postfix variations for 10 repetitions and then the average is taken. The resistance of attack data is shown in Table 3.

Brute force works by trying all possible combinations to guess the password to be attacked. The duration of brute force is highly dependent on how long the character set is and the computational capabilities of the device used by the hacker [21]. The following is the command used to perform brute force penetration using hashcat:

$$hashcat\ -a\ 3\ -m\ 0\ hash\_value.txt --force$$

the above command consists of several arguments, namely:
- $hashcat$: to call hashcat application
- $-a$: to determine attack mode
- 3 : to determine brute force as attack mode
- $-m$: to determine hash function
- 0 : to determine md5 as hash function
- $hash\_value.txt$: to call file contains hash value
- $--\ force$: to ignore any warnings

In hashcat, brute force attacks are similar to mask attacks. The difference between mask attack and brute force attack is in determining the keyspace. Brute force attacks use the default keyspace, while mask attacks use a keyspace that has been adjusted to the hacker's target. Keyspace is a limitation of possibilities that are determined before carrying out an attack [23]. For example, when hackers use brute force, by default hashcat will use a combination of character sets that include uppercase letters, lowercase letters, and all numbers or known as mixalpha-numeric. The use of keyspace allows hackers to speed up the attack process as it minimizes the number of combinations that need to be executed by the computer. However, the use of keyspace may actually prolong the hacking process or even thwart it due to the hacker's mistake in determining the keyspace. Keyspace determination is closely related to social engineering techniques. Social engineering plays a role in the process of finding out how someone designs a password. By knowing how someone designs a password, hackers can determine a narrower keyspace so that the attack time becomes shorter [24].

This research uses a brute force attack with a default keyspace because brute force is more general and objective. In short, when a hash is immune to brute force, it is almost certainly immune to other types of attacks [25]. Here is the keyspace brute force attack on hashcat:

$$“?1?2?2?2?2?2?2?3?3?3?3?d?d?d?d”$$

where,

?1	: $?l?d?u$ (lowercase letters, numbers, and uppercase letters)
?2	: $?l?d$ (lowercase letters and numbers)
?3	: $?l?d*!\$@\_$ (lowercase letters, numbers, and five selected special characters)

The results of the resistance of attack test can be seen in Table 3.

Table 3. Resistance of attack test data

No	Average duration of attack							
	Prefix Non LG (s)	Cracked	Prefix LG (s)	Cracked	Postfix Non LG (s)	Cracked	Postfix LG (s)	Cracked
1	5639.3	Yes	28798.4	No	17774.7	Yes	28651.7	No
2	1450.5	Yes	28793.3	No	11637.6	Yes	28488.8	No
3	6316.8	Yes	28831.6	No	2469.8	Yes	28543.1	No
4	22421.9	Yes	28669.8	No	3277.1	Yes	28705.9	No
5	7506.7	Yes	28965.2	No	7759.2	Yes	28474.1	No
6	6345.7	Yes	20715	No	19217.9	Yes	28568.7	No
7	2161.7	Yes	28450.8	No	22954.2	Yes	28601.5	No
8	775.3	Yes	28456.5	No	2415.6	Yes	28660.3	No

Table 3 shows that conventional or non-LG hashes can be cracked within a certain period of time. The difference in time is due to the varying level of complexity of the hashed password. Based on the experiments conducted, it is known that passwords that use numbers are more vulnerable than passwords that use letters. This is because numbers only have 10 different variations, namely numbers 0-9. While letters have 26 different variations, namely A-Z. For example, password number 1 (qwerty) has a longer hack time than password number 2 (123456). In addition, the order in which passwords are created has a significant effect on the duration of the hack [13]. Putting letters at the beginning of the password shows better strength than putting numbers at the beginning of the password. Password No. 2 when combined with salt (d8123456) has a longer duration than password No. 8 (67abc123) even though the number of letters in password No. 8 is more than No. 2. Meanwhile, the LG-strengthened hash has a much longer hacking duration than the conventional hash. Table 2 shows that the duration of the LG algorithm has a small difference between one password variation and another. This is very different from the non-LG algorithm whose duration varies greatly according to the level of password complexity. The similarity in the duration of the LG hash is because hashcat has run out of password candidates with a length of 8 characters. So, the attack duration on the LG hash is the total amount of time it takes hashcat to test password candidates from 1 character to 8 characters. In the LG hash, all password variations cannot be found in plaintext. This method is much more effective for improving password security because there is no need to force someone to create a complex password for security, but the computer will automatically strengthen the entered password no matter whether it is vulnerable or not [26].

5. CONCLUSION

The development of digitalization makes information security issues very important. Fraud, hacking, data theft that is increasingly happening shows that research in the field of information security is needed. The main focus of this research is on methods to improve password security using hash functions by utilizing the logical gates (LG) algorithm. The LG algorithm is an algorithm that functions to randomize the username and password at the binary level before being entered into the hash function. Tests were conducted to measure the execution time of the LG algorithm and measure the resistance of the hash generated using a brute force attack.

The test results show that the LG algorithm requires four times longer execution time than the use of hash without LG. However, the hash combined with the LG algorithm has a much higher resistance to brute force attacks. Experimental results show that none of the 8 password variations used can be cracked by brute force. Even though in the worst-case hackers can hack the hash of LG, hackers still need to try to find the original plaintext password from the modified password generated by LG. This research can be a solution to build a system that is user friendly, meaning that users do not need to create passwords that are too complicated, but still take into account in terms of information security. Future research is expected to develop the LG algorithm so that it can be embedded in a hardware so that its application in strengthening information security becomes wider.

REFERENCES

[1] S. Samonas and D. Coss, “The cia strikes back: Redefining confidentiality, integrity and availability in security,” *Journal of Information System Security*, vol. 10, no. 3, pp. 21–45, 2014.

[2] K. Y. Chai and M. F. Zolkipli, “Review on confidentiality, integrity and availability in information security,” *Journal of ICT In Education*, vol. 8, no. 2, pp. 34–42, 2021, doi: 10.37134/jictie.vol8.2.4.2021.

[3] A. Sadiqui, “Securing access using AAA,” *Computer Network Security*, pp. 67–78, 2020, doi: 10.1002/9781119706762.ch4.

[4] E. Kuka and R. Bahiti, “Information security management: Password security issues,” *Academic Journal of Interdisciplinary Studies*, vol. 7, no. 2, pp. 43–47, 2018, doi: 10.2478/ajis-2018-0045.

[5] M. Burnett and J. C. Foster, *Chapter 1 Managing users solutions in this Chapter*. Syngress Publishing, Inc, 2004, doi: 10.1016/B978-1-932266-65-8.50034-5.

[6] A. Singh and S. Raj, “Securing password using dynamic password policy generator algorithm,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 4, pp. 1357–1361, 2022, doi: 10.1016/j.jksuci.2019.06.006.

[7] M. Yıldırım and I. Mackie, “Encouraging users to improve password security and memorability,” *International Journal of Information Security*, vol. 18, no. 6, pp. 741–759, 2019, doi: 10.1007/s10207-019-00429-y.

[8] M. C. Ah Kioon, Z. Wang, and S. Deb Das, “Security analysis of MD5 algorithm in password storage,” in *Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA-13)*, Feb. 2013, pp. 706–709, doi: 10.2991/isccca.2013.177.

[9] D. Balu, A. G, T. S, K. V, and P. V, “Implementation of security in login page using salt and pepper algorithm,” *SSRN Electronic Journal*, 2019, doi: 10.2139/ssrn.3358813.

[10] P. Patel, P. Goswami, A. Mishra, S. Khan, and A. Choudhary, “Brute force, dictionary and rainbow table attack on hashed passwords,” *International Journal of Creative Research Thoughts (IJCRT)*, vol. 9, no. 4, pp. 1899–1905, 2021.

[11] U. Rathod, M. Sonkar, and B. R. Chandavarkar, “An experimental evaluation on the dependency between one-way hash functions and salt,” *2020 11th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2020*,

2020, doi: 10.1109/ICCCNT49239.2020.9225503.
[12] Python Software Foundation, "Secrets — Generate secure random numbers for managing secrets," Python Software Foundation, 2024, https://docs.python.org/3/library/secrets.html (accessed Feb 20, 2024).
[13] Sutriman and B. Sugiantoro, "Analysis of password and salt combination scheme to improve hash algorithm security," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 11, pp. 420–425, 2019, doi: 10.14569/IJACSA.2019.0101158.
[14] J. Polpong and P. Wuttidittachotti, "Authentication and password storing improvement using SXR algorithm with a hash function," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 6, pp. 6582–6591, Dec. 2020, doi: 10.11591/ijece.v10i6.pp6582-6591.
[15] A. Karrar, T. Almutiri, S. Algrafi, N. Alalwi, and A. Alharbi, "Enhancing salted password hashing technique using swapping elements in an array algorithm," *International Journal of Computer Science and Technology* (*IJCST)*, vol. 9, no. 1, pp. 21–25, 2018.
[16] F. E. De Guzman, B. D. Gerardo, and R. P. Medina, "Implementation of enhanced secure hash algorithm towards a secured web portal," *2019 IEEE 4th International Conference on Computer and Communication Systems, ICCCS 2019*, pp. 189–192, 2019, doi: 10.1109/CCOMS.2019.8821764.
[17] B. Garg and S. Kaur, "A review of logic gates and its applications," *Journal of Emerging Technologies and Innovative Research*, vol. 6, no. 5, pp. 124–129, 2019.
[18] K. Swathi, "Brute force attack on real world passwords," *International Journal of Research Publication and Reviews*, vol. 3, no. 11, pp. 552–558, 2022.
[19] J. Bielecki and M. Śmiałek, "Estimation of execution time for computing tasks," *Cluster Computing*, vol. 26, no. 6, pp. 3943–3956, 2023, doi: 10.1007/s10586-022-03774-1.
[20] D. P. Putra, I. W. A. P. Putra, and I. G. W. P. Sucipta, "Comparison of password attacks using the tools barshwf, hashcat, and hash cracker console," (in Bahasa), *JTIK (Jurnal Teknik Informatika Kaputama)*, vol. 7, no. 1, pp. 181–187, 2023, doi: 10.59697/jtik.v7i1.62.
[21] D. J. Bernstein, "Understanding brute force," *ECRYPT STVL Workshop on Symmetric Key Encryption*, pp. 10–19, 2005.
[22] R. B. Miller, "Response time in man-computer conversational transactions," *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, vol. 33, no. pt 1, pp. 267–277, 1968, doi: 10.1145/1476589.1476628.
[23] M. Curtin and J. Dolske, "A brute force search of DES keyspace," *Login*, vol. 23, no. 3, pp. 15–25, 1998.
[24] R. Hranický, L. Zobal, O. Ryšavý, and D. Kolář, "Distributed password cracking with BOINC and hashcat," *Digital Investigation*, vol. 30, pp. 161–172, 2019, doi: 10.1016/j.diin.2019.08.001.
[25] I. Alkhwaja *et al.*, "Password cracking with brute force algorithm and dictionary attack using parallel programming," *Applied Sciences (Switzerland)*, vol. 13, no. 10, 2023, doi: 10.3390/app13105979.
[26] H. Hussain, "Password Security: Best practices and management strategies," *SSRN Electronic Journal*, 2022, doi: 10.2139/ssrn.4136333.

BIOGRAPHIES OF AUTHORS

Muhamad Zaki Anbari received his B.Sc degree in physics from Diponegoro University, Indonesia, in 2020. Currently, he is a master of computer science degree candidate at the Department of Informatics, Faculty of Science and Technology, Sunan Kalijaga State Islamic University. His research interests include automation, internet of things, computer networks, and cybersecurity. He can be reached at email: mzakianbari@gmail.com.

Bambang Sugiantoro earned his B.Sc and Dr degrees in computer science from Gadjah Mada University, Indonesia. MT degree in electrical engineering at Bandung Institute of Technology. He is currently the head of the informatics master study program at the Informatics Department of Sunan Kalijaga State Islamic University. His research interests include computer networks, cybersecurity, and machine learning. He can be contacted via email: bambang.sugiantoro@uin-suka.ac.id.