Provable Secured Hash Password Authentication

T.S.Thangavel
AP / Dept. of MCA
K.S.Rangasamy College of Technology
Tiruchengode, Tamil Nadu

A. Krishnan
Dean
K.S.Rangasamy College of Technology
Tiruchengode, Tamil Nadu

ABSTRACT
The techniques such as secured socket layer (SSL) with client-side certificates are well known in the security research community, 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 SSL. 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. While password authentication could be abandoned in favor of hardware tokens or client certificates, both options are difficult to adopt because of the cost and inconvenience of hardware tokens and the overhead of managing client certificates.

Recently, some collisions have been exposed for a variety of cryptographic hash functions including some of the most widely used today. Many other hash functions using similar constructions can however still be considered secure. Nevertheless, this has drawn attention on the need for new hash function designs. 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 and report the result of initial user.

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 with 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 cryptographic hash makes it difficult to compute hash(pwd,dom2) from hash(pwd,dom1) for any domain dom2 distinct from dom1. For the same reason, passwords gathered by breaking into a low security site are not useful at any other site. The hash attack is always exponential in terms of the length of the hash value. We also study the work-factor of this attack, along with other attacks from coding theory, for non asymptotic range, i.e. for practical values. Accordingly, we propose a few sets of parameters giving a good security and either a faster hashing or a shorter description for the function.

Keywords: Password Authentication, Hash Functions, Message Digest, Secure Socket Layer, Random Password Generator, Pseudo Random Function.

1. 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.

Some computer operating systems provide much stronger random number generators. One example, common on most Unix platforms, is /dev/random. The Java programming language includes a class called SecureRandom. Windows programmers can use the Cryptographic Application Programming Interface function CryptGenRandom. Another possibility, is to derive randomness by measuring some external phenomenon, such as timing user keyboard input. Using random bytes from any of these sources should prove adequate for most password generation needs.

The main cryptographic hash function design in use today iterates a so called compression function according to Merkle's and Damgard's 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, 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. For instance, there is no polynomial time algorithm to decode a random code, thus there is no polynomial time algorithm to invert the compression function and/or find a collision. However, for the practical parameters which have been proposed, there is an efficient attack with a cost as low as 243 (or 262 depending on the set of parameters), as demonstrated by Coron and Joux , using Wagner's method for the generalized birthday problem.

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.

2. 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, 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.[1].

One common method for circumventing a cryptographic application is to guess at potential passwords or cryptographic key, which are then submitted on a trial basis. This process is repeated until, by happenstance, the valid password is chanced upon. Fortunately, this process is extremely time consuming and inefficient. Also, preventative action can be taken to render this type of attack highly ineffective. However, if the random number generator used in generating valid passwords is somehow flawed in any way, the hostile agent can potentially take advantage of this flaw to circumvent the security of the system. For instance, a security system based on English text passwords, are susceptible to a dictionary attack. Thus, in order to ensure the utmost security, 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 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 [4].

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. 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.[3].

Furthermore, the behavior of chaotic systems 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. 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.

Referring back to the cloud example, knowing some of the picture can help one predict the rest of the picture, and knowing the state of the clouds at some time allows the hostile agent to reasonably guess the future state of the clouds. These flaws in chaotic systems make them potentially harmful choices for use in random number generators for cryptographic security systems in generating passwords. This is because an hostile agent can make use of the local properties of the random number generator by simply observing or affecting the system, to determine the generated passwords. Likewise, an hostile agent can take advantage of the local properties to substantially reduce the number of possibilities, thereby allowing the possibilities to be exhaustively tested much more quickly and effectively.

A further disadvantage of using a chaotic system as a source of randomness is that transforming the state of the system into a random number is a much slower process than typical computation done on a computer. Repeatedly generating random numbers from such a system can become a time bottleneck. The time bottleneck can be avoided in most computer applications by using pseudo-random numbers instead of random numbers [5]. A pseudo-random number generator deterministically generates a sequence of numbers by some computational process from an initial number, called a seed. The goal of the computational process is to generate a sequence of numbers from the seed that appear to be random. In

other words, an outside observer cannot predict the next number to be generated from the list of numbers previously generated without expending a great deal of computational effort. Thus, to generate a long sequence of pseudo-random numbers, one need only generate a much shorter random number to use as the seed for the pseudo-random number generator.

Password hashing with a salt is an old idea. However, web password hashing is often implemented incorrectly by giving the remote site the freedom to choose the salt. For example, HTTP1.1 Digest Authentication defines password hashing as follows i.e., digest = Hash (pwd, realm, nonce, username,,,) where realm and nonce are specified by the remote web site. Hence, using an online attack, a phisher could send to the user the realm and nonce the phisher received from the victim site. The user's response provides the phisher with a valid password digest for the victim site. Password hashing implemented in Kerberos 5 has a similar vulnerability. The first systems we are aware of that provide proper web password hashing are the Lucent Personal Web Assistant [2] and a system from DEC SRC [1]. To facilitate deployment, LPWA was implemented as a web proxy, which worked fine back when LPWA was implemented. However, many password pages these days are sent over SSL, and consequently a web proxy cannot see or modify the traffic. It was necessary to build PwdHash as a browser extension so that we could alter passwords before SSL encryption. Although it might be feasible to build a proxy that forges SSL certificates on the fly (essentially mounting a man in the middle attack on SSL), such a proxy would not be able to identify or protect passwords that are typed into mock password fields.

The DEC SRC system was implemented as a standalone Java Applet and did not take into account the various challenges in implementing password has inside a modern browser. The Password Maker extension for Mozilla Firefox is functionally similar to password hash, but with a slightly more prominent user interface. Users can indicate that they would like to insert a hashed password by pushing a toolbar button or selecting an option from the password field's context menu. The password is then entered into a dialog box and stored so that it can be filled in automatically in the future. Password Maker may be a good solution for users who do not mind the security risks of storing their password in the browser, but it demands significant changes in the password entry model that people have used for years, and thus maintains a steep learning curve.

The Password Composer extension for Mozilla Firefox modifies password fields on the current page, allowing the user to enter a hashed password into a new password field that is superimposed over the old one. Password Composer is also provided as a book marklet and as a JavaScript file that can be loaded for each page using the Grease Monkey Firefox extension. A malicious script could read the pre-hashed password as it is typed into the superimposed password field, however. The Password Composer user interface also seems vulnerable to spoofing. Halderman et al. [2] study how to secure password hashing from dictionary attacks by using ultra-slow hash functions. As discussed earlier, these techniques can be integrated into PwdHash to help defend against dictionary attacks. We note that our focus here is very different from that of [2]. Primarily the system concerned with how to implement password hashing inside a modern browser so that phishing sites cannot steal cleartext passwords, with minimal change to user experience.

Finally, a number of existing applications including Mozilla Firefox 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.

3. METHODOLOGY

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 passphrase. 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) (eg. PIN)	10	3.32 bits
Single case letters (a-z)	26	407 bits
Single case letters and digits (a-z, 0-9)	36	5.17bits
Mixed case letters and digits (a-z, A-Z,0-9)	62	5.95bits
All standard US keyboard characters	94	6.55 bits
Dicsware 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 essentailly 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.

3.1 Secure Hashing

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.

Compression Hash function Algorithm

Input : s bits of data.

1. Split the S input bits in w parts $S_1 \ldots\ldots S_w$ 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.

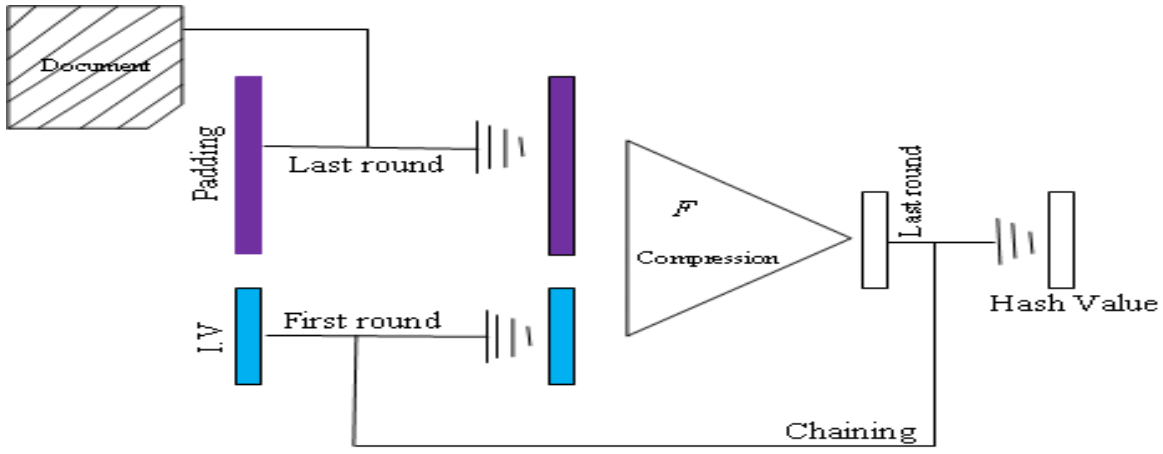

Figure 1: Iterative Hash Function Structure

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 has can then take steps to protect the password as it is being entered. There are two closely related methods i.e., password-prefix and the second password-key. In addition the system model contains distributed hash table to handle the browser utility replicas of the multiple users across hash authentication mode.

4.1 Password Prefix

Password-prefix is an elegantly unobtrusive mechanism to defend against the JavaScript attacks. Users are asked to prefix their passwords with a short, publicly known sequence of printable characters. Password hash monitors the entire key stream and takes protective action when it detects the password-

prefix sequence. The password-prefix must be short but unlikely to appear frequently in normal text input fields. A common prefix shared among all users of the extension allows the extension to be portable without requiring any changes of settings. For internationalization, the password prefix should not be an English word at all, but something that could be easily remembered and typed.

The proposed extension has two modes i.e., normal mode and password mode. The extension monitors all keyboard events. In normal mode, it passes all keyboard events to the page as is. When the password-prefix is detected in the key stream, the extension switches to password mode and does the following i.e., it internally records all subsequent key presses, and it replaces the user's keystrokes with a fixed sequence and passes the resulting events to the browser. The first keystroke following the password-prefix is replaced with "A," the second with "B," and so on. This translation continues until focus leaves the password field, at which point the extension reverts back to normal mode. In other words, all keystrokes entered following the password-prefix are hidden from the browser and from scripts running inside the browser until focus leaves the field.

Hence, JavaScript key loggers cannot steal the clear text password. Hashing can take place at one of two times. The first option is to replace the contents of the field with the hashed password when focus leaves the field. The second option is to trap the form submission event and then replace the contents of all password fields with the appropriate hashed passwords. The first option is more jarring to the user, because his password could potentially change length immediately after entering it (once it gets hashed). However, it allows the extension to work automatically at sites like yahoo.com that implement their own password hashing algorithm using JavaScript on their login pages.

Finally, if the password-prefix is ever detected while focus is not on a password field, our browser extension reminds the user not to enter a password. Thus, users are protected from mock password field attacks that confuse them into entering a password into an insecure location. This password-prefix approach blocks the JavaScript attacks and provides a number of additional benefits. Legitimate web pages often collect PIN's or social security numbers via password fields. Password Hash will not hash the data in such fields because this data does not contain the password prefix.

Password reset pages often ask users to enter both the old and the new password. New Password Hash users must visit these pages to "change" their old passwords to the new, hashed versions. The password entered in the "current password" field should not be hashed, while the password entered in the "new password" section should be hashed. The password prefix mechanism automatically provides the right functionality, assuming the old password does not contain the password-prefix. The password-prefix conveniently lets users decide which passwords they want to protect using hashing and which passwords they want left as is.

4.2 Password Key

Password-key is an alternative to the password prefix mechanism. Instead of using a printable sequence the idea is to use a dedicated keyboard key called a "password-key." Users are asked to press the password-key just before entering a password. Imagine that future keyboards might have a dedicated key marked "password," but for now use the 'F2' key, which is not currently used by Internet Explorer, Firefox, or Opera.

The semantics of the password-key inside our extension are very similar to the password-prefix. When the user presses the password-key the extension enters password mode as described previously. All subsequent keystrokes are hidden from the browser and scripts running within the browser. The extension returns to normal mode when focus leaves the field. If the password-key is pressed while focus is not in a password field, the user is warned not to enter a password. The password-key, however, is less prone to mistake, whereas the password-prefix could appear naturally in the key stream and trigger undesired protection, password-key protection can only be initiated in response to decisive action by the user.

With respect to user experience, however, a password-key seems inferior to a password-prefix. First, novice users need to know to press the password-key when entering their password, but not to press the key when entering a PIN. While the prefix mechanism also demands a special attention to passwords, it may be easier to teach users that "all secure passwords begin with (@@)" than asking them to remember to press a certain key before entering a password. Second, upon resetting their password at a password reset page just after installing PwdHash users need to know to press the password-key for their new password, but not to press the key for their old password. Password-prefix is the preferable method of triggering password protection.

4.3 Key Stream Monitor

Key stream monitor is a web password hashing implementation that detects unsafe user behavior. This defense would consist of a recording component and a monitor component. The recording component records all passwords that the user types while the extension is in password mode and stores a one-way hash of these passwords on disk. The monitor component monitors the entire keyboard key stream for a consecutive sequence of keystrokes that matches one of the user's passwords. If such a sequence is keyed while the extension is not in password mode, the user is alerted.

4.4 Distribute Hash Table

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

Fig2 gives 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.

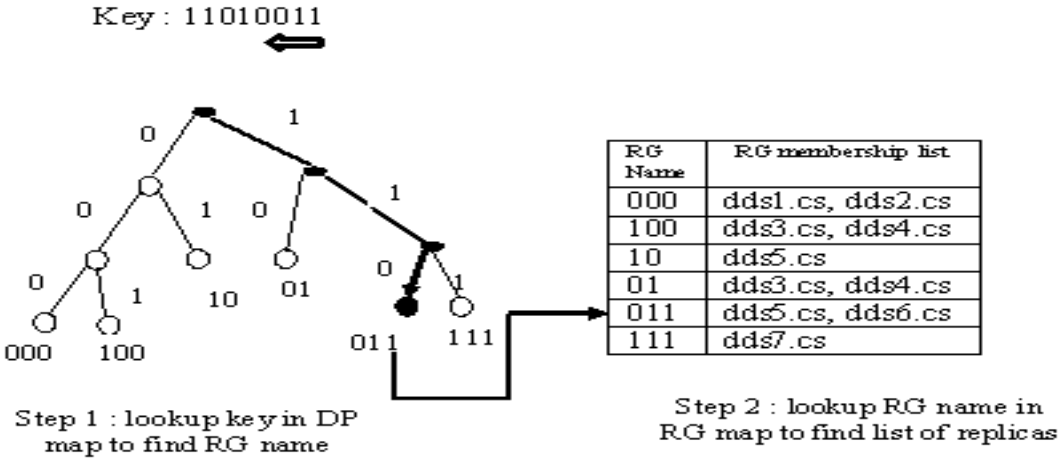

Figure 2: Distribute Hash Table

5. EXPERIMENTAL IMPLEMENTATION

Using Password Hash, 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 intercepts all keystrokes of printable characters until the focus leaves the field. The keystrokes are canceled and replaced with simulated keystrokes corresponding to the "mask" characters. The first mask character is "A," then "B," and so on. The extension maintains a translation table for each of these password fields, mapping mask characters back to the original keystrokes. This method allows the user to backspace and delete characters at arbitrary positions within the password field without confusing the extension.

For the Internet Explorer version of the extension, leave the masked characters in the field until the user submits the form, then we intercept the submission event with a BeforeNavigate2 handler. Internet Explorer does not allow extensions to edit the form data in BeforeNavigate2 directly. Rather, cancel the original Navigate2 event and fire a new, modified one. The extension includes a data structure to detect which Navigate2 events were fired by the extension, and which ones were fired as a result of user action, so that it does not attempt to translate the form data more than once and get stuck in a loop.

5.1 Secured Hash Implementation

The system implementation of secured hash password authentication is accomplished through following process.

Client

The client consists of service-specific software running on a client machine that communicates across the wide area with one of many service instances running in the cluster. The mechanism by which the client selects a service instance is beyond the scope of this work, but it typically involves DNS round robin, a service-specific protocol, or level 4 or level 7 load-balancing switches on the edge of the cluster. An example of a client is a web browser, in which case the service would be a web server. Note that clients are completely unaware of DDS's: no part of the DDS system runs on a client.

Service

The service is a set of cooperating software processes, each of which we call a service instance. Service instances communicate with wide-area clients and perform some application-level function. Services may have soft state (state which may be lost and recomputed if necessary), but they rely on the hash table to manage all persistent state.

Hash table API

The hash table API is the boundary between a service instance and its ``DDS library". The API provides services with `put()`, `get()`, `remove()`, `create()`, and `destroy()` operations on hash tables. Each operation is atomic, and all services see the same coherent image of all existing hash tables through this API. Hash table names are strings, hash table keys are 64 bit integers, and hash table values are opaque byte arrays; operations affect hash table values.

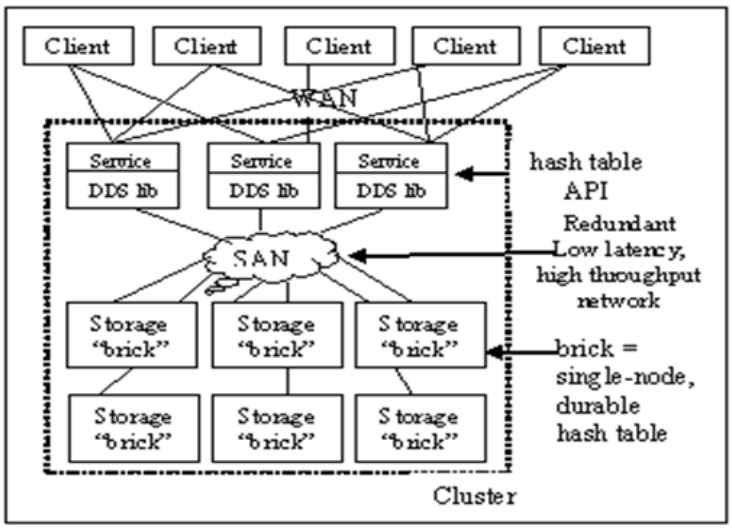

Figure 3: Distributed Hash Table Architecture

Each box in the diagram represents a software process. Each process runs on its own physical machine, however there is nothing preventing processes from sharing machines.

Distributed Data Structure (DDS) library

The DDS library is a Java class library that presents the hash table API to services. The library accepts hash table operations, and cooperates with the ``bricks" to realize those operations. The library contains only soft state, including metadata about the cluster's current configuration and the partitioning of data in the distributed hash tables across the ``bricks". The DDS library acts as the two-phase commit coordinator for state-changing operations on the distributed hash tables.

Brick

Bricks are the only system components that manage durable data. Each brick manages a set of network-accessible single node hash tables. A brick consists of a buffer cache, a lock manager, a persistent chained hash table implementation, and network stubs and skeletons for remote communication. Typically, we run one brick per CPU in the cluster, and thus a 4-way SMP will house 4 bricks. Bricks may run on dedicated nodes, or they may share nodes with other components

The browser extended secured password authentication tool generates the hashed password using either SHA1 or MD5 hashing algorithm depending on the choice you make. It will display the hashed password in the read only text box, it can also copy the hashed password to clipboard on your choice for easy paste operation.

6. 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 security depends directly of the output size and is truly exponential, it can hence be set to any desired level. The high output rates requires the use of a large matrix. On classical architectures this will only fix a maximum speed.

It is easy to introduce a trapdoor in a matrix by simply choosing one column to be the sum of some other columns of the matrix. This will then allow the person who generated the matrix to easily generate collisions. Concerning the hash size/collision security ratio, this construction does not allow to have the usual ratio of 2, obtained when using a classical paradox to find collisions. This can be changed by simply applying a final output transformation to the last hash, this transformation can further compress it to a size of twice the expected security against collision search.

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 cryptographic hash makes it difficult to compute hash (pwd, dom2) from hash (pwd, dom1) for any domain dom2 distinct from dom1. For the same reason, passwords gathered by breaking into a low security site are not useful at any other site, thus protecting financial institutions from sites with lacking security. The proposed model implements the password hashing as a secure and transparent extension to modern browsers.

7. REFERENCES

[1] N. Chou, R. Ledesma, Y. Teraguchi, and J. Mitchell, "Client-side defense against web based identity theft ", In Proceedings of Network and Distributed Systems Security (NDSS), 2004.

[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. Zieli´nski, "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]C.Ellison, C.Hall, R.Milbert, and B.Schneier, "Protecting secret keys with personal entropy" Journal of Future Generation Computer Systems", February 2000.

[7] P.Mackenzie, T.Shrimpton, and M.Jakobsson, "Threshold password-authenticated key exchange" In M.Yung, editor, CRYPTO 2002.

[8] Abdalla M., Catalano D., Chevalier C., and Pointcheval D., "Efficient Two-Party Password-Based Key Exchange Protocol in the UC Framework", Springer-Verlag Berlin, PP. 335 – 351, 2008.

Author Profile

T.S.Thangavel received the 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

Dr. A. Krishnan received his Ph.D degree in Electrical Engineering from IIT, Kanpur. 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 156 technical papers at various National/ International Conference and journals.