Bcrypt: Secure Password Hashing Function

Daniel Ryan Levyson 13516132
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516132@std.stei.itb.ac.id

***Abstract*—Password is a string which is used to authenticate someone who want to gain access into certain resources. Password should be kept secret by the owner in order to prevent resources stealing or misuse by those who do not have permission. In order to authenticate person by using password, computer should have mechanism to verify the password. Saving plain password somewhere to be used for verification produces security problem. Someone who can somehow find the place where the password is stored and read the password can gain access to any resources in the system. The authentication system should have secure way to store and verify password. Bcrypt is hashing function which can be used to store and verify password securely.**

***Keywords*—hash, encryption, security, password, authentication.**

I. Introduction

Naïve way and also insecure way to create authentication gate of application is by registering new identity followed by the password to verify the identity, then store the password somewhere in one file, and sometime later get the stored password to be matched with received string when doing the verification. Attacker, someone who try to gain restricted access, just need to find the way in the system to retrieve the file which stores all the password used for verification. So, this kind of authentication system is not reliable, when the attacker succeed, authentication become useless.

The more sophisticated way to store the password is using encryption technique which will be explained more in the next chapter. Instead of storing plain password, the password will be stored after it is encrypted. When the attacker gained access to the stored passwords, the attacker can not know the real passwords. It seems good, but to encrypt password, the authentication system needs secret key which is used to change password to encrypted form and reverse. After the attacker got the encrypted passwords, the attacker only needs to search for the secret key somewhere in the system. When the attacker found the secret key, all encrypted passwords can be decrypted and the real passwords are revealed.

Encryption, by its nature, is not reliable to be used for password storage, because the real password definitely can be revealed from the encrypted password by knowing the secret key. In the other hand, hash function, by its nature, produces possibility to store password securely, because hash function is invented to be one way, that means the input of hash function has no way to be retrieved by knowing its output.

Trying to protect password by creating own implementation of hash function is a bad idea, because a hash function can be said secure if it is studied extensively by attempting to break its security. Moreover, the security can not be relied on hiding the algorithm, because hiding the algorithm will produce another concern of security. Through history of computer security, various cryptographic algorithms have been invented. New kind of algorithm was invented whenever the existing one's security had been severely compromised.

MD4, MD5, HAVAL-128, RIPEMD, and SHA-1 are known as cryptographic hash function which are also known suffering from collision attack. SHA-2 is general purpose hash function. SHA-2 has fast calculation. With current power of hardware computation performance, SHA-2 is not recommended for password hashing by concerning brute force attack. SHA-3 is the latest SHA and it is faster than SHA-2, so it has worse concern than SHA-2. In the other hand, Bcrypt has been tested and chosen for a long time for protecting password.

II. Hash Function and Encryption

Hash function and encryption are the main players in cryptography. The main difference between them is whether the input can be known by knowing the output.

A. Hash Function

Hash function basically is a function which takes random size input *k* and map *k* to value *v* which has fixed size. The very simple hash function use modulo operation, it has following form:

$$h(k) = k \bmod m = v$$

In that case, the size of v will depend on m. Because v has fixed size, there are cases when different input k will output the same value v. For example, with m = 10, h(5) and h(15) will have the same output v = 5. That condition is called collision. When using hash function for any purpose besides security, policy can be defined to handle the collision. For security purpose, collision in hash function is weakness and should be avoided for any security usage.

Good algorithm of cryptographic hash function should have following properties:

1. Pre-Image Resistance
   Input value k should be hard to find from known hash

value v.

2. Second Pre-Image Resistance
   For input value k which has hash value v, it should be hard to find another input value which also output the same hash value v.

3. Collision Resistance
   It should be hard to find pair of input value which has the same output value.

In order to proof whether a hash function secure or not, we just need to proof that the hash function does not have one of above properties.

B. Encryption

Encryption is the process of hiding a message by changing its form in such way so that unauthorized parties cannot read the hidden message. The hidden message is called plaintext, and the result of encryption is ciphertext. Authorized parties should have the secret key to hide and read the message. The process of returning the plaintext from ciphertext is known as decryption.

Mathematically, we can write encryption as a function which map plaintext P to chipertext C.

$$E(P) = C$$

In reverse, we can write decryption as a function which map chipertext C to plaintext P.

$$D(C) = P$$

We can also substitute C to E(P).

$$D(E(P)) = P$$

From above expression, we can conclude that decryption function D is actually inverse of encryption function E.

Encryption has been used since the age of roman empire. The Roman emperor, Julius Caesar, use encryption to hide message sent to his governors, The encryption technique is called Caesar Cipher. In Caesar Cipher, every alphabetical characters in the plaintext is substituted by the next three character in the alphabetical order.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

plaintext

D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

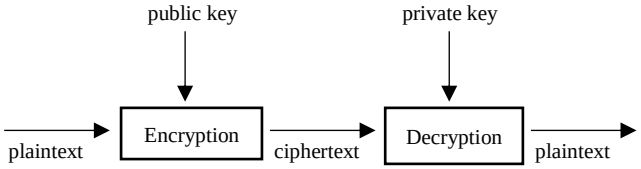
chipertext

Caesar Cipher can be represented more general with “three” substituted by variable K. So mathematically, we can write Caesar Cipher in the expression below.

$$E(P) = (P + K) \bmod 26 = C$$
$$D(C) = (C – K) \bmod 26 = P$$

In this case, K is the cipher key. Because the cipher key used in encryption and decryption is the same, Caesar Chiper is called symmetric-key encryption. If the key used for encryption and decryption is different, the encryption is called asymmetric-key encryption or also known as public-key encryption.

In asymmetric-key encryption, there are public key and private key. As the name suggest, public key is for public use, it is not secret for everyone. Public key is used to encrypt the message. Private key is used to decrypt the ciphertext. Therefore, in asymmetric-key encryption, everyone can encrypt the message because the key used to encrypt is public. But only authorized parties can read the hidden message.



RSA algorithm is known as one of asymmetric encryption implementation. RSA algorithm has three parts: generating public and private key pair, message encryption, and message decryption. DES is known as one of symmetric encryption algorithm. DES is no longer considered as secure, but it becomes the fundamental understanding of block cipher.

Block cipher is a function which takes two input: k-bit string and n-bit string, and then returns n-bit string. k-bit string is a symmetric key for block cipher. Block cipher is known as powerful technique behind the strong encryption algorithm. In block cipher, there is basic component used to obscure the relationship between the key and the ciphertext which is called S-Box. Besides DES, another encryption algorithms that use the concept of block cipher are AES and Blowfish. Both are considered secure encryption algorithm until today.

	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x
0yyyy0	14	4	13	1	2	15	11	8
0yyyy1	0	15	7	4	14	2	13	1
1yyyy0	4	1	14	8	13	6	2	11
1yyyy1	15	12	8	2	4	9	1	7

Figure 1 DES S-Box
source: https://apprize.info/security/cryptography/9.html

C. Password Hashing Function

We already talked about two way function that is not good to be used in creating password storage. Using two way function for protecting password requires the system to store the secret key to be used in verifying password. Problem will occur when attacker can gain access to passwords and the secret key. All the encrypted passwords can be decrypted using the secret key.

Instead of using two way function, we can use one way

function to protect password. The authentication system does not need to know the plain password, the system only needs to enter password into the function and compare the result with the one stored in password storage. It means we can actually use hash function to protect password.

There are several known attacks to break password hashing function besides brute force such as preimage attack, collision attack, dictionary attack, and rainbow table attack. Any hash function that is weak to preimage attack and collision attack should be avoided for further usage in security. Dictionary attack is faster version of brute force, because it narrows the space of guessing by registering known words in dictionary to be used in brute force.

Because human always needs to remember the password, the password should be not too different from any meaningful and familiar words for human. Human's password is said to have high entropy. Even though human combines the password with number and non-alphabetical character, the entropy is still relatively high. Suppose someone's password is "?/Qu1cKbR0wNf0X/?". The password is hard enough but we can still see that it is created by modifying the words "quick brown fox". This is why dictionary attack exists.

Dictionary attack is faster than brute force but it still needs time. Rainbow attack is actually dictionary attack which decreases the processing time significantly but in the same time increases the required disk space. Rainbow attack precomputes the dictionary and it makes the process of knowing the plain password from its hash becomes much faster, because the process only compare the precomputed hash and the password hash, when any hash is matched, than the attacker knows the plain password before the password is precomputed. Even though this technique requires the attacker to gain access to all the hashed password, but good password protection should be able to become the next layer of security.

To overcome the weakness of human's password, an attempt is made to lower human's password entropy. The idea is generating another string to be combined with human's password so that the combination will produce more rainbow table attack resistant hash. The generated string should be random for each password and long enough. The longer the string, the lower the entropy of password. This generated string is called Salt. Salt can be put before or after the password. Salt is also useful to make same password to have different hash. The authentication system should also store the Salt to be used when verifying password and it is considered as safe, not compromising security.

III. Bcrypt Password Hashing

As stated in the introduction, SHA-2 is actually good general hash function, but it is not secure enough to be used for hashing password. Hardware computational power is increasing over time. There might be possibility for attacker to brute force password by using combined high-end hardware computational power. Moreover, quantum computer exists nowadays which significantly faster than classical computer when doing brute force. In order to make brute force almost impossible, we should use slow hash function to hash password.

Bcrypt algorithm is hash function which has expensive key setup phase. Bcrypt can follow the increasing computational power of hardware, because it can be configured to be slower by increasing the number of iteration. Bcrypt utilizes Blowfish to setup the key. Blowfish is notable as complex symmetric-key block cipher. Following parts will discuss Blowfish and Bcrypt in more detail.

A. Blowfish

Blowfish is symmetric-key block cipher, designed by Bruce Schneier, which is known by its large key-independent S-Boxes and highly complex key schedule. Blowfish has 16 rounds Feistel network. A round-specific data derived from the cipher key is called a round key. A key schedule is an algorithm that calculates all the round keys from the key. Blowfish has 64-bit block size and 32 bits up to 448 bits key length.

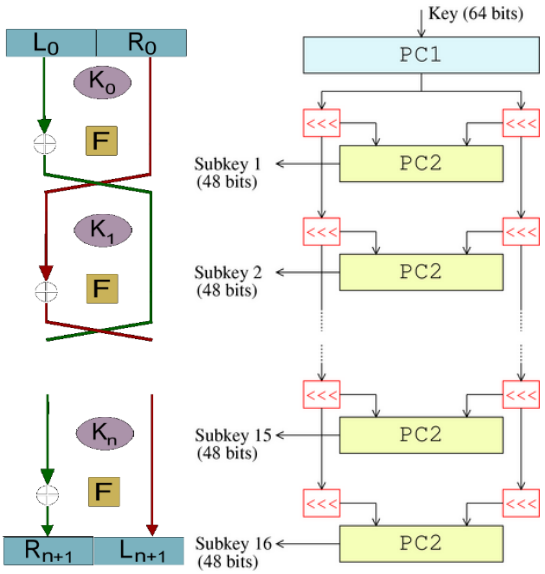


Figure 2 Feistel Network and Key Schedule
source: https://en.wikipedia.org/wiki/Feistel_cipher; https://en.wikipedia.org/wiki/Key_schedule

For every round, Blowfish algorithm does the following actions:

1. XOR the left half of the data with the r-th P-array entry.
2. Use the XORed data as input for Blowfish's F-function.
3. XOR the F-function's output with right half of the data.
4. Swap L and R.

Blowfish's F-function will split 32-bit input into four eight bit quarters. S-boxes will transform the quarters which is 8-bit length to 32-bit output. Then, the output is added module 2^{32} and XORed to produce the final 32-bit output.

Following are the steps of key schedule algorithm in Blowfish:

1. Initialization of P-array and S-boxes with values derived from hexadecimal digits of pi (first 12 digits of pi in hexadecimal 3.243F6A8885A3…..).

2. Byte by byte, secret key is XORed with all the P-entries in order.
3. A 64-bit all-zero block is then encrypted with the algorithm as it stands.
4. The resultant ciphertext then replaces P1 and P2.
5. The same ciphertext is encrypted again with new subkeys, and the new ciphertext replaces P3 and P4.
6. This process will continue to replace the entire P-array and all the S-boxes.

Blowfish algorithm will run 521 times to generate all the subkeys. Blowfish only needs about 4 KB space of RAM. Its small usage of RAM makes Blowfish possible to be used in embedded systems.

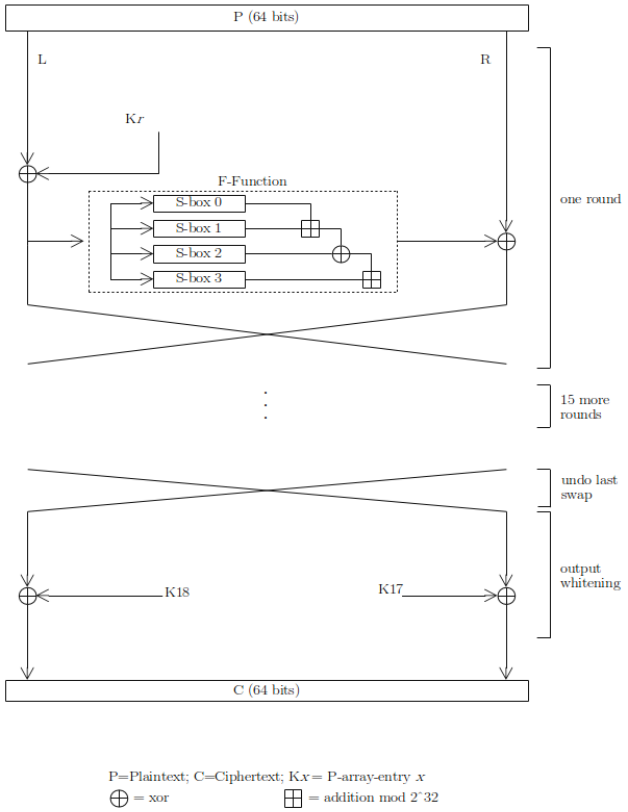


Figure 3 Feistel Network in Blowfish
source: https://en.wikipedia.org/wiki/Blowfish_(cipher)

B. Bcrypt

Bcrypt is hash function designed by Niels Provos specifically for password hashing. It was developed to harden password storage of Unix authentication system. Bcrypt is adaptive to computational power of hardware. To make Bcrypt faster or slower, the iteration count can be configured to the value which is secure enough but also fast enough to do verification.

Bcrypt take advantages of the expensive key setup in Eksblowfish. Eksblowfish refers to expensive key schedule blowfish, it is a cost parameterizable and salted variation of the Blowfish block cipher. Eksblowfish takes three parameters: cost, salt, and key. The cost parameter is what makes this algorithm adaptive. Increasing the value of cost creates more expensive key schedule to compute. The key parameter is user-chosen password. Eksblowfish will return set of subkeys and S-boxes.

Following are steps of Eksblowfish algorithm:

1. Initialization by copying the digits of number π first into subkeys, then into S-boxes.
2. Expand the key by modifying the P-array and S-boxes based on the value of the 128-bit salt and the variable length key. It XORs all the subkeys in the P-array with encryption key. The i-th 32 bits of key are XORed with i-th of P.
3. Encrypt the key (password) using Blowfish encryption algorithm for 2^{cost} times.

After getting set of subkeys and S-boxes from Eksblowfish, Bcrypt encrypts the text "OrpheanBeholderScryDoubt" repeteadly for 64 times in Electronic Codebook mode, that is one of block cipher mode of operation. Result of encryption is then concatenated with the cost and salt to provide information for later verification process.

Even though there are several encryption process, Bcrypt is not encryption algorithm. It does utilize encryption algorithm, but there is no way to know user-chosen password from the result of Bcrypt function. Note that, instead of trying to hide user-chosen password by transforming the password to non-meaningful string, in Bcrypt, the password is used as encryption key by Blowfish algorithm in the key setup process. So the password contributes in transforming the text "OrpheanBeholderScryDoubt". Authentication system does not need to save user's password which becomes the encryption key, so Bcrypt is not encryption algorithm, it behaves like hash function.

Bcrypt has following scheme (based on the latest scheme, February 2014):

$2b$[iteration]$[salt][hash value]

Following Bcrypt's password string has cost parameter equal to 12 which indicates 2^{12} key expansion rounds, salt "ZMqo8uLOikgx2eNcRZoMy9", and resulting hash "xad68L7lJZdL1ZAgcfl7p92hWyIjldG":

```
$2b$12$ZMqo8uLOikgx2eNcRZoMy9xad68L7lJZdL1ZAgcfl7p92hWyIjldG
```

The length of salt used in Bcrypt is 128 bits with Radix-64 encoding, so it is 22 characters. The length of hash value of Bcrypt is 184 bits, it is 31 characters length with Radix-64 encoding. So in total, authentication system will store string with length 58 plus digit of iteration number. For the input, user-chosen password length should not be longer than 72 bytes or the password will be truncated.

Bcrypt has been implemented in many programming languages such as C, C++, C#, Go, Java, Javascript, Perl, PHP, Python, Ruby, and other languages. It was originally used for OpenBSD authentication system, but nowadays it has been widely used to securely store password in many web applications.

IV. Example and Test

We are already known how Bcrypt works and also how Blowfish which is utilized by Bcrypt works. We need to know the example case of Bcrypt's usage and also proof that it is adaptive to hardware computational power.

Below is the code written in Javascript trying to demonstrate the usage and the result of Bcrypt algorithm. The code below use Bcrypt library from github.com/kelektiv/node.bcrypt.js. What the code does is hashing a password "?/Qu1cKbR0wNf0X/?" with different value of rounds, starting from 4 until 20. We will see how big the difference of processing duration among those various rounds value and also observer the result of the hash function.

```
const bcrypt = require('bcrypt')

const password = '?/Qu1cKbR0wNf0X/?'

for (let saltRounds = 4; saltRounds <= 20; saltRounds++) {
    let start = Date.now()
    let salt = bcrypt.genSaltSync(saltRounds)
    let hash = bcrypt.hashSync(password, salt)
    let duration = Date.now() - start
    console.log(saltRounds, salt, hash, duration)
}
```

R	Result	ms
4	$2b$04$jGn2iQK6Pv4zi.RiUsQzou**VeP1x3szXjdw7mePyTNt7BS5uWheS8u**	1
5	$2b$05$tJzFpr0ulW9NRzAvP7ZXw.**sITx67tMm9LE3pjYYdWhKWeXF7AC3Aq**	2
6	$2b$06$gST9qN7Cdjp5Kydo1ycd4.**TT4MuEpXC5YR5R1v/6Kl7l9xFk5NXdC**	5
7	$2b$07$9YT02ZA1FMDDJHeBC4AHQO**0EhS00iQrFrIoW2PDTbNBNQnKC3NcaC**	9
8	$2b$08$qoqqmqCv8vVPF/I75FM.be**kZV68yo9JHRgNBfh.rcvk5Xcgj.3ArG**	17
9	$2b$09$YKy/XQt6j55K2dfT7B.MIe**pXjVnm87KjQETa8EBivAxdKWG.tMkb6**	34
10	$2b$10$t9GOsh/PCjaJc7caC./SgO**cBAJBs.io1GgJtbBcJhOYYeGwGnsBTC**	66
11	$2b$11$b4tAAqnQhpSGnUev3vjGk.**dAcOiWM.8ZJ/9uU8H4y4xQD3rdlK.LO**	131
12	$2b$12$qKZU/muNnbnWPuWiK9Oaie**gthbcoSvw6N0nDzWN.mshPHa9DbK.16**	252
13	$2b$13$ZFbSZyn1qkkXgqVZmiPhcu**3DO5EuOeM8b5EY3Ec/PA2UPqu0y4ewG**	529
14	$2b$14$.Wyufgq0b2.PRA06avAF6O**8YEHlEFHxoWmtsi8SIDUicy05k3l5ny**	1050
15	$2b$15$VdilfJOtX2RV7gE7DDKN/e**R6WKwcBu4R3cnp1FO6MjaUMHjUEjbEC**	2049
16	$2b$16$zm3rZiRUrMiewXqIPVWd2u**0N4C.CN4UxzM6E3fuxrUqs0hrpLKtk2**	4842
17	$2b$17$iFYcV0p36n3sJ08jeSMsj.**BFMFytopTQS7QgqlXlTUZ7mUFoOkW3u**	8194
18	$2b$18$AKm.jbepZVU0tpDNMAe5Nu**LMA9UZ.aNtAp6nCLAADIAHvslvwmjQ6**	18295
19	$2b$19$9jnC6HFVtOHoELnDpdls0.**IYbRAD/G2FJE1WJ/BWy481Mm5fvKh4q**	36211
20	$2b$20$UthOBhRytt2HEd7zUEgwO.**NNvJbrYiu2kOTR8oaLH5WeM6sXLVrZy**	73530

The first column is value of rounds used to get the hash. The next column is the result of Bcrypt. The last column is the processing duration in millisecond. The bold substring in the second column is the encryption result of text "OrpheanBeholderScryDoubt", whereas the non-bold substring is the hash identifier including: crypto prefix "$2b$", the rounds value, and the generated salt.

We can see that, same password will produce different result. It is caused by the random generated salt. Even same password and same rounds value will also produce different result. The processing duration is increasing about twice every time rounds value increased by 1. The last trial with rounds value equal to 20 has processing duration more than 1 minute.

Now we have proof that Bcrypt can be scaled to follow increasing computation power of hardware. The better the performance of hardware, we can increase the rounds value, so that brute force will be kept almost impossible. It is also proof that we cannot use general hash function such as SHA-2, because it does not scale with increasing hardware computational power. With the same environement, running SHA-2 hash function only needs 1 millisecond, same as Bcrypt with rounds value equal to 4, and will always stuck there.

V. Conclusion

Password is become the main chosen way to do authentication, although there are various ways of authentication besides using password. Because of its extreme significance in securing access to resources, a system should have reliable authentication gate. Footprint of password should be kept inside reachable place by the authentication system no matter how does it look like as long as the authentication system can verify corresponding password. When the password is reachable by the authentication system, it means attacker also has possibility to reach the password. Therefore, to give another layer of security, the password should be stored in such way so that the attacker get nothing after successfully stealing the passwords.

Hash function is good choice to change the form of password so that password is not stored with its original form. Hash function also give possibility to do password verification. Good hash function should be resistance to preimage attack, second preimage attack, and also collision attack. For password hash function, it should also slow enough and should be able to scale to follow increasing computational power of hardware to resist brute force, dictionary, and rainbow table attack.

Bcrypt has already proven its security and scalability. By utilizing strong symmetric-key block cipher, Blowfish, Bcrypt use expensive and complex setup key, so that it always has constantly changing S-boxes. Its Eksblowfish makes it scalable by providing parameterized cost.

VII. Acknowledgment

The author would like to thank Mr. Dr. Ir. Rinaldi Munir, MT as lecturer in Discrete Mathematics class who provides the basic

knowledge needed to write this article and also his book always fills the missing knowledge for author. The author also would like to thank the contributors of node.bcrypt.js who provide useful library for author's project which gave inspiration for author to write about this topic.

REFERENCES

[1] Munir, R. (2016). Matematika Diskrit. Bandung: INFORMATIKA.
[2] Niels Provos, David Mazières. (1999). A Future Adaptable Password Scheme. Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference.
[3] Schneier, Bruce. (1994). Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). https://www.schneier.com/academic/archives/1994/09/description_of_a_new.html. Accessed on December 7, 2018.
[4] Bellare, Mihir; Rogaway, Phillip (11 May 2005), Introduction to Modern Cryptography (Lecture notes), chapter 3.
[5] Schneier, Bruce. (2005). Cryptanalysis of SHA-1. https://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html. Accessed on Deccember 5, 2018.
[6] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, Cryptology ePrint Archive Report 2004/199, 16 Aug 2004, revised 17 Aug 2004. Retrieved December 5, 2008.
[7] Kennedy, David. (2015). Of History & Hashes: A Brief History of Password Storage, Transmission, & Cracking. https://www.trustedsec.com/2015/05/passwordstorage. Accessed on December 5, 2018.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Desember 2018

Daniel Ryan Levyson 13516132