# The N/R One Time Password System

Vipul Goyal[1], Ajith Abraham[2], Sugata Sanyal[3] and Sang Yong Han[2]

[1]*OSP Global, Mumbai, India*
*vgoyal@ospglobal.com*

[2]*School of Computer Science and Engineering, Chung-Ang University, Korea*
*ajith.abraham@ieee.org, hansy@cau.ac.kr*

[3]*School of Technology & Computer Science, Tata Institute of Fundamental Research, India*
*sanyal@tifr.res.in*

## Abstract

*A new one time password system is described which is secure against eavesdropping and server database compromise at the same time. Traditionally, these properties have proven to be difficult to satisfy at the same time and only one previous scheme i.e. Lamport Hashes also called S/KEY one time password system has claimed to achieve that. Lamport hashes however have a limitation that they are computationally intensive for the client and the number of times a client may login before the system should be re-initialized is small. We address these limitations to come up with a new scheme called the N/R one time password system. The basic idea is have the server aid the client computation by inserting 'breakpoints' in the hash chains. Client computational requirements are dramatically reduced without any increase in the server computational requirements and the number of times a client may login before the system has to be re-initialized is also increased significantly. The system is particularly suited for mobile and constrained devices having limited computational power.*

## 1. Introduction

In the past decade, computer networks have grown at an explosive rate. In a wide range of environments, such networks have become a mission critical tool. Organizations are building networks with larger scales than ever before, and connectivity with the global internet has become indispensable. Along with this trend has come an explosion in the use of computer networks as a means of illicit access to computer systems. In the past, intruders have used flaws in network software to gain entry into remote computer systems. As more vendors and more sites fix the known flaws in their network software, many crackers are now looking for other weaknesses to exploit.

One particularly widespread attack is to capture and replay passwords commonly used to authenticate users. Since so many protocols send their passwords in clear text, anyone who can read network traffic can gain access to whatever is protected by clear text passwords. Crackers can use network management tools to sniff packets to discover clear text passwords, thereby gaining unauthorized access to systems using clear text reusable passwords.

One solution to this problem is to encode the password in such a way that an encoded password can only be used once and cannot be used to generate any other encoded password. Such an encoded password is called one-time password because it is usable exactly once. If an adversary captures such a password from a stream of data sent over a network, she cannot use it to gain access to the target system either by using it again (the first condition) or by performing any new coding on it (the second condition). In practice, the second condition is guaranteed by computational infeasibility rather than by impossibility – it would take an attacker an inordinately long time to discern any useful data from the intercepted one-time password.

Such an encoding was first devised by Lamport [1] and later popularized with the development of S/KEY at Bellcore [2]. S/KEY was further improved and re-implemented by United States Naval Research Laboratory to give rise to the OPIE (One Time Password in Everything) software distribution [13].

As we describe in Section 2, the above systems work on the principle of hash chain. However, all of

them have two common limitations i.e. (a) The client computational requirements are high making the system unsuitable for mobile devices with limited computational power and memory, and (b) The number of times the client may login before the system is required to be re-initialized is small. Note that re-initialization means that some manual intervention is required, e.g. the client should go personally to the server administrator to have the system re-initialized. There is no secure way of automatic re-initialization.

In this paper we device a novel construction of hash chains. The basic idea here is to repeatedly require the insertion of user password after a fixed distance in the hash chain. The links at which the insertion of the password is required may be made public and stored at the host (server). The host would then transfer one of those links to the user for aiding the computation of the user. This results in a significant decline in the computational requirement of the user. Since, this makes the construction of hash chain possible in manner such that increasing the length of the hash chain does not increase the user computational requirements; hash chains of very large length are feasible. Hence, the number of times a user may login before re-initialization is required is very high.

An important point to note here is that in all 'password based authentication systems', the client is assumed to be stateless and cannot be assumed to store anything. The only thing a user is required in order to be able to login is the password. Hence, the intermediate values of the hash chains cannot be stored by the client.

Rest of the paper is organized as follows- Section 2 gives the description of related one time password systems, i.e. Lamport Hashes. Note that S/KEY and other systems are conceptually the same as Lamport Hashes and differ only in the implementation details. This section also gives an idea of hash chains in general. Section 3 describes the proposed construction of the one time password system and various issues involved with it. Section 4 concludes the paper.

## 2. Related Research

The only password based symmetric key authentication protocol achieving resistance to password file compromise and eavesdropping is Lamport Hashes [1]. Later it was implemented by Phil Karn to give rise to the S/Key™ One Time Password system [2]. It was also standardized in RFC 1760 titled "The S/KEY One-Time Password System" [3] and in RFC 2289 titled "A One-Time Password System" [5] which preceded RFC 1938 [4]. A nice property of the system is that it avoids all kind of encryption mechanisms and uses only one way hash functions making it quite efficient. Further, S/KEY was improved by the United States Naval Research Laboratory to give rise to the OPIE (One Time Password in Everything) software distribution [13]. OPIE mainly differs with S/KEY in the implementation details, the basic idea remaining the same.

Lamport hashes are based on one way hash function (OWHF). One way functions are public functions that are easy to compute but computationally infeasible to invert, for suitable definitions of "easy" and "infeasible". If the output of a one-way function is of fixed length, it is called a one-way hash function (OWHF). More precisely, the definition of OWHF is given as:

**Definition:** A function h that maps bit strings, either of an arbitrary length or a predetermined length, to strings of a fixed length is a OWHF if it satisfies three additional properties:

- → Given x, it is easy to compute h(x)
- → Given h(x), it is hard to compute x
- → It is hard to find two values x and y such that h(x) = h(y), but $x \neq y$.

The system proceeds as follows:

Alice (the client) remembers a password. Bob (the server that will authenticate Alice) has a database where it stores, for each user:

- The username
- n, an integer which decrements each time Bob authenticates the user
- $h^n(p)$, i.e. h(h(…(h(p))…))

Where h is a one way hash function like MD5 and p is the user password. Note that none of the stored quantities is considered to be security sensitive. Hence the system is suitable for authentication in scenarios where the server (or the host) is either considered to be untrusted or is vulnerable to compromise.

For system initialization, Alice chooses a password p and n, the number of times she wants to authenticate to Bob. She then computes n iterations of the one way hash function over this password, i.e. $h^n(p)$. Alice then somehow securely sends n and $h^n(p)$ along with her username to Bob to initialize the system.

For authentication, Alice sends her username to Bob which in turn sends n. Then Alice computes $h^{n-1}(p)$ and sends the result to Bob as the next one time password (OTP). Bob calculates the hash of the received OTP and compares it with the stored $h^n(p)$. If they match, Bob overwrites $h^n(p)$ with the received $h^{n-1}(p)$ and

decrements n. Alice is now logged in. It is easy to see that the system is secure against both eavesdropping and server database compromise since the attacker cannot determine $h^{n-1}(p)$ from $h^n(p)$ (this follows from the non-invertibility of h). When n reaches 1, Alice should select a new password and should reinitialize the system as described before. There is no known secure way of automatic re-initialization and it should be done through manual or physical means. Thus, this is an inconvenience for the user.

### 2.1. Limitations of Lamport Hashes

Lamport Hashes use the principle of Hash chains. Hash chains have interesting public key cryptography like properties and have been widely used to replace / complement public key cryptography e.g. password based authentication [1], certificate revocation [6], micropayments [11], online auctions [10], secure logs [12], efficient multicasting [7-9] and server-supported signatures [14, 15].

Lamport hashes use hash chains attempting to replace public key cryptography in password based authentication. They, however, suffer from some serious limitations:

1) As discussed before, n, the number of times a user can authenticate to the server, is finite (see 2 for why n cannot be made very large). Further, Alice is forced to choose a new password every time n reaches 1 and the system should be reinitialized. The old password cannot be reused again. This user unfriendly requirement may not be desirable in many environments.

2) The system is computationally intensive for the client especially when n is large. For example, with n=500, the client should compute 499 hash functions (i.e. $h^{499}(p)$) for authenticating first time, 498 for second time and so on. Hence with n=500, the client should compute about 250 hash functions per authentication on an average. Clearly, the scheme is unsuitable for mobile devices having low computational resources.

Despite these limitations, being the only symmetric key password based authentication system to resist eavesdropping as well as password file compromise, Lamport hashes is implemented, standardized and is widely used [2, 3, 4, 5, 13].

## 3. The Proposed Construction

Now we proceed to describe the proposed construction for the authentication system. The basic idea as discussed before is to repeatedly require the insertion of user password after a fixed distance in the hash chain. The links at which the insertion of the password is required may be made public and stored at the host (server). The host would then transfer one of those links to the user for aiding the computation of the user. This results in a significant decline in the computational requirement of the user.

### 3.1 The System Description

We define two system parameters, N and R. N is the maximum number of times the user might possibly authenticate using this scheme before re-registration is required; and $\lfloor N/R \rfloor$ is the distance (or number of links) in the hash chain after which password insertion is required. Note that $\lfloor N/R \rfloor$ also represents the maximum number of hashing operations that the user may be required to do for authentication at any point in time. Further, R represents the storage required at the host H. The selection of the right values N and R calls for requirement analysis and is a tradeoff between computation and storage.

We now consider the case when the user U sets up an account with the host H for the first time. U sends a desired user name and H returns with the pair (N, R). Then U selects a password p and computes the following function

$$\mathbf{И}^{i(N - N\%R)/R}(p)$$

for every integer i ( $1 \le i \le R$ ). In addition, U also computes $\mathbf{И}^N(p)$.

The function $\mathbf{И}^x(p)$ is defined by the following recurrence relation

$$\mathbf{И}^{k+1}(p) = h(\mathbf{И}^k(p) + \delta * p)$$

where $\delta = 1$ for $k = i(N-N\%R)/R$

$= 0$ for $k \neq i(N-N\%R)/R$

and, $\mathbf{И}^0(p) = p$

User U then sends all these computed components i.e. $\mathbf{И}^{i(N - N\%R)/R}(p)$ for i ( $1 \le i \le R$ ) and $\mathbf{И}^N(p)$ to the host H which stores them in its database for future authentication sessions. Thus, note that H stores R hash function values in its database for aiding the user computation during future authentication sessions.

**Simplified Equations**

The above description is the generalized form of the system when R may not be a multiple of N. However, if N is a multiple of R, the equations simplify. In this

case U sends a desired user name and H returns with a pair (N, R). Then U selects a password p and computes

$$\mathbf{И}^{iN/R}(p)$$

for every integer i ( $1 \leq i \leq R$ )

The function $\mathbf{И}^{x}(p)$ is defined by the following recurrence relation

$$\mathbf{И}^{k+1}(p) = h(\mathbf{И}^{k}(p) + \delta * p)$$

where $\delta = 1$ for $k = iN/R$

$= 0$ for $k \neq iN/R$

and, $\mathbf{И}^{0}(p) = p$

User U then sends these R computed values to the host H which stores them in its database for future authentication sessions.

An important security property of the system is that for all values of k where $\mathbf{И}^{k}(p)$ is sent to the host, $\delta$ would be equal to 1, i.e. $\mathbf{И}^{k+1}(p)$ would not be computable without the knowledge of password p. Recall that $\mathbf{И}^{k-1}(p)$ is never computable from $\mathbf{И}^{k}(p)$ due to the non-invertibility of the hash function used. Hence, all the values sent to the server are non-security sensitive.

The generalized equations are just provided for the sake of completeness. This is because since N and R are user selected parameters, she can always select them in such a way that N is a multiple of R. Hence, for all further discussions, we take this simplified case to avoid confusion. However, unless otherwise stated, the discussion will also apply to the generalized case.

**Authentication of the user U**

Suppose that U wishes to authenticate herself to host H for the $t^{th}$ time. The process operates as follows.

1. The user U identifies herself to the remote host H by login name.

2. H sends the following pair of values to U

   $$(n , \mathbf{И}^{k}(p))$$

   where $n = (N-t)\%R$ with $n \neq 0$ and $k = N-t-n$

   For values of t for which n = 0, H simply increments t by one and does the calculation again. This happens when t is a multiple of R. This case requires no computation by U and therefore can be easily exploited.

3. U calculates $\mathbf{И}^{k+n}(p)$ and sends it back to H as a one time password.

   From the password equation, this is equal to

   $$h^{n}(\mathbf{И}^{k}(p) + p)$$

   It should be noted here that during every login the knowledge of p is required and p is never transferred as plaintext.

4. H takes this value $\mathbf{И}^{k+n}(p)$ and hashes it R-n times and then matches it with $\mathbf{И}^{k+R}(p)$ that is there in the database with H. The authentication succeeds if the value matches. Alternatively, a better method is to have H store the last one time password. In that case, H just needs to hash the received one time password and compare with the stored one.

5. Next time U wants to access the system, she will be prompted with values for t+1.

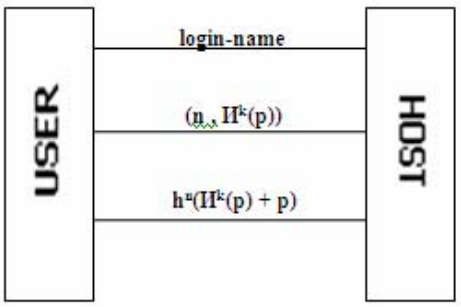


**Figure 1:** Architecture of the proposed protocol

An instance of the authentication session is given below for N = 1000, R = 100 and t = 348.

1. The user U identifies herself to the remote host H by login name.

2. H sends the following pair of values to U

   $$(52 , \mathbf{И}^{600}(p))$$

   since $n = (1000-348)\%100 \Rightarrow n = 52.$

   $k = 1000-348-52 \Rightarrow k = 600.$

3. U calculates $\mathbf{И}^{652}(p)$ and sends it back to H as a one time password. From the password equation, this is equal to

   $$h^{52}(\mathbf{И}^{600}(p) + p)$$

4. H takes this value $\mathbf{И}^{652}(p)$ and hashes it once. H now compares the obtained value with the last one time password stored in its database i.e. with $\mathbf{И}^{653}(p)$. The authentication succeeds if the value matches. H replaces the stored last one time password i.e. $\mathbf{И}^{653}(p)$ with the sent $\mathbf{И}^{652}(p)$.

5. Next time U wants to access the system, she will be prompted with values for 349.

| | Resistance to eavesdropping | Resistance to server compromise | Number of authentications before re-initialization | Client computational requirements | Suitable for mobile clients? | Resistance to DoS attacks |
|---|---|---|---|---|---|---|
| Lamport's System | Yes | Yes | Low | High | No | Yes |
| Proposed System | Yes | Yes | High | Low | Yes | Yes |

**Table 1** An objective Comparison of the Proposed System with Lamport's System

### 3.2 Discussion and Analysis

Architecture of the proposed scheme is depicted in Figure 1. We now consider practical aspects of the scheme. A major improvement over the previous methods [1, 2, and 13] is the significant reduction in computational requirements per authentication session and increase in the number of logins before re-initialization.

#### 3.2.1 Choices of N and R

We now consider how N and R should be chosen. N is the 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. For N = 10,000, even if N/R is kept equal to 10, i.e., the host is required to store 1000 hash function outputs which are commonly of 128 bits (16 bytes) each, even an ordinary hard disk drive of 20 Gigabytes is enough for supporting more than a million users. It is worthwhile to remark here that today; even personal computers have more storage than 20 GB.

Thus for N=10,000 and R=1000, a user U is required to compute N/2R = 5 hash functions per authentication session. Considering that U logs in 3 times a day, the system would last for about 10 years before a re-registration is required.

#### 3.2.2 Complexity

We start by considering the storage, computation and communication complexity of the scheme.

- Storage: the requirements for the host are to store R hash values, the last one time password and one integer t that accounts for the number of logins done till date. An obvious optimization that may reduce the storage to about half on an average is to delete the used up hash values that would never be required again. As with all password based authentication systems, the user U is not required to store anything; all he needs to do is remember a password p that he will use for all authentications.

- Computation: the host verifies the one time password sent by user by computing just a single hash function and one comparison with the stored last one time password. The user is required to do N/2R hash function computations on an average for authentication. At no point in time does the number of hashes computed exceed N/R.

- Communication: the host sends the user a hash value and an integer t. The user returns only a single hash value.

We summarize the comparison of the proposed one time password system with the Lamport's system in Table 1.

## 4. Conclusion and Future Work

The N/R One-Time password system seems to be effective and easy to deploy. It requires very low computing power available with almost all computing devices. The system was specifically designed keeping in mind the mobile devices with constrained computing and communication capacity. The system overcomes the limitations of the previous one time

password systems by significantly reducing the client side computing requirements. This is done by a introducing a hash chain whose links are computed in such a way that a set of the links is not security sensitive and thus can be stored at the host. One of these links is supplied by the host to the user aiding in the one time password computation of the user.

We believe that our construction of hash chains may be of independent interest. Given the large number of systems in which hash chains are deployed, we believe that our construction should also be usable in other environments apart from one time passwords.

## References


[1] L. Lamport, “Password Authentication with Insecure Communication”, Communications of the ACM 24.11 (November 1981), pp 770-772.

[2] N Haller, "The S/KEY One-Time Password System", Proceedings of the ISOC Symposium on Network and Distributed System Security, pp 151-157, February 1994.

[3] N. Haller, “The S/KEY One-Time Password System”, RFC 1760, February 1995. Available from http://www.ietf.org.

[4] N Haller, “A One-Time Password System”, RFC 1938, May 1996. Available from http://www.ietf.org.

[5] N Haller, C. Metz, P. Nesser and M. Straw, “A One-Time Password System”, RFC 2289, Feb 1998. Available from http://www.ietf.org.

[6] S. Micali, "Eficient Certificate Revocation," Proceedings of RSA '97, and U.S. Patent No. 5,666,416.

[7] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Eficient and Secure Source Authentication for Multicast," Proceedings of Network and Distributed System Security Symposium NDSS 2001, February 2001.

[8] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Eficient Authentication and Signing of Multicast Streams over Lossy Channels," Proc. of IEEE Security and Privacy Symposium S & P 2000, May 2000.

[9] A. Perrig, R. Canetti, D. Song, and D. Tygar, "TESLA: Multicast Source Authentication Transform", Proposed IRTF draft, http://paris.cs.berkeley.edu/ ~perrig/

[10] S. Stubblebine and P. Syverson, "Fair On-line Auctions Without Special Trusted Parties," Financial Cryptography '01.

[11] R. L. Rivest and A. Shamir. PayWord and MicroMint-two simple micropayment schemes. In Mark Lomas, editor, Proceedings of 1996 International Workshop on Security Protocols, volume 1189, Lecture Notes in Computer Science, pages 69-87. Springer, 1997.

[12] B. Schneier and J. Kelsey, “Cryptographic support for secure logs on untrusted machines”, In Proceedings 7th USENIX Security Symposium (San Antonio, Texas), Jan 1998.

[13] D.L. McDonald, R.J. Atkinson, C. Metz "One-Time Passwords in Everything (OPIE): Experiences with Building and Using Strong Authentication," In Proc. of the 5th USENIX UNIX Security Symposium, June 1995.

[14] N.Asokan, G.Tsudik and M.Waidners, “Server-supported signatures”, Journal of Computer Security, November 1997.

[15] X. Ding, D.Mazzocchi and G.Tsudik. Experimenting with Server-Aided Signatures, Network and Distributed Systems Security Symposium (NDSS ’02), February 2002.