Computing, Information Systems, Development Informatics & Allied Research Journal
Vol. 8 No. 1, March, 2017 - www.cisdijournal.net

Empirical Evaluation of Data Hashing Algorithms for Password Checks in PHP Webapps Using Salt and Pepper

Ajayi, O.O.
[1]Department of Computer Science
Faculty of Science,
Adekunle Ajasin University
Akungba-Akoko, Ondo State, Nigeria
olusola.ajayi@aaua.edu.ng

Falana, T.F.
[2]Department of Science and Technical Education
Faculty of Education,
Adekunle Ajasin University
Akungba-Akoko, Ondo State, Nigeria
fredrick.falana@aaua.edu.ng

ABSTRACT

Authentication is key in any network setting. It is an act of verifying and disallowing the penetration of false or fake data. The quest to ensure adequate authentication gave rise to the design and deployment of different hashing algorithms. Researches have however shown the different hash algorithms having different loopholes. This study furthers existing works in this domain by examining the existing hash algorithms. The experimental work considers the password check, testing with SHA256, SALT, BCRYPT, CRYPT, and MD5 and injecting Pepper code to the algorithms. Just as it is in the domestic world, the result shows that adding Pepper to Salt proves a more secured algorithm for preserving and protecting password.

Keywords: WebApps, Hashing, Algorithm, Security, Cryptography, Salt, MD5, SHA, Crypt, BCrypt, Pepper.

CISDI Journal Reference Format
Aziken, G.O. and Egbokhare, F.A. (2017): Resolving the User-Developer Requirements Elicitation Conflict Using a Psychological Lens Computing, Information Systems, Development Informatics & Allied Research Journal. Vol 8 No 1. Pp 21-28.
Available online at www.cisdijournal.net

1. RESEARCH BACKGROUND

Web Application introduced in 1990, was a general, delivery mechanism. It is transform from a for static hypertext documents to a complete dynamic run-time environment for multi-party and distributed applications. The emerging trend was popular in peer-to-peer web applications and multiple applications. But the transformation of the web application from the server-centric model creates a significant and numerous challenges in web applications security (Alanazi & Sarrab 2011). MD4 was the first widely used dedicated hash function. Developed by Rivest in 1990, it started encountering attacks after which Rivest created MD5, a stronger function in 1992. However, Den Boer, Bosselaers and Dobbertin reported semi free start collision and pseudo collision attack on MD5. By the mid-1990s, BSD extended DES-BASED crypt to properly support long passwords and permit configurable iteration counts of up to 725 iterations, along with 24-bitsalting (Peslyak, 2012).

In 1994, FreeBSD introduced the MD5-based crypt library, which enables the use of long passwords and 1000 iterations of MD5 with up to 48-bit salting. Most Linux distributions adopted the FreeBSD MD5 library by the late 1990s. Early rainbow table attack tools, such as Qcrack and BitSlice, were released around 1997, thereby allowing fast pre-computations of DES hashes while supporting salting designed to attack a dictionary with 4096 possible salts. As computers and users became connected and networked in the late 1990s, password attacks extended to networks. During this period, non-switched Ethernet was a commonly used technology. Passwords were secured at rest but usually transmitted across networks during operation (Peslyak, 2012). In the early 1990s, attackers targeted passwords in transit by sniffing compromised servers, thereby intercepting passwords transmitted by all the users in a local segment. Developers adapted to the networked computing movement and implemented challenge response pairs, Kerberos, S/Key, and SSH to provide network authentication without exposing protected keys or passwords. With the expansion of the World Wide Web, web application authentication became widely adopted. This innovation also extended the allowable password length and highlighted the need to secure passwords. In the late 1990s and early 2000s, numerous web developers built application password security around the PHP MD5 module. However, the adopted PHP MD5 hashing function lacks password salting and reiteration (Peslyak, 2012).

In 1999, Niels Provos and David Mazieres proposed ways of building systems in which password security keeps up with hardware speeds. They formalized the properties desirable in a good password system and show that the computational cost of any secure password scheme must increase as hardware improves. They presented two algorithms with adaptable eksblowfish, a block cipher with a purposefully expensive key schedule and BCrypt, a related hash function. The key setup begins with a modified form of the standard Blowfish key setup, in which both the salt and password are used to set all sub-keys. There are number of rounds in which the standard Blowfish keying algorithm is applied, using alternately the salt and the password as the key, each round starting with the sub-keys state from the previous round. Crypto-theoretically, this is no stronger than the standard Blowfish key schedule, but the number of rekeying rounds is configurable; this process can therefore be made arbitrarily slow, which helps deter brute-force attacks upon the hash or salt. Failing a major breakthrough in complexity theory these algorithms should allow password based systems to adapt to hardware improvements and remain secure well into the future.

According to Sriramya and Karthika (2015), BCrypt is currently the secure standard for password hashing. It's derived from the Blowfish block cipher which, to generate the hash, uses look up tables which are initiated in memory. This means a certain amount of memory space needs to be used before a hash can be generated. This can be done on CPU, but when using the power of GPU it will become a lot more cumbersome due to memory restrictions. BCrypt has been around for 14 years, based on a cipher which has been around for over 20 years. It's been well vetted and tested and hence considered the standard for password hashing. When BCrypt was originally developed its main threat was custom ASICs specifically built to attack hash functions. These days those, ASICs would be GPUs (password brute forcing can actually still run on GPU, but not in full parallelism) which are cheap to purchase and are ideal for multithreaded processes such as password brute forcing. FPGAs are similar to GPUs but the memory management is very different. On these chips brute forcing BCrypt can be done more efficiently than on GPUs, but if you have a long enough passwords, it will still be unfeasible. The iteration count is a power of two, which is an input to the algorithm. In 2009 Colin Percival introduced SCrypt which is an improvement upon BCrypt. As noted, the main threat against BCrypt in 1999 was ASICs with low gate counts, but today the threat is FPGAs (Field Programmable Gate Arrays) and BCrypt was not designed to protect against that threat. In the above cases, a function is being provided that developers can put passwords into to get an encoded result. The solutions are improvements upon Morris and Thompson's work for protecting passwords in UNIX systems.

Thulasimani et al (2012) proposed Hash functions that are the most widespread among all cryptographic primitives, and are currently used in multiple cryptographic schemes and in security protocols. The basic design of SHA- 192 is to have the output length of 192. The SHA-192 has been designed to satisfy the different level of enhanced security and to resist the advanced SHA attacks. The security analysis of the SHA-192 is compared to the old one given by NIST and gives more security. The SHA-192 can be used in many applications such as public key cryptosystem, digital sign encryption, message authentication code, random generator and in security architecture of upcoming wireless devices like software defined radio etc.

At the Rump Session of CRYPTO 2006, Christian Rechberger and Christophe De Cannière claimed to have discovered a collision attack on SHA-1 that would allow an attacker to select at least parts of the message. As of 2012, the most efficient attack against SHA-1 is considered to be the one by Marc Stevens with an estimated cost of $2.77M to break a single hash value by renting CPU power from cloud servers. After some improvements in SHA-1, NSA designed a set of hash functions named SHA-2. SHA-2 was first published in 2001 by NIST. And in August 2002, it became the new Secure Hash Algorithm. Among other benefits, it provides better security than SHA-1 because it has bigger message digest size. Due to this, it is harder to break. The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 (Chaitya et al, 2014).

In May 2014 NIST announced the SHA-3 as the newest proposed scheme for hash functions. SHA-3 is a subset of the cryptographic primitive family Keccak and a cryptographic hash function designed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche, building upon RadioGatún. SHA-3 is not meant to replace SHA-2, as no significant attack on SHA-2 has been demonstrated. Because of the successful attacks on MD5 and SHA-0 and theoretical attacks on SHA-1 and SHA-2, NIST perceived a need for an alternative, dissimilar cryptographic hash, which became SHA-3 (Chaitya et al, 2014).

2. RELATED WORKS

Shweta et al (2013) shows how MD5 is used for authenticating any file and how it is used for hashing any string using some helper functions. In Richa et al (2013), the authors proposed a new hash function algorithm that includes a 64bits key as an ingredient to the function. It produced 128bits digest with a secure and simple technique as compared to many other existing techniques. The author submitted that, the use of key adds the source integration facility while creating digest just for integrity purpose. Amar et al (2013) evaluated five (5) selected hash functions (Blake, Grostel, JH, Keccak, and Skein) meant to be deployed ad the SHA-3 algorithm. The analysis breaks down the power consumption and clock frequency of each SHA-3 selected algorithms using a mixture of different synthesis options. The results show that the Keccack algorithm as the strongest algorithm when examining the trade-off between power and speed.

In their work, Chaitya et al (2014) made a review to the birth and strength of SHA-1. The paper also made allusion to SHA-2, its variants and the proposed SHA-3. Priyanka et al (2014) carried out a comparative analysis of SHA-0, SHA-1, and SHA-2, reporting SHA-2 as the most secured.

In Disha (2015), the paper justified MD algorithm as one that enhances security of data by generating digital signature. He proposed an algorithm that has five (5) phases: key generation, digital signing, encryption, decryption, and signature. The evaluation of the result shows the algorithm would be a high security algorithm for data transfer. The paper, Sahni (2015) discusses common cryptographic hashing algorithms and compare their relative performance. Majoring on MD5 and SHA-1, the evaluation result shows MD5 produced a hash value of 128bits, whereas SHA-1 produced 160bits. While submitting that SHA-1 proved more secured than MD5, he however pointed out that in terms of ease of implementation, MD5 is more suitable. In Sriramya et al (2015), the study focuses on providing security to user's data using salted password hashing technique. To ensure this and enforce a tight security procedure, BCrypt algorithm was implemented to secure user's privacy when shopping online. Vijay et al (2015) analysed the major algorithms that relate to cryptography (AES, SHA and MD5) and implementing in CUDA. The authors also compare the performance between GPU implementation and CPU implementation.

Study by Marc et al (2016) further exposes the vulnerabilities of SHA-1 and how GPUs can be used very efficiently in actualizing the attack. In Borkar et al (2016), the study surveyed existing hash algorithms parallel to SHA-1 algorithm with encryption and LSB substitution technique. The proposed SHA-1 proved capable of achieving a superior working frequency and also higher throughput. Pandey (2016) analyses the security risks of the hashing algorithm MD5 in password storage and discusses different solutions such as Salts and Iterative Hash. The experimentation carried out showed that MD5 can be made safer when it is included by salt key.

3. RESEARCH METHODOLOGY

3.1 Research Problem

The new millennium ushers in a new web application frameworks, with the birth of Web 2.0, which allows user to do more than just information retrieving. It was indeed a time of new era that promotes creation of web–aware applications with high interactivity and that are user-centric (Ben-Natan, 2005). Other numerous benefits include the growth and usage of electronic commerce, web mails, internet messaging, internet banking, international share trading, forums, web communities to mention just but few. However, to every beautiful fruit of the tree, there is always an evil behind. This development is not without a challenge. As the growth has witnesses many accolades, likewise there were wide reports of threats and security breaches; websites are often attacked directly as they stand as the face of business (Gavin et al, 2005). This work closely follows the work of Pandey (2016) curbing vulnerability in password checks. Due to these challenges, there is a need to evaluate the different types of hashing algorithm broadly in use as it relate to securing password in PHP web applications.

3.2 Research Goal

The goal of this study is to carry out an empirical evaluation of data hashing algorithms for password checks in PHP Web Applications taking SHA1, MD5, BCrypt, SCrypt, Salt and SHA256 into consideration and implementing a comparative analysis of the specified algorithms to prove their vulnerabilities and strengths.

The research methodology takes the following phases:

Phase 1: Review of existing work

Under this phase, extensive literature covering issues on existing data hashing algorithms methodology were reviewed to ascertain the level of work done by other researchers.

Phase 2: Design, Development and Evaluation

Action 1: Design

Here, the intended model was designed and presented.

Action 2: Development

At this point, a GUI-based login application was developed, taking into consideration the different hashing algorithms of WebApps.

Action 3: Evaluation

The result of our development will help us to evaluate the different hashing algorithms specified in this study.

Phase 3: Improvement

The new way of improving data with the highly secured hashing algorithm will also be discuss.

4. RESEARCH DESIGN MODEL

Figure 1 shows the proposed model for the empirical evaluation of data hashing algorithms for password checks in PHP Web Applications.

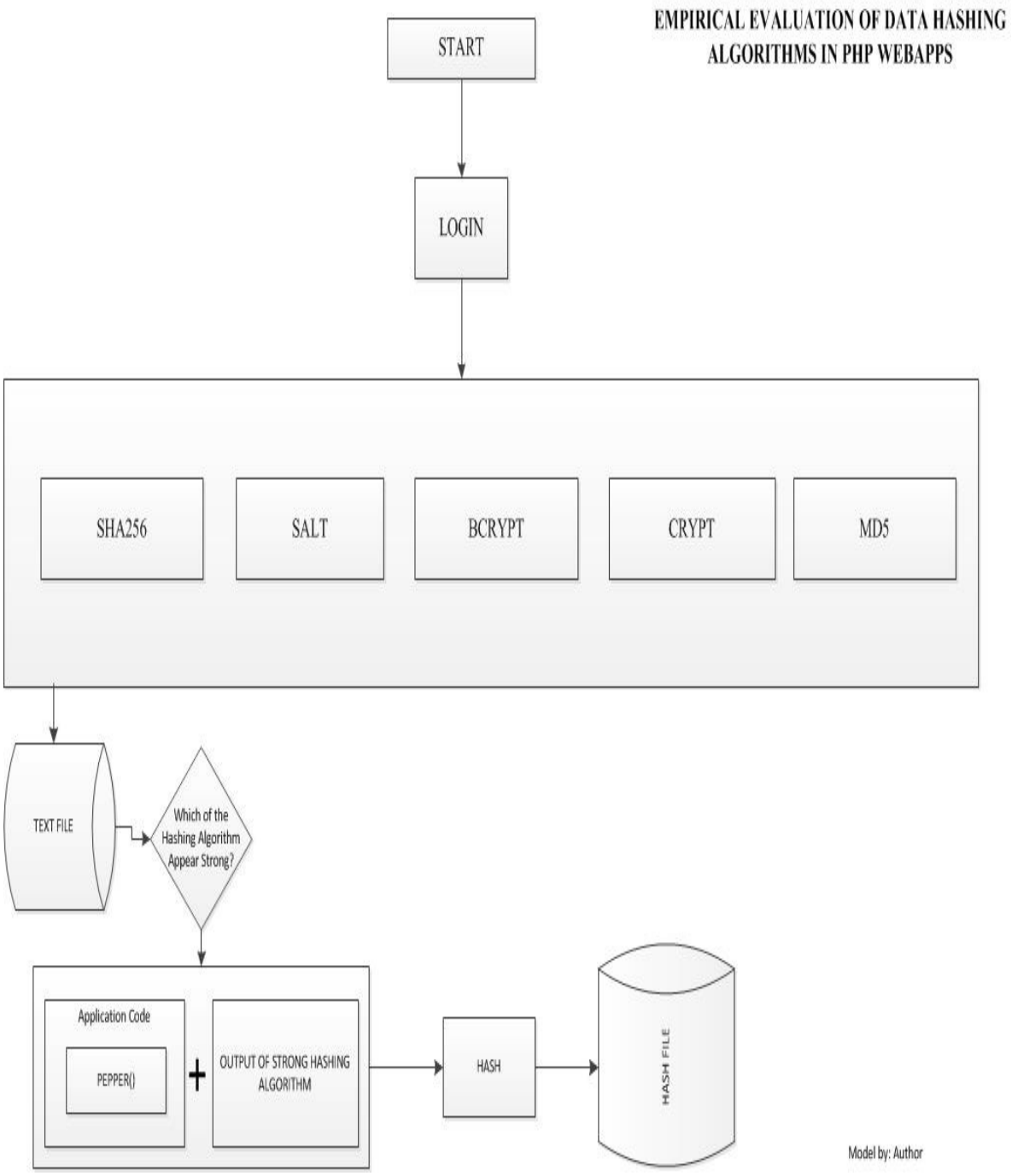

Figure 1: Architectural Design for Data Hashing in PHP WebApps

The above model work as follow

i. Start
ii. Log in to the system
iii. Test the password with
    a. SHA256
    b. SALT
    c. BCRYPT
    d. CRYPT
    e. MD5
iv. Check the strength and weakness using the bit strength of each algorithm
v. Add a pepper to the highest strength algorithm
vi. Hash
vii. Store the hash in a database

5. IMPLEMENTATION

The proper method for secure password storage hashing is salt and crypt hashing algorithm in combination with Sha512 has been tested by the model above.

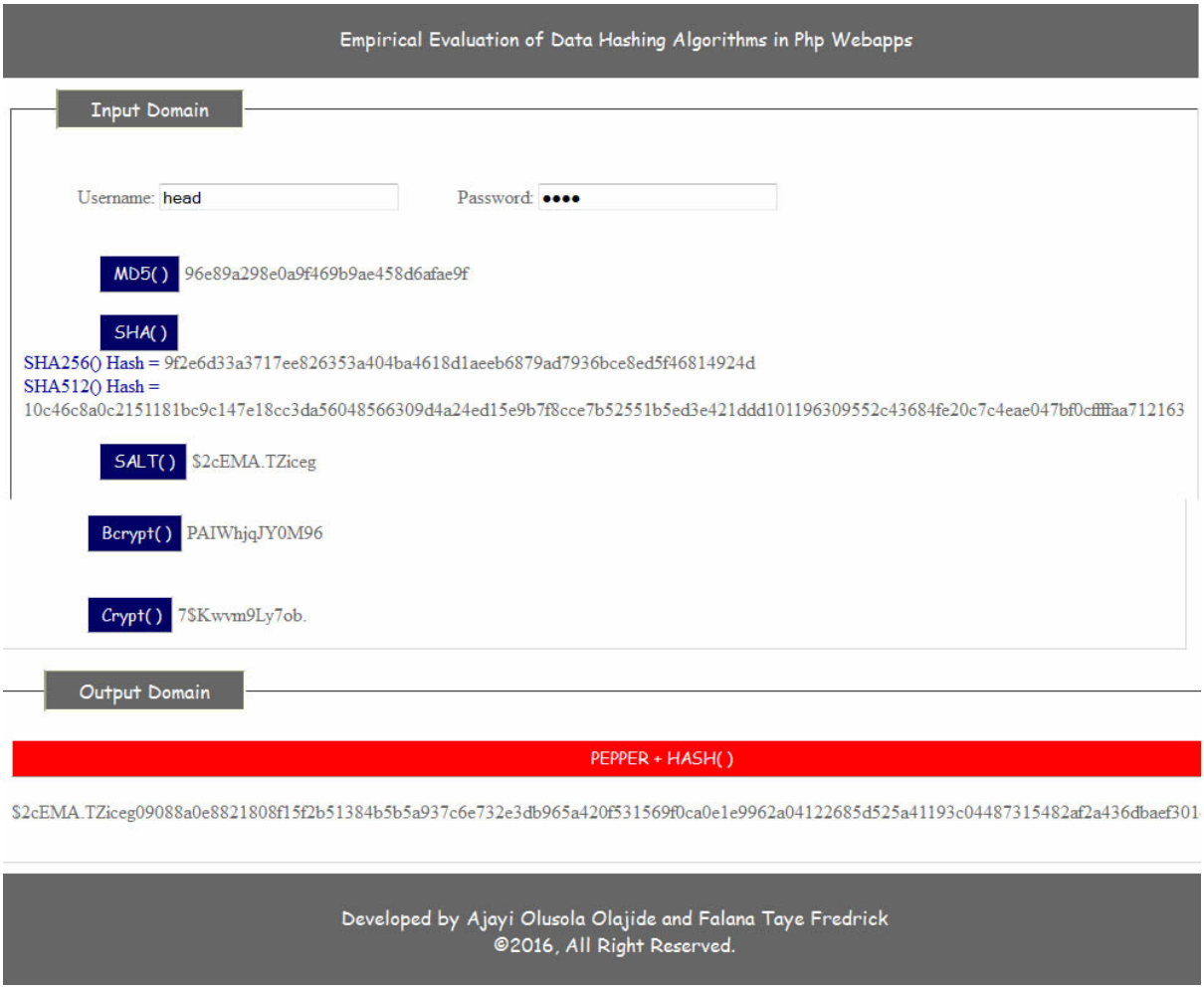

Figure 2: Implementation Interface

The recommended/approved method for storing password verifiers is to store using the line as stated in this pepper pseudocode:

$verifier = $salt + hash($salt + $password)

where:

hash( ) is a cryptographic hashing algorithm, $salt is a random, evenly distributed, high entropy value and $password is the password entered by the user.

The latest introduction to the concept of secure hashing algorithm is the addition of secret key into the mix (sometimes called pepper). Where the pepper is a secret, high entropy, system-specific constant. The rationale seems to be that even if the attacker gets hold of the password verifiers, there is a good chance he or she does not know the pepper value. So mounting a successful attack becomes harder.

We need to store passwords securely in the database, and come up with something on the form of: $hashed_password = hash($salt .$password) where $salt is stored in plaintext in the database, together with the $hashed_password representation and randomly chosen for each new or changed password and $pepper is a constant stored in plaintext in the application code (or configuration if the code is used on multiple servers or the source is public).

$hashed_password = hash($pepper .$salt.$password)

Adding this pepper is easy. We are just creating a constant in our code, entering a strategy, for example rehash the password on the next login and store a version number of the password hashing strategy alongside the hash.

```
<?php
 $a = $_POST['textfield']; // name
$b = $_POST['textfield2']; // password
if(isset($a)&&isset($b))
    {
     $rounds = 5000;
     $salt = mt_rand(); // random salt
     $salt_chars = array_merge ( range ('A','Z'), range('a','z')); // secured random salt merge
     for($i=0; $i < 22; $i++)
    {
       $salt .= $salt_chars[ array_rand ($salt_chars)];
      }
   $aj = crypt($b, sprintf('$2a$ 02d$', $rounds) . $salt); // salt result
 $verifier = $aj.hash('sha512', $aj.$b ); //Add pepper with salt, sha512 and crypt
}
?>
```

Using the pepper does add to the strength of the password, even if the database is being compromise, this does not imply compromise of the application. Without knowledge of the pepper the passwords remain completely secure. Because of the password been salted with specific pepper, attacker can't find out if two passwords in the database are the same or not. The reason is that hash ($pepper . $salt .$password) effectively build a pseudo random function with $pepper as key and $salt. $password as input (for same hash candidates like PBDKF2 with SHA*, BCrypt or SCrypt). Two of the guarantees of a pseudo random function are that we cannot deduce the input from the output under a secret key and neither the output from the input without the knowledge of the key.

This sounds a lot like the one-way property of hash functions, but the difference lies in the fact that with low entropy values like passwords we can effectively enumerate all possible values and compute the images under the public hash function and thus find the value whose image matches the pre-image. With a pseudo random function, we cannot do so without the key (i.e. without the pepper) as we cannot even compute the image of a single value without the key.

6. CONCLUSION

This research work helps to evaluate the different hashing algorithms. MD5, SHA1, SHA256 appear so weak, MD5 allows attackers to create multiple, differing input sources that, when this algorithm is used, result in the same output fingerprint. On the other hand, SHA1-2 levels are vulnerable to length-extension and partial-message collision attacks. The Salt and Crypt hashing algorithm appear strong, they protect against rainbow table attacks, and also helps to deter brute-force attacks upon the hash.

The use and introduction of pepper help to increase security level and help to secure data, this mean even if the database is being compromise, this does not imply compromise of the application. Without knowledge of the pepper the passwords remain completely secure. On the other hand, even if the application source codes are being exposed, attackers cannot deduce the primary data. Because the password been salted with specific pepper, attacker can't find out if two passwords in the database are the same or not.

7. FUTURE RESEARCH DIRECTION

The future trends in WebApps security of data should be gear towards easy on-click of pepper hashing algorithm, a MySQL–PHP inbuilt approach in which a pepper and user data will be the parameters. This will make securing data easy to use for developers.

REFERENCES

1. Alanazi, F., and Sarrab, M. (2011). The History of Web Application Security Risks. International Journal of Computer Science and Information Security. Vol. 9, Issue 6, pgs. 40-47.
2. Amar, J., and Christopher, J. M. (2013). Detail Power Analysis of the SHA-3 Hashing Algorithm Candidates on Xilinx Spartan-3E. International Journal of Computer and Electrical Engineering, Vol. 5, No. 4, pgs. 410-413.
3. Bertoni, G., Daemen, J., Peeters, M. and Assche, G. V. (2011). The Keccak Reference, Submission to NIST (Round 3), [online]. http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html.
4. Biham, E. and Chen, R. (2004). Near-Collision of SHA-0, Proceedings of the Advances in Cryptology – CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19.
5. Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., and Jalby, W. (2005). Collisions of SHA-0 and Reduced SHA-1, Proceedings of the Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26.
6. Boer, B. D., and Bosselaers, A. (1993). Collisions for the compression function of MD5, Proceedings of the Advances in Cryptology - EUROCRYPT '93, Workshop on Theory and Applications of Cryptographic Techniques, Lofthus, Norway, May 23-27.
7. Borkar, V., and Khobragade, A. S. (2016). Implementation of Secure Hash Algorithm-1 using FPGA. International Journal of Advanced Research in Computer and Communication Engineering. Vol. 5, Issue 6, pgs. 793-795.
8. Chaltya, B. S., and Drashti, R. P. (2014). Secured Hash Algorithm-1: Review Paper. International Journal for Advanced Research in Engineering and Technology. Vol. 2, Issue X, pgs. 26-30
9. Disha, S. (2015). Digital Security using Cryptographic Message Digest Algorithm. International Journal of Academic Research in Computer Science and Management Studies. Vol. 3, Issue 10, pgs. 215-219.
10. National Institute of Standards and Technology. (2012). Recommendation for Applications Using Approved Hash Algorithms http://csrc.nist.gov/publications/nistpubs/800-107-rev1/sp800-107-rev1.pdf
11. National Institute of Standards and Technology, (2012). FIPS PUB 180-4, Secure Hash Standard (SHS), US Department of Commerce, Washington D. C., March.
12. Pandey, T. (2016). A Secure Data Transmission over the Cloud Computing: Using Salted MD5. International Journal of Innovative Research in Computer and Communication Engineering. Vol. 4, Issue 2, pgs. 2784- 2790.
13. Peslyak, S. D. (2012). The History of Password Security. http://www.throwingfire.com/the-history-of-password-security/
14. Priyanka, V., and Blumika, L. (2014). Review Paper on Secure Hashing Algorithm and Its Variants. International Journal of Science and Research. Vol. 3, Issue 6, pgs. 629-632.
15. Provos, N., and Mazieres, D. (1999). A Future-Adaptable Password Scheme. Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference, Monterey, California.

16. Richa, P., Upenda, M., and Abhay, B. (2013). Design and Analysis of a New Hash Algorithm with Key Integration. International Journal of Computer Applications. Vol. 81, No. 1, pgs. 33-38.
17. Sahni, N. (2015). A Review on Cryptographic Hashing Algorithms for Message Authentication. International Journal of Computer Applications. Vol. 120, No. 16, pgs. 29-32
18. Shweta, M., Shikha, M., and Nilesh, K. (2013). Hashing Algorithm: MD5. International Journal for Scientific Research and Development. Vol. 1, Issue 9, pgs. 1931-1933.
19. Sriramya, P., and Karthika, R. A. (2015). Providing Password Security by Salted Password Hashing using BCrypt Algorithm. ARPN Journal of Engineering and Applied Sciences. Vol. 10, N0. 13, pgs. 5551-5556.
20. Thulasimani, L., and Madheswaran, M. (2012). A Novel Secure Hash Algorithm for Public Key Digital Signature Schemes. The International Arab Journal of Information Technology. Vol. 9, No. 3, pgs. 262-267.
21. Vijay, J. B., Bangal, D. B., and Dhatrak, K. B. (2015). Parallelization of Encryption and Hashing Algorithm using GPU. International Research Journal of Engineering and Technology. Vol. 2, Issue 6, pgs. 642-647.