Efficient Secured Hash Based Password Authentication in Multiple Websites

[1]T.S.Thangavel
AP/ Dept. of M.C.A,
K.S.Rangasamy College of Technology,
Tiruchengode 637215,Tamilnadu,India.

[2]A. Krishnan
Dean,
K.S.Rangasamy College of Technology,
Tiruchengode 637215,Tamilnadu,India.

Abstract

The most commercial web sites rely on a relatively weak form of password authentication, the browser simply sends a user's plaintext password to a remote web server, often using secure socket layer. Even when used over an encrypted connection, this form of password authentication is vulnerable to attack. In common password attacks, hackers exploit the fact that web users often use the same password at many different sites. This allows hackers to break into a low security site that simply stores username/passwords in the clear and use the retrieved passwords at a high security site. This work developed an improved secure hash function, whose security is directly related to the syndrome decoding problem from the theory of error-correcting codes. The proposal design and develop a user interface, and implementation of a browser extension, password hash, that strengthens web password authentication. Providing customized passwords, can reduce the threat of password attacks with no server changes and little or no change to the user experience. The proposed techniques are designed to transparently provide novice users with the benefits of password practices that are otherwise only feasible for security experts. Experimentation are done with Internet Explorer and Fire fox implementations.

***Keywords**: password authentication, password hash, multiple-website password, pseudo random number, hash value, hash function, Password phishing.*

I. INTRODUCTION

A random password generator is software program or hardware device that takes input from a random or pseudo-random number generator and automatically generates a password. Random passwords can be generated manually, using simple sources of randomness such as dice or coins, or they can be generated using a computer. While there are many examples of "random" password generator programs available on the Internet, generating randomness can be tricky and many programs do not generate random characters in a way that ensures strong security. A common recommendation is to use open source security tools where possible, since they allow independent checks on the quality of the methods used. Note that simply generating a password at random does not ensure the password is a strong password, because it is possible, although highly unlikely, to generate an easily guessed or cracked password.

A password generator can be part of a password manager. When a password policy enforces complex rules, it can be easier to use a password generator based on that set of rules than to manually create passwords. In situations where the attacker can obtain an encrypted version of the password, such testing can be performed rapidly enough so that a few million trial passwords can be checked in a matter of seconds. The function rand presents another problem. All pseudo-random number generators have an internal memory or state**.** The size of that state determines the maximum number of different values it can produce; an n-bit state can produce at most 2^n different values. On many systems rand has a 31 or 32 bit state, which is already a significant security limitation.

The main cryptographic hash function design in use today iterates a so called compression function according to Merle's [10] and Damgard's[11] constructions. Classical compression functions are very fast but, in general, cannot be proven secure. However, provable security may be achieved with compression functions designed following public key principles, at the cost of being less efficient. This has been done for instance by Damgard, where he designed a hash function based on the Knapsack problem. Accordingly, this function has been broken by Granboulan and Joux,[9] using lattice reduction algorithms. The present paper contributes to the hash function family by designing functions based on the syndrome decoding problem, which is immune to lattice reduction based attacks.

Unlike most other public key cryptosystems, the encryption function of the McEliece cryptosystem is nearly as fast as a symmetric cipher. Using this function with a random matrix instead of the usual parity check matrix of a Goppa code, a provably secure one-way function has been constructed since there is no trapdoor, its security can be readily related to the difficulty of syndrome decoding.

The purpose of this paper is to improve updated parameters for the hash function. Our paper analyzes asymptotical behavior of their attack. We shall establish that this attack is exponential, such that the design for the hash function is sound.

II. LITERATURE REVIEW

Computer applications may require random numbers in many contexts. Random numbers can be used to simulate natural or artificial phenomena in computer simulations, many algorithms that require randomness have been developed that outperform deterministic algorithms for the same problem, and random numbers can be used to generate or verify passwords for cryptography-based computer security systems. The present invention relates to the use of random numbers in such security systems, called as cryptographic applications. Specifically, the present invention pertains to generating a random number in a secure manner for such cryptographic applications. In the context of cryptographic applications [1], there may be an hostile trespasser or agent, who desires to infiltrate the security of cryptographic security system in order to gain access to sensitive, confidential, or valuable information contained therein. For example, banks often encrypt their transactions and accounts.

In order to ensure the utmost security [5], it is essential that the security system implements a method for generating a random number that appears completely random. In this manner, a completely random password or cryptographic key presents no opening or prior knowledge that can be exploited by an hostile agent. [2].

Many prior art methods exist for generating random numbers. These prior art methods typically involve the use of some type of chaotic system. A chaotic system is one with a state that changes over time in a largely unpredictable manner. To use the chaotic system [4] to generate a random number, there is some means of converting the state of the system into a sequence of bits (i.e., a binary number). In the past, chaotic systems were based on various sources, such as the sound of radio static, the output of a noisy diode, output of a Geiger counter, or even the motion of clouds. These chaotic systems can be converted to produce binary numbers by using standard techniques.

For instance, a pseudo-random binary string can be generated from the digital recording of static noise via a digital microphone. Alternatively, a noisy diode can be sampled at a suitable frequency and converted into a digital signal, or a picture of an area of the sky can be taken and subsequently scanned and digitized. These resulting binary strings that are generated over time are generally random in nature. However, there are several problems associated with simply using a chaotic system as a source of random numbers.[3] First, chaotic systems can be completely or partially predicted over small amounts of time. For example, the position of clouds in some area of the sky at some time can be used to achieve reasonably accurate predictions of the position of clouds in the same area a short time into the future.

Furthermore, the behavior of chaotic systems [6] can be far from completely random. For instance, a digitized picture of a cloud formation will not look like a picture of random information, but instead, will look like a cloud formation. Moreover, chaotic systems may be biased by outside sources which may be predictable. As an example, a radio signal can be affected by a strong external signal, or the behavior of a noisy diode can be changed by the surrounding temperature. All of the above problems arise because the behavior of a chaotic system may not be completely random [8]. More specifically, an adversary observing or wishing to affect the random number source can take advantage of certain localities that may be inherent in chaotic systems. These localities can occur either in space or time.

Finally, a number of existing applications including Mozilla Fire fox provide convenient password management by storing the user’s web passwords on disk, encrypted under some master password. When the user tries to log in to a site, the application asks for the master password and then releases the user’s password for that site. Thus, the user need only remember the master password. The main drawback compared to PwdHash is that the user can only use the web on the machine that stores his passwords. On the plus side, password management systems do provide stronger protection against dictionary attacks when the user chooses a unique, high entropy password for each site. However, many users may fail to do this.

III. SECURED HASH BASED PSEUDO RANDOM PASSWORD SCHEME

The proposed methodology of the secure hash password system contains one-way hash functions that can process a message to produce a condensed representation called a message digest. This algorithm enables the determination of a message’s integrity, any change to the message will, with a very high probability, results in a different message digest. This property is useful in the generation and verification of digital signatures and message authentication codes, and in the generation of random numbers. The algorithm is described in two stages, preprocessing and hash computation. Preprocessing involves padding a message, parsing the padded message into m-bit blocks, and setting initialization values to be used in the hash computation. The hash computation generates a message schedule from the padded message and uses that schedule, along with functions, constants, and word operations to iteratively generate a series of hash values.

The final hash value generated by the hash computation is used to determine the message digest. The design principle of hash functions is iterating a compression function (here denoted *F*), which takes as input *s* bits and returns *r* bits (with $s > r$). The resulting function is then chained to operate on strings of arbitrary length (Fig 1). The validity of such a design has been established and its security is proven not worse than the security of the compression function.

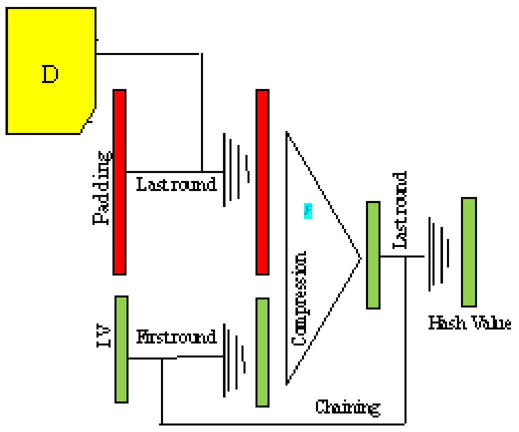


Fig 1: Iterative hash function structure

Compression Hash function Algorithm

Input: s bits of data.

1. Split the S input bits in w pars $S_1 \ldots \ldots S_w$ of $\log_2 \left(\frac{n}{w} \right)$ bits.

2. Convert each S_i to an integer between 1 and $\frac{n}{w}$.

3. Choose the corresponding column in each H_i;

4. Add the w chosen columns to obtain a binary string of length r.

Output: r bits of hash.

The core of the compression function is a random binary matrix *H* of size $r \times n$. The parameters for the hash function are *n* the number of columns of *H*, *r* the number of rows of *H* and the size in bits of the function output, and *w* the number of columns of *H* added at each round. Random password generators normally output a string of symbols of specified length. These can be individual characters from some character set, syllables designed to form pronounceable passwords, or words from some word list to form a pass phrase. The program can be customized to ensure the resulting password complies with the local password policy, say by always producing a mix of letters, numbers and special characters. The strength of a random password can be calculated by computing the information entropy of the random process that produced it. If each symbol in the password is produced independently, the entropy is just given by the formula

$$H = L \log_2 N = L \frac{\log N}{\log 2}$$

where N is the number of possible symbols and L is the number of symbols in the password. The function $\log_2$ is the base-2 logarithm. H is measured in bits.

Symbol Set	N	Entropy / Symbol
Digits only (0-9) (e.g. PIN)	10	3.32 bits
Single case letters (a-z)	26	4.7 bits
Single case letters and digits (a-z, 0-9)	36	5.17 bits
Mixed case letters and digits (a-z, A-Z,0-9)	62	5.95 bits
All standard U.S. Keyboard characters	94	6.55 bits
Dice ware word list	7776	12.9 bits

Thus an eight character password of single case letters and digits would have 41 bits of entropy (8 x 5.17). The same length password selected at random from all U.S. computer keyboard characters would have 52 bit entropy; however such a password would be harder to memorize and might be difficult to enter on non-U.S. keyboards. A ten character password of single case letters and digits would have essentially the same strength (51.7 bits). Any password generator is limited by the state space of the pseudo-random number generator, if one is used. Thus a password generated using a 32-bit generator has maximum entropy of 32 bits, regardless of the number of characters the password contains.

IV. SYSTEM MODEL

The system model concerned with attacks on the extension that originate on malicious phishing sites. Password hashing is computed using a Pseudo Random Function (PRF) as follows:

Hash (pwd, dom) = PRFpwd (dom)

Where the user's password pwd is used as the PRF key and the remote site's domain name dom or some variant is used as the input to the PRF. The hash value is then encoded as a string that satisfies the site's password encoding rules, under control of a configuration file used by the browser extension.

Password hashing is implemented naively inside a browser with rudimentary knowledge of HTML form components. Forms begin with a tag <form action=URL> that tells the browser where the form is to be submitted, and HTML password fields are tagged using <input type="password">. The naive browser extension listens for blur events, which fire when focus leaves a field. When the blur event occurs, the extension replaces the contents of the field with the hashed value, using the form action attribute. Thus, after the user enters a password into a form, the clear text password is replaced by a hashed version.

The goal, however, is to defend against web scripting attacks with minimal change to the user experience. For this leverage the browser extension as a protective but largely transparent intermediary between the user and the web application. All input can be first monitored and secured by the browser extension before the web application is aware that the user is interacting with it. This requires a mechanism by which users can notify password hash browser extension that they are about to enter a password. Password hash can then take steps to protect the password as it is being entered. A distributed hash table is introduced to handle the browser utility replicas of the multiple users across hash authentication mode.

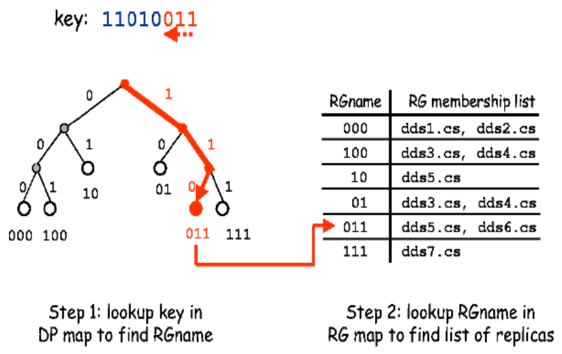
A. Distribute Hash Table

In Figure2, The distributed hash table provides incremental scalability of throughput and data capacity as more nodes are added to the cluster. To achieve this, we horizontally partition tables to spread operations and data across bricks. Each brick thus stores some number of partitions of each table in the system, and when new nodes are added to the cluster, this partitioning is altered so that data is spread onto the new node. Because of our workload assumptions, this horizontal partitioning evenly spreads both load and data across the cluster.

Given that the data in the hash table is spread across multiple nodes, if any of those nodes fail, then a portion of the hash table will become unavailable. For this reason, each partition in the hash table is replicated on more than one cluster node. The set of replicas for a partition form a replica group; all replicas in the group are kept strictly coherent with each other. Any replica can be used to service a `get()`, but all replicas must be updated during a `put()` or `remove()`. If a node fails, the data from its partitions is available on the surviving members of the partitions' replica groups. Replica group membership is thus dynamic; when a node fails, all of its replicas are removed from their replica groups. When a node joins the cluster, it may be added to the replica groups of some partitions.

The illustration below describe the steps taken to discover the set of replica groups which serve as the backing store for a specific hash table key. The key is used to traverse the DP map tries and retrieve the name of the key's replica group. The replica group name is then used looked up in the RG map to find the group's current membership.

We do have a checkpoint mechanism in our distributed hash table that allows us to force the on-disk image of all partitions to be consistent, the disk images can then be backed up for disaster recovery. This checkpoint mechanism is extremely heavy weight, however; during the check pointing of a hash table, no state-changing operations are allowed. We currently rely on system administrators to decide when to initiate checkpoints.


key: 11010011
0
1
0
1
0
1
0
1
0
1
10
01
000
100
011
111
Step 1: lookup key in DP map to find RGname
Step 2: lookup RGname in RG map to find list of replicas

RGname	RG membership list
000	dds1.cs, dds2.cs
100	dds3.cs, dds4.cs
10	dds5.cs
01	dds3.cs, dds4.cs
011	dds5.cs, dds6.cs
111	dds7.cs

Figure 2: distribute hash table

V. EXPERIMENTAL RESULT AND DISCUSSION

A. Experimental Implementation

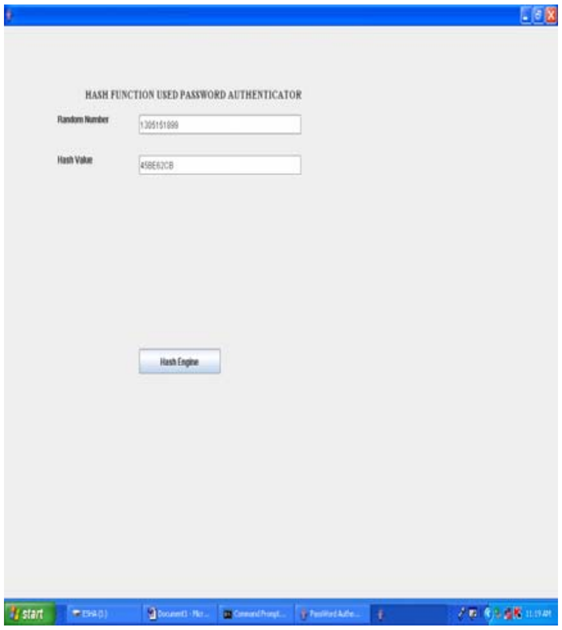
In the proposed hash based password authentication, a user can change her password at a given site without changing her password at other sites. In fact, the recommended method for using password hash is to choose a small number of strong, distinct passwords, one for every security level (e.g. one password for all financial sites, one password for all news sites, etc). The password hash extension ensures that a break-in at one financial site will not expose the user's password at all other banks.

The system implemented the prototype as a browser helper object for Internet Explorer. The extension registers three new objects i.e., an entry in the Tools menu (to access extension options), an optional new toolbar, and the password protection service itself. Internet Explorer support COM event sinks that enable Browser Helper Objects to react to website events. Use these sinks to detect focus entering and leaving password fields, drag and drop events, paste events and double click events. The DHTML event model used by Internet Explorer allows page elements to react to these events before they "bubble" up to the extension at the top level. Since extension must handle keystroke events before scripts on the page, we intercept keystrokes using a low-level Windows keyboard hook.

When the password-key or password-prefix is detected, the browser extension determines whether the active element is a password field. If it is not a password field, the user is warned that it is not safe to enter his password. If it is a password field, the extension until the focus leaves the field. The keystrokes are canceled and replaced with simulated keystrokes corresponding to the "mask" characters. The system implementation of secured hash password authentication is intercepts all keystrokes of printable characters accomplished through following process. The client utility is in a web browser, generating hash password as shown in Fig. 3.

B. Result and Discussion

The proposed hash based multi-site pseudo random password mechanism shows proposal considers N number of times that the user U might authenticate before re-registration is required. This suggests that high values of N are desirable. The host H has to store R hash function values at the server. This implies that to reduce the storage requirements, it is desirable to have a low value of R. However, N/2R is the average number of hash function computations that U has to do for every authentication session. Thus, it is desirable to have a high value of R. The parameter R therefore represents a tradeoff between computational requirements of the user U and the storage requirements of the host H. This implies that the value of N and R are best selected by the system administrator keeping in mind the system requirements. We believe that given the current state of storage technologies, the storage requirement is significantly less important than the computational requirement. Major improvement over the previous cryptographic method is the significant reduction in computational requirements per authentication session and increase in the number of logins before re-initialization.


HASH FUNCTION USED PASSWORD AUTHENTICATOR

Random Number: 1305151899

Hash Value: 45BE62CB

Fig 3: Client side hash password generation

Table 1: Effectiveness of Proposed Hash Based Pseudorandom Password Authentication over Existing Cryptographic Password Authenticity

Technique	Resistance to eaves dropping	Web Browser Compatibility	Web browser Compatibility Number of rounds for authentication	Computational Efficiency	Storage Capacity	Communication effectiveness
Existing Cryptographic password authentication	Feasible	False	Low	Low	High	False
Proposed hash based pseudo random password authentication	Highly feasible	True	High	High	Low	True

Regarding the computation evaluation the host verifies the proposed hash password sent by user by computing just a single hash function and one comparison with the stored last one time password. For the investigation of communication factor the host sends the user a hash value and an integer t. The user returns only a single hash value. The resultant of the proposed hash based pseudo random password

Authentication and cryptographic password authentication are listed in the below Table1.

VI.CONCLUSION

The paper proposed a provably secure hash functions based password authentication scheme. This construction provides features such as both the block size of the hash function and the output size are completely scalable. The password hashing method is extremely simple, rather than send the user's clear text password to a remote site, it sends a hash value derived from the user's password, and the site domain name. Password Hash captures all user input to a password field and sends hash (pwd, dom) to the remote site. The hash is implemented using a Pseudo Random Function keyed by the password. Since the hash output is tailored to meet server password requirements, the resulting hashed password is handled normally at the server; no server modifications are required. This technique deters password phishing since the password received at a phishing site is not useful at any other domain. The proposed model implements the password hashing as a secure and transparent extension to modern browsers.

REFERENCES

[1] Kumar Magnitude and Rajendra Katti, "A Hash-based Strong Password Authentication Protocol with User Anonymity", International Journal of Network Security, Vol.2, No.3, PP.205–209, May 2006.

[2] J. A. Halderman, B.Waters, and E. Felten "A convenient method for securely managing passwords" To appear in Proceedings of the 14th International World Wide Web Conference (WWW 2005), 2005.

[3] F. Hao, P. Zielinski, "A 2-round anonymous veto protocol," Proceedings of the 14th International Workshop on Security Protocols, SPW'06, Cambridge, UK, May 2006.

[4] Muxiang Zhang, "Analysis of the SPEKE password-authenticated key exchange protocol," IEEE Communications Letters, Vol. 8, No. 1, pp. 63-65, January 2004.

[5] Z. Zhao, Z. Dong, Y. Wang, "Security analysis of a password-based authentication protocol proposed to IEEE 1363," Theoretical Computer Science, Vol. 352, No. 1, pp. 280–287, 2006.

[6] Abdalla M., Catalano D., Chevalier C., and Point cheval D., "Efficient Two-Party Password Based Key Exchange Protocol in the UC Framework", Springer-verlag Berlin, pp 335-351, 2008

[7] Feng Hap and Peter Reyan "Password Authenticated key exchange by juggling" IEEEP1363-2008.

[8] O. Hallaraker and G. Vigna. Detecting Malicious JavaScript Code in Mozilla. In Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), pages 85–94, Shanghai, China, June 2005.

[9] Antoine Joux. Multicollisions in iterated hash functions. Application to cascaded construction. In Advances in Cryptology - CRYPTO '04 Proceedings, Lecture Notes in Computer Science, Vol. 3152, M. Franklin, ed, Springer-Verlag, 2004, pp. 306-316.

[10] R.C. Merkle. A Certified Digital Signature. In Advances in Cryptology - CRYPTO '89 Proceedings, Lecture Notes in Computer Science Vol. 435, G. Brassard, ed, Springer-Verlag, 1989, pp. 218-238.

[11] I. Damgård. A Design Principle for Hash Functions. In Advances in Cryptology - CRYPTO '89 Proceedings, Lecture Notes in Computer Science Vol. 435, G. Brassard, ed, Springer-Verlag, 1989, pp. 416-427.

[1]**T. S. Thangavel** received his Bsc degree in Computer Science (Bharathiyar University) in 1991 and the MSC degree in Computer science Bharathidasan University) in 1993 and the Mphil degree in Computer Science (Bharathidasan University) in 2003. He is pursuing the PhD degree in department of science and humanities (Anna University). He is working as an Assistant Professor in MCA department at K.S.Rangasamy College of Technology, Tiruchengode.

[2]**Dr. A. Krishnan** received his Ph.D degree in Electrical Engineering from IIT, Kanpur. He has been in the field of technical teaching and research for more than four decades at Government College of Technology and Coimbatore Institute of Technology, Tamilnadu, India. From 1994 to 1997, he was an Associate Professor in Electrical Engineering at University Pertanian Malaysia (UPM), Malaysia. He is now working as an Academic Dean at K.S.Rangasamy College of Technology, Tiruchengode and research guide at Anna University Chennai. His research interest includes Control system, Digital Filters, Power Electronics, Digital Signal processing, Communication Networks. He has been published more than 250 technical papers at various National/ International Conference and journals. Dr. Krishnan is a senior member of IEEE, Life fellow Institution of Engineers (India), IETE (India) and Computer Society of India.