Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms

Levent Ertaul, Manpreet Kaur, Venkata Arun Kumar R Gudise
CSU East Bay, Hayward, CA, USA.
levent.ertaul@csueastbay.edu, makur94@horizon.csueastbay.edu, varunkrg@aol.com

***Abstract*- With the increase in mobile wireless technologies, security breaches are also increasing. It has become critical to safeguard our sensitive information from the wrongdoers. So, having strong password is pivotal. As almost every website needs you to login and create a password, it's tempting to use same password for numerous websites like banks, shopping and social networking websites. This way we are making our information easily accessible to hackers. Hence, we need a strong application for password security and management. In this paper, we are going to compare the performance of 3 key derivation algorithms, namely, PBKDF2 (Password Based Key Derivation Function), Bcrypt and Scrypt. We have developed an android application by which we will measure the complexity and time required to generate the hash of the password. This will give us an idea about the effectiveness of these 3 algorithms. Performance comparison and analysis is also given in this paper.**

I. INTRODUCTION

Cryptographic hash functions have a feature of determinism which means they will take large amount of data as input and generate a fixed length output [1].The fixed length output is also called message digest or hash. It is not possible to recreate the input data from its hash value. These one-way hashing functions have following properties:

- Computes hash of any message quickly.
- Not possible to regenerate original value from its hash.
- Not possible to change the message without modifying the hash.
- No two messages have same hash.

Usually, user chosen passwords are hashed and stored in the database. These hashed passwords are then encrypted using cryptography algorithms. Typical hash functions are MD5, SHA1 and SHA256. Hashed passwords are vulnerable to Dictionary/Rainbow table attack [4] and Brute Force Attack [4]. Applications of hash functions are enormous in cryptography and programming practice. Encryption and hash functions are two related and complementary fields and are not the replacement technologies for one another. PBKDFs are generally designed to be computationally insensitive, so that it takes relatively long time to compute. Hence, it is tough for the hackers to retrieve the password.

Hashing algorithms are used for mapping of variable length data to fixed output, retrieving data from the database or data lookup. Whereas, Cryptographic hash functions are used for building blocks for HMACs which provides message authentication. They ensure integrity of the data that is transmitted. Collision free hash function is the one which can never have same hashes of different output. If *a* and *b* are inputs such that *H (a) =H (b)*, and $a \neq b$.

User chosen passwords shall not be used directly as cryptographic keys as they have low entropy and randomness properties [2].Password is the secret value from which the cryptographic key can be generated. Figure 1 shows the statics of increasing cybercrime every year. Hence there is a need for strong key generation algorithms which can generate the keys which are nearly impossible for the hackers to crack. So, PBKDF2, Bcrypt and Scrypt provide a solution to this issue.

PBKDF2 works on pseudorandom function (PRF) with fixed number of iterations, denoted as C. It takes salt, user chosen password and desired length of output key as an

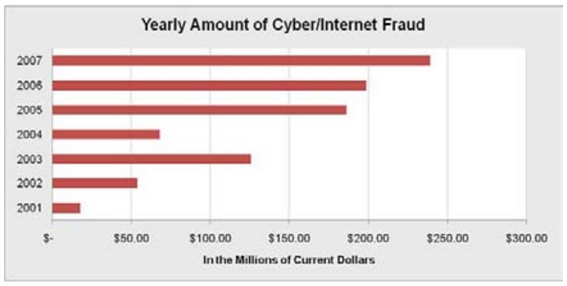


Figure 1: Cybercrime every year

input. By repeating the process (PRF) to the number of iteration count, the cryptographic key is generated [9].

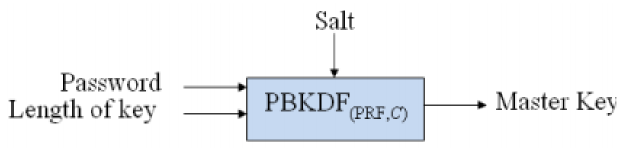


Fig 2: A generic diagram of PBKDF2

Increasing the computation makes it complex which is known as key stretching and resists the brute force and dictionary attacks.

Bcrypt has expensive key setup schedule and is a cross platform encryption utilty. It uses EBC (Electronic Code Block) and is a cross platform encryption utility.It divides the input data into subkeys and then starts block encryption of the subkeys.The resultant is encrypted subkeys appended with some value.This process will keep on repeating until all the subkeys are hashed[3].Bcrypt has lot of computation which makes it extremely invulnerable to dictionary and brute force attack.Hence,bcrypt is very secure to use.

Scrypt is hashing algorithm which makes use of password based key derivation function. It generates large vector of pseudorandom bit strings.It takes large amount of memory and cpu cost. Many pseudorandom numbers are generated in the whole process that are stored in random access memory so it occupies immense memory space.It is considered as an expensive algorithm as each element that is generated during the time of hashing requires more memory and computation.This is very secure as it is very hard for attackers to crack this hashed message due to lack of resources and memory[11].

Section II focuses on the working of the key streching algorithms (also called Salted hashing) and how they are different from the traditional hashing. Section III contains explanation about the algorithms of PBKDF2, Bcrypt and Scrypt.

II. TRADITIONAL HASHING VS KEY DERIVATION FUNCTION

Passwords are never stored in plain-text format, so to store passwords in database, hash of the passwords are generated. Hash algorithms are one-way functions. They can turn any amount to variable data to fixed length output. The generated output is impossible to reverse to get the plain-text. Hence, it provides a level of security as there is no threat to your data even if the password file is compromised. Examples of these cryptographic hash algorithms are SHA256, SHA512, WHIRLPOOL and RipeMD[1][6]. This is traditional way of storing and securing the passwords. This type of hashing is still susceptible to cracking as there are more applications and resources available to do evil to your data. These types of passwords can be recovered with Brute Force and Dictionary attacks, lookup tables, reverse lookup tables and Rainbow tables.

Salted hashing provides the security from these attacks and make password cracking extra difficult. Salt helps us to randomize the hashes [12]. It is a random string of bits which can be prepended or appended to the user chosen password before hashing the password [7]. It makes all the stored hashes unique as the salt generated different every time and needs not to be a secret. Adding salt to the passwords before hashing makes lookup table and rainbow table attacks ineffective. But passwords are still open to brute force or dictionary attacks. These few attacks are still effective because of High-end graphic cards and custom hardware that is able to compute billions of hashing per second.

Password cracking can be made harder by key stretching. Purpose of key stretching is to add computation to the process of key generation to make the algorithm slow. Hence, in PBKDF2, Bcrypt and Scrypt we have a security factor which is iteration count. With the value of iteration count, we can decide how slow we want the algorithm to be. Iteration count can be chosen such that it makes hash function slow enough to impede the attack, but still fast enough not to cause a noticeable delay for the user.[8]
Following are the common features for the key derivation functions:

- Deterministic Functions.
- One-way Functions.
- Slow hashing functions.
- Key stretching functions.

Next section gives information about the algorithms for PBKDF2, Bcrypt and Scrypt.

III. ALGORITHMS OF PBKDF2, BCRYPT AND SCRYPT

PBKDF2, Bcrypt and Scrypt are the latest key derivation functions. They provide strongest password security. They have key stretching and salted hashes which makes very tough for the hacker to break into the security of these password hashes.

PBKDF2 is a key generation algorithm and it is a part of RSA Public Key Cryptographic Standards (PKCS #5 V2.0).It is also published by Internet Engineering Task Force as RFC 2898.This Key derivation function is designed to be slow by increasing its computation and complexity. So it is also called key stretching function which safeguards it from many cryptographic attacks. It applies a pseudorandom function such as cryptographic hash, cipher or HMAC.
PBKDF2 has following inputs:

- Password (P): User chosen password which is needs to be hashed.
- Number of Iterations (c): The algorithm is iterates this number of times before returning the hash password. This parameter slows down the algorithm and helps to safeguard against security attacks.
- Salt: Salt is a random number which is appended to the password to make it more secure.
- dkLen: Length of the derived key in octets which is at most (2^32 -1)* hlen (digest size of hash function).

Output:

- Derived key (DK)

The effectiveness of dictionary and brute-force attacks is reduced because it is a slow algorithm and it takes lots of time for the attacker to get the original password. The salt value and number of iterations parameter makes it even harder to guess the password. The salt value which is added to password lessens the capability of rainbow attack. [4]
Key Derivation process
The key derivation function accepts the following input parameters.
Key = PBKDF2(PRF, Password, Salt, c, dkLen)
PRF is a pseudorandom function which takes two input parameters and output length hLen .Password is the master password from which a derived key is generated.
Each hLen-bit block Ti of derived key DK, is computed as follows:

DK = T1 || T2 || ... || Tdklen/hlen Ti = F (Password, Salt, c, i)

The function f is known as xor (^) of c which iteration of chained PRFs.

F (Password, Salt, c, i) = U1 ^ U2 ^ ... ^ Uc

Where: U1 = PRF(Password, Salt || INT_32_BE(i)) U2 = PRF(Password, U1) ... Uc = PRF(Password, Uc-1).

Basically, in order to recover passwords from the system attackers either brute force technique or the dictionary attack. The intruders estimate the passwords by using some hashing techniques and then differentiating the hashing results to store them to recognize if the results are similar with the user's password in the system. Normal cryptographic hash functions can be used by attackers to guess huge number of passwords per second.

Here comes the PBKDF2 which helps users in every way to lessen such privacy attacks and at the same time it becomes hard for intruder to guess the original passwords. They would be successful in getting few thousand passwords per second as far as PBKDF2 systems are concerned. So, PBKDF2 systems create some crucial situations that would make it impossible for hackers to attack them. Also, if we use salt in the hashing process, the ability to use precompiled hashes would be reduced by the passwords itself for attacks in the system.

PBKDF2 Strategies:

- Computing output in host endianess.
- Vigorously lines everything in the inner loop.
- Buffering and padding are not required inside loop.
- Parallelization can be done for lengthy outputs.
- Minimal copies can be made inside loop.

Where PBKDF2 is being used?

- AES Encryption scheme by WinZip.
- For secure wireless networks through Wi-Fi Protected
- Firefox Sync
- Cisco IOS
- For the protection of user passwords and pass codes in Apple's IOS mobile operating systems.

Bcrypt

Bcrypt is the key derivation function for the passwords being designed for the systems. It is one of the most popular and powerful algorithms which is quite successful in restraining the password hacking and other unwanted attacks in the system. It functions similar to Blowfish Block Cipher. Therefore, this bcrypt is based on EKSBlowfish procedure which strengthens the password encryption in order to avoid attacks. It encrypts 192 bit magic values [5] by using 128-bit salt. Above all, bcrypt is using expensive key setup in Eksblowfish.

There are two phases in which bcrypt algorithm is being executed. In the very first phase, the Eksblowfish Setup is called with the salt, password and cost to process the Eksblowfish state. However, the expensive key schedule consumes lots of time. On the basis of 192-bit value of OrpheanBeholderScryDoubt is encrypted at 64 times from the previous phase to the particular state using Eksblowfish in ECB mode. The 128-bit salt would be concatenated with final result of encryption loop to provide the output. One issue with bcrypt is that salting is not good enough to hash a string. But Still Bcrypt keeps on fighting with processing power using iterations of encryption. This process is known as work or cost factor.

How Bcrypt Works?

Bcrypt makes use Blowfish encryption algorithm consisting of keying schedule [6]. It includes work factor as well. The hashed value being created comprises of the steps listed below:

- First of all, Bcrypt algorithm version identifier is executed
- Cost factor is included
- Every 16-byte salt value in Bcrypt is encoded in a adapt Base-64 (22 characters)
- cipher text (remaining characters) is produced

For example Bcrypt hash is;

$21$10$MN9CW1vkR2xSXT8jqchug.wvLZbl4mtapxK0u/SLbTcgl9Ldzlq60

It will be shown as follows:

- Crypt algorithm version is 2a
- It uses a Cost factor of 10
- Salt is MN9CW1vkR2xSXT8jqchug.
- The cipher text comes out to be is wvLZbl4mtapxK0u/SLbTcgl9Ldzlq60

We are implementing Java here via Spring-security 3.2.5. Several methods are being tendered by Bcrypt object that makes the usage of API relatively easy. GenSalt is one of the general methods processed in the salt generation. The various kinds of genSalt are following:

- genSalt() – In the process, gensalt() would use a cost factor of 10. In order to generate the 16-byte salt for hashing, this procedure would take a new instance of Secure-Random class.
- genSalt(int log_rounds) – It uses an updated Cost Factor to the value that is stated as integer and to create the 16-byte salt , it uses a new instance of Secure-Random class.
- genSalt(int log_rounds, SecureRandom) – It also uses a modified Cost Factor with the value of the specific integer. In order to create the 16-byte salt it uses the provided SecureRandom -instance.

Hashed passwords can be generated easily once the salt is implemented by simple call in program;

- hashpw (String password, String salt) – The given password is hashed with the given salt.

Algorithm

```
bcrypt(cost, salt, input)
state   EksBlowfishSetup(cost, salt, input)
ctext  "OrpheanBeholderScryDoubt" // 64-bit bocks
repeat (64)
```

ctext EncryptECB(state, ctext) //encrypt using standard Blowfish in ECB mode

return Concatenate(cost, salt, ctext)

Implementations are fluctuated in the approach of changing passwords into initial numeric values here. It leads to reducing the strength of passwords sometimes which contains special characters. [6]

Cost Factor

The real value of Bcrypt gives us the Cost Factor. The Present processors and technologies permits us to generate the brute force attacks easily that can select targets straight in the system. The increase in the cost factor is exponential in the cost factor (as 2^cost factor).The Cost factor being created in every process is stored as hashed value.

Scrypt

Scrypt is a key derivation function which is computational intensive and it consumes more time to compute. For every operation, the authenticated users have to perform the function and the time taken would be negligible as well [11]. High level of security is being provided to users and their data through this function that makes it next to impossible for the intruders to crack the original passwords. Even it is so powerful that it makes such complicated situations that the attacker won't get the actual passwords if he makes millions of guesses too [10].

The guessing technique used is Brute Forcing. However, script functions are developed to avoid the attack attempts by increasing the requirements of resources of algorithms. Specifically, this algorithm is implemented in such a way that it should use the highest amount of memory allocated to other password based Key Derived Functions. This can be done by making both size and cost of hardware implementation for any particular device more expensive and at the same time parallelism used by an attacker should be of minimum amount for the limited amount of financial resources[10].

```
Function scrypt(Passphrase,Salt,N,p,dkLen):
(B0 ... Bp−1) ← PBKDF2HMAC_SHA256(Passphrase, Salt, 1, p * MFLen)
    for i = 0 to p-1 do
        Bi ← SMix(Bi,N)
    end for
Output ← PBKDF2HMAC_SHA256(Passphrase, B0 || B1 ... Bp−1, 1, dkLen)
```

MFLen - Length of block mixed by SMix()

hLen – length of produced by HMAC_SHA256()

dkLen- output length, positive integer satisfying dkLen ≤ (232− 1) * hLen.

N- CPU/memory cost parameter.

p - Parallelization parameter; a positive integer satisfying p ≤ (232− 1) * hLen / MFLen.

We have applied these algorithms in our application to check how much time they are taking to create the hash. Also, we can analyze the behavior of each algorithm as we are varying the even parameters in the 3 algorithms. Next section will explain about the application.

IV. IMPLEMENTATION OF PBKDF2,BCRYPT AND SCRYPT

We have implemented an android application where we are securing the contacts numbers saved on the android device by using these 3 algorithms. User can select any one of these 3 algorithms to generate the encrypted contacts. We are measuring the performance of the algorithms by varying the input parameters which are common in all 3 algorithms. Here are the software (Table I) and hardware specification (Table II) for the application.

Table1. Software Specification

Type	Specification
Operating system	Windows, Linux
Language	Java Server Pages
Version	JDK 1.7
Back-end	MySQL(XAMPP Server)
Server	Apache
Tool	Netbeans, Elipse

Table II. Hardware Specification

Type	Specification
Processor	PENTIUM IV
Clock Speed	2.7 GHZ
Ram Capacity	1 GB
Hard Disk Drive	250 GB
Monitor	15 VGA Color

'**Contact Securing Application'** is an android application where user can register with his credentials and will use those credentials to login in the application. That password will be stored in hashed version which will be in non-readable format to humans. User can use that password to decrypt his contacts later in the application. This user chosen password will be used as input to the one of the 3 key generation algorithms (PBKDF2, Bcrypt and Scrypt).The cryptographic key generated by these algorithms will be used as input to AES algorithm (128 bit) which is used to encrypt the contact numbers.

Figure 3 is the Login page of the application. It will appear when user runs the application. Here, we have Email and password as text fields, sign-in and register button. User needs to create an account if he does not have one. User can redirect to registration page while hitting register button; otherwise he can directly sign in. Valid credential will lead him to the next screen. An error message will appear in case of incorrect credentials.

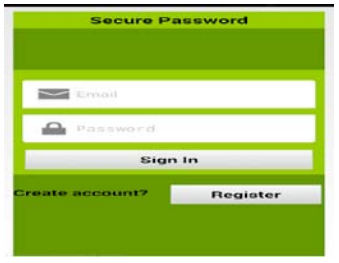


Figure 3. Login Screen

On clicking the register button, register screen will be shown as in figure 4.We have 3 textboxes; name, email id and password. On clicking register, all his information will be stored in the database.

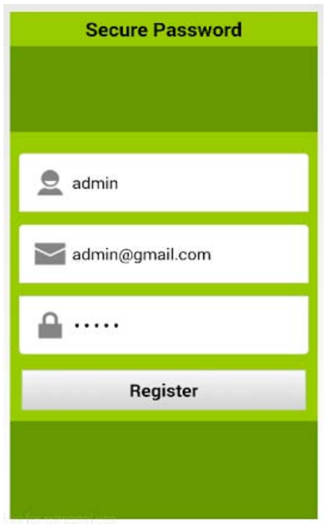


Figure 4.Registration

After creating an account and login, User will be navigated to next screen (Figure 5).Here we have 3 buttons that are having the name of 3 algorithms on them, respectively. User can select any of the algorithms to encrypt his contacts. We also have a graph button here which will show in time taken (in nanoseconds) to generate the hash. After using all 3 algorithms, user can compare the time taken by 3 algorithms. The 'Logout' Button will help user to log the application out.

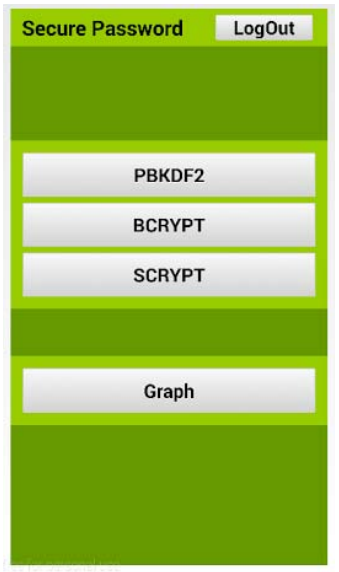
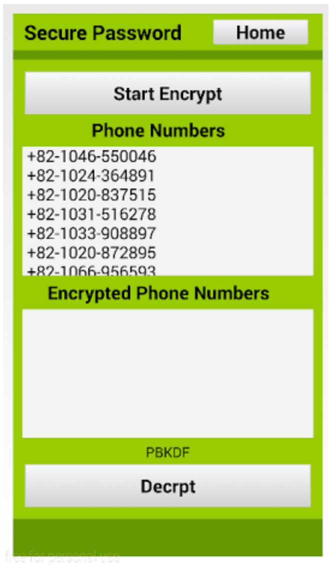


Figure 5 Home Screen

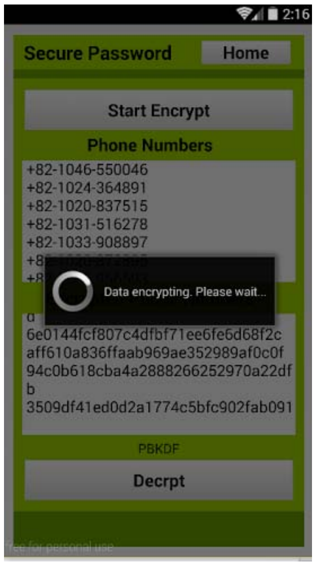
On choosing any one of them, Figure 6 is the next screen that will pop up on the android device.



```
+82-1046-550046
+82-1024-364891
+82-1020-837515
+82-1031-516278
+82-1033-908897
+82-1020-872895
+82-1066-956593
```

Figure 6 Encryption

Here, it is showing all the contact numbers that are stored in the android device. On tapping 'Start Encrypt' button, the contact numbers will start encrypting. Processing will be shown on screen (figure 7).



```
+82-1046-550046
+82-1024-364891
+82-1020-837515
+82-1031-516278
+82-1033-908897
```

Data encrypting. Please wait...

```
d
6e0144fcf807c4dfbf71ee6fe6d68f2c
aff610a836ffaab969ae352989af0c0f
94c0b618cba4a2888266252970a22df
b
3509df41ed0d2a1774c5bfc902fab091
```

Figure 7 Encryption Process

After encrypting, we can also decrypt the contact numbers but it will need the same password. Here, authenticity of the user can be verified. Only the intended user, who has encrypted the contacts, can decrypt them. See Figure 8.

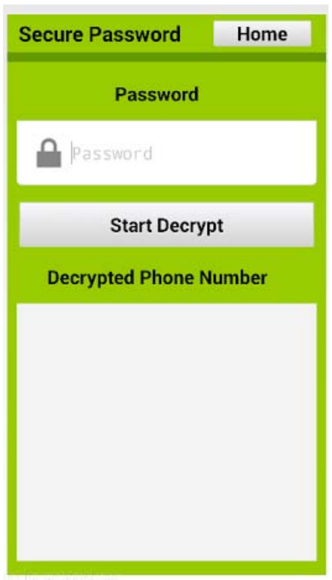


Figure 8 Decryption

We can click 'Home' button to go back to the home screen where 3 algorithms are listed (Figure 5). We can try encrypting the contact numbers using all the three algorithms one by one. After we are done doing that, there is graph button on home screen (Figure 5) to check the performance of each algorithm.

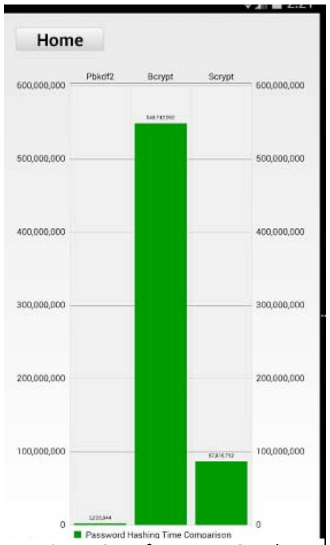


Figure 9 performance Graph

On taping the graph button, Figure 9 will appear on screen.

Let us see how the performance of the algorithms varies if we change some important and even parameters in these algorithms. Next section is about performance analysis.

V. TESTS AND RESULTS

For testing purpose, we experimented by varying the even parameters: salt value, key length and Number of iterations. We will take these three parameters as basis and will compare the output on the basis of these parameters in terms of time taken to generate the hash of the password.

Table 3. Constant parameters

Parameters	PBKDF2	Bcrypt	Scrypt
Salt(bytes)	16	16	16
Iteration count(int)	1000	1000	1000

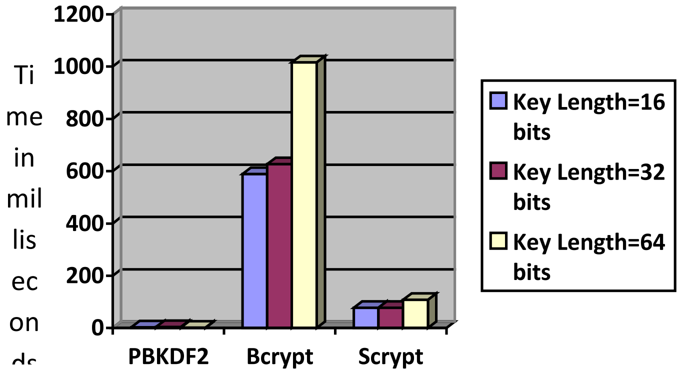


Figure 10. Varying Key Length

Figure 10 shows the performance when we have kept Salt (Bytes) and Iteration Count (integer) as constant and we are varying the Key Length. Constant parameters are shown in Table 3. In the graph, the first 3 bars are of PBKDF2 with different key lengths (16 bits, 32 bits and 64 bits).

Table 4. Constant Parameters

Parameters	PBKDF2	Bcrypt	Scrypt
Salt(bytes)	16	16	16
Key Length(bytes)	128	128	128

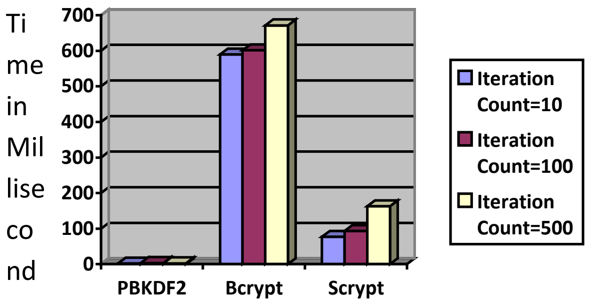


Figure 11: varying iteration counts

The 3 bars of each algorithm is showing the time taken in milliseconds with Salt (bytes) and Key Length (bytes) as constant(as shown in table 4) with different variation count(10,100 and 500).

Table 5 constant parameters

Parameters	PBKDF2	Bcrypt	Scrypt
Salt(bytes)	16	16	16
Key Length(bytes)	128	128	128

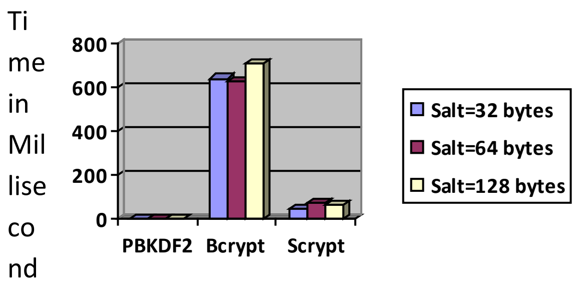


Figure 12. Varying Salt value

In the last graph, we have kept Salt (bytes) and Key Length as constant as in table 5.3 and we are changing salt to 3 cases (32 bytes, 64 bytes and128 bytes).

VI. CONCLUSION AND FUTURE IMPLEMENTATION

Security is the major issue in today's paperless world. There are lots of applications that are running on PBKDF2 and are working very successfully. They are considered The Best Password Managers for 2016[13]. In our application, we have presented the performance analysis of the three algorithms. These algorithms are already implemented on desktop platform. So, we have chosen the mobile platform to measure the performance of these algorithms. We have concluded that PBKDF2 is fast and can be considered the best among 3 algorithms. But PBKDF2 is cracked as reported in the news [14]. Bcrypt is slow because it is using Blowfish which is using iterations as 2 to the power number of iterations parameters. Bcrypt and Scrypt are memory hard functions. They take large resources and computation power to crack. Hence, they are nearly impossible to crack.

There is utmost need to have strong password as well as manage them well. So, these algorithms provide solution to the problem of password generation and management. With all performance tests, we are clear that which algorithm is providing what type of security. In future, we are trying to modify it as multimedia securing application. Also, we will try to implement it on platforms other than android like iOS.

VII. REFERENCES

1. S. K. Pal, D. Bhardwaj, R. Kumar and V. Bhatia, "A New Cryptographic Hash Function based on Latin Squares and Non-linear Transformations," *Advance Computing Conference, 2009. IACC 2009. IEEE International*, Patiala, 2009, pp. 862-867
2. Turan, M. S., E. B. Barker, W. E. Burr, and L. Chen (December 2010). "Recommendation for Password-based Key Derivation".*http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf.*
3. Antonopoulos, A. M. (3 December 2014). "Mastering Bitcoin: Unlocking Digital Cryptocurrencies". *O'Reilly Media*, 221-223.
4. Kak, Avi. "Lecture 24: The Dictionary Attack and the Rainbow-Table ..." Purdue University, (27 Apr. 2015). Web. *https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture24.pdf.*
5. Assurance Technologies. (2015). passlib.hash.bcrypt - BCrypt. *http://pythonhosted.org/passlib/lib/passlib.hash.bcrypt.html*.
6. Bansal, S. K. (April 10, 2014). "Securing Passwords with Bcrypt Hashing Function". *Retrived from: http://thehackernews.com/2014/04/securing-passwords-with-bcrypt-hashing.html*.
7. Bard, A. (July 11, 2013). "3 Wrong Ways to Store a Password And 5 code samples doing it right". *https://adambard.com/blog/3-wrong-ways-to-store-a-password/*.
8. Provos, N., Mazières, D., (June 1999)."A Future-Adaptable Password Scheme". *USENIX '99, Freenix Track. Monterey, CA. http://www.usenix.org/events/usenix99/provos.htm*
9. Bezzi, M., & al., e. (2011). "Data privacy". *In Camenisch, Jan et al. Privacy and Identity Management for Life. Springer*, 185–186.
10. C. Percival, S. J. (2012-09-17). "The scrypt Password-Based Key Derivation Function". *IETF*.
11. CoinPursuit . (2014). "SHA-256 and Scrypt Mining Algorithms". *Retrived from: CoinPursuit*.
12. *Goldberg, J. (June 6, 2012). A salt-free diet is bad for your security. Retrieved from: https://blog.agilebits.com/2012/06/06/a-salt-free-Diet-is-bad-for-your-security/.*
13. *"The Best Password Managers for 2016."* PCMAG. *Web. 18 Mar. 2016.*
14. Goldberg, Jeffrey. "Crackers Report Great News for 1Password4." 10 Mar. 2014. *https://blog.agilebits.com/2014/03/10/crackers-report-great-news-for-1password-4/.*