Independent One-Time Passwords

Aviel D. Rubin
Bellcore

Independent One-Time Passwords

Aviel D. Rubin
rubin@bellcore.com
Bellcore
445 South St.
Morristown, NJ 07960

Abstract

Existing one-time password (OTP) schemes suffer several drawbacks. Token-based systems are expensive, while software-based schemes rely on one-time passwords that are dependent on each other. There are disadvantages to authentication schemes that rely on dependent OTP's. It is difficult to replicate the authentication server without lowering security. Also, current authentication schemes based on dependent OTP's only authenticate the initial connection; the remainder of the session is assumed to be authenticated. Experience shows that connections can be hijacked. A new scheme for generating one-time passwords that are independent is presented. The independence property enables easy replication of the authentication server, and authentication that is persistent for the lifetime of a connection. This mechanism is also ideally suited for smart card applications. Our implementation and several applications are discussed.

1 Introduction

The authentication of users in a large, distributed environment is an increasingly difficult task. As networks and software grow in sophistication, so do the means and methods of malicious attackers. These intruders must be denied access to protected systems and data without also excluding legitimate users.

There are several ways to verify the identity of a user in a computer system. The most common are *what they know, what they have,* and *who they are.* The last involves biometrics such as retinal scans and fingerprints; these technologies have not yet arrived. The other two techniques are more common. Users may prove their identity by knowing a password or possessing a token.

Computer crackers utilize enormous resources to obtain the information necessary to impersonate other users. Sniffer programs that capture password information from packets from well-known services such as *telnet* and *ftp* have been found all over the Internet. In addition, studies have shown that users pick poor passwords [7], and thus they are easy to attack with dictionary search. Systems such as Kerberos [11] are especially vulnerable to this because the attack can take place off-line. In most systems, as long as the compromise of a reusable password is not detected, the imposter can assume all of the privileges of the unsuspecting user.

Authentication systems based on one-time passwords are more secure than ones that rely on reusable passwords. For example, remote access usually requires the user to enter a password or pass phrase. This secret usually travels across insecure networks in the clear. In the case of one-time passwords, the danger of eavesdropping is eliminated because once a password is used, it is no longer useful. If a one-time password system is implemented properly, breaking it requires sophisticated, active attacks that are beyond the abilities of most attackers, such as meet in the middle attacks.

The two best-known one-time passwords systems are S/KEY™[5] and Secure ID™. S/KEY is a software solution, and Secure ID is an expensive hardware solution that requires a secure authentication server, and careful administration. We will focus on software solutions as they are much cheaper and easier to install. S/KEY has the additional benefit that there are no secrets stored on the authentication server. The value of this benefit is discussed in Section 2.2.2. However, the one-time passwords in S/KEY are not entirely independent. This causes several security risks and poses limitations on how the system can be used. These are discussed in a later section.

This paper describes a technique for achieving independent one-time passwords. The method is compared to S/KEY, and their relative merits are evaluated.

2 Previous work

This section describes the two most popular authentication systems that use one-time passwords. Secure ID is a hardware solution, while S/KEY works entirely in software.

S/KEY is a trademark of Bellcore.
Secure ID is a trademark of Security Dynamics.

2.1 Secure ID

Secure ID is a one-time password system where physical tokens are used to authenticate users. Each user possesses a card that displays a six digit number through a glass display. The user also picks a PIN number. The card is about 3 millimeters thick and is relatively fragile; it cannot fit in a wallet or pants pocket. The number on the card changes every n seconds, where n is a configurable quantity, usually about 30 seconds. The algorithm used by the card is proprietary, but it is known that each card contains a unique secret seed. A copy of each seed also exists at the authentication server. The seed is used to generate the next number that is displayed by the card.

There are several strategies for breaking Secure ID. The product is sold on the premise that these are infeasible. One way to defeat it is to break the secret algorithm to predict the next number that will be displayed. In addition, the attacker must eavesdrop on a previous authentication to obtain the PIN, which is sent in the clear each time. Another attack is the meet in the middle attack. Here, an attacker eavesdrops on an authentication session, records the one time password, and prevents the message from reaching the authentication server. Then, he uses the one time password, within the time window allowed by the card, to authenticate himself. If the authentication server is replicated, then this attack works even if the real authentication message is not blocked. Active meet in the middle attacks are very difficult to prevent, and no authentication system in wide-spread use is immune to them.

2.2 S/KEY

This section briefly describes the S/KEY authentication system. Further details can be found in the original paper [5].

2.2.1 How S/KEY works

Before using S/KEY for authentication, users perform an initialization step. A user logs into a secure authentication server. The login must be local or over a secure connection; a remote login here defeats the purpose of S/KEY. Then, he selects a secret password and n, the number of one-time passwords to generate. The software then applies n iterations of a one-way hash function to the password. The final result is stored on the authentication server, and the initialization is complete.

The authentication server keeps track of the number of times that each user authenticates himself. The first time the user logs in with S/KEY, he is prompted with the number $n-1$. The user types in his secret password on his local machine, and the software applies $n-1$ iterations of the one-way hash function to the password. The result is sent across the network to the authentication server. The authentication server applies the hash function one time to this message. The result is compared to the value that was stored earlier. If they match, then the authentication is successful. The authentication server then replaces the stored value with the new message that it receives and decrements the password count, n, to prepare for the next authentication.

If the user does not have the S/KEY client software (for example, when using a dumb terminal) or does not trust his machine, then there is another mode of operation. Before leaving his trusted environment, the user generates and prints a list of one-time passwords. This printout must be guarded very carefully. Then, when the user authenticates, he simply uses his list to send the requested one-time password to the authentication server.

2.2.2 Secrets on the server

One of the touted advantages of S/KEY over other schemes is that no secrets are stored on the server. All the server needs to maintain is the last OTP that was used for authentication. However, it is not clear that this is really an advantage in the Unix®environment. Although the n^{th} password is not a secret, it is still important to guarantee that its value on the server cannot be changed. It is safe to say that preventing an intruder from becoming *root* on the server is a requirement. However, if we can guarantee that no intruder will be able to assume root privileges, then storing secrets on the server is easy. Any secret can be stored in a directly that is only readable to root. Therefore, a server that is secure from data tampering can easily be used to store secrets. Thus, it is not clear that the fact that S/KEY does not require secrets on the server is such an advantage in the Unix environment. The OTP scheme described in this paper requires storage of a secret database on the authentication server.

2.2.3 Weaknesses of dependent OTP's

This section discusses several shortcomings of authentication systems that rely on dependent OTP's derived from a secret password.

Susceptibility to off-line dictionary attack

The one-time passwords travel across the network in the clear. Any eavesdropper can record them. If the relationship among the OTP's is well-known (e.g. a hash function), a malicious user can apply the function to candidate passwords such that if the result matches a one-time password, then the

Unix is a registered trademark of Unix Systems Laboratories.

reusable password is compromised. Thus, the security of such systems relies on users picking good passwords - a very bad assumption. The next release of S/KEY will place constraints on the passwords to make them more difficult to guess.

Danger of reusable password compromise

The secret password chosen by the user is the key to all of the one-time passwords. This password must be protected at all costs. There are many ways an inexperienced user may accidentally send the password across the network. For example, if the user performs authentication from a remote login shell, every keystroke of his travels across the network, although he might not realize it. Often, in a Unix/X-windows environment, users have windows open to different machines in their network. Anything typed in a nonlocal window travels across the network. In addition, malicious users can exploit weaknesses of X-windows to read keystrokes on another machine. There is a program, *xkey*, that has been widely distributed on the Internet, that accomplishes just that. Authentication systems with OTP's that are seeded with reusable passwords offer users poor protection from people on the local network of the client. They are therefore not very suitable for a university or any public environment.

Difficult to maintain multiple servers

It is often desirable to have more than one authentication server for higher availability of the service. Dependent one-time password systems make it difficult to replicate the authentication server (AS). Each AS must know the current one-time password number. That is, if a user authenticates 10 times, then every AS must know that 10 passwords have been used. Any time one AS is out of sync with any of the others, the system is easy to defeat. For example, say that AS-1 believes that there have been 10 authentications, and AS-2 believes there have been 9. An eavesdropper who recorded the last authentication to AS-1 can replay the one-time password to AS-2 and authenticate successfully.

All of the shortcomings described in this section result from the dependency of the one-time passwords on each other and on the reusable password of the user. This paper presents a one-time password scheme that generates *independent* one-time passwords for the users. Section 3.4 discusses how replication of the authentication server is accomplished.

Hijacked connections

In addition to the shortcomings listed above, there is a weakness shared by most current authentication systems. After the initial connection is authenticated, the remainder of the session remains unchallenged. Therefore, any malicious intruder that can duplicate the state of the authenticated client, while breaking off his connection, can take over the session. Attacks such as these are alluded to by Bellovin [1], and they are occurring more frequently [3].

3 A new approach

This section presents an authentication scheme based on a new mechanism for generating one-time passwords (OTP's) that are independent.

3.1 Pseudo-random functions

The new approach presented here is based on a class of functions called *pseudo-random functions (PRF's).* The notion of a PRF was introduced by Goldreich *et. al.* [4]. A function is considered random if no polynomial-time algorithm can distinguish a computation on chosen inputs that outputs the correct values from one that outputs random values. Goldreich *et. al.* give a construction to transform any one-to-one one-way function to a PRF.

It is currently believed that there are good PRF's. A good encryption algorithm should have the property that a polynomial-bounded adversary, without knowledge of the key, should not be able to gain any information about the plaintext, given a ciphertext. Thus, strong encryption algorithms are good candidates for PRF's.

3.2 Generating independent OTP's

If OTP's are truly independent, then the authentication server must store them all individually. This is an unreasonable requirement for a large system with many users. However, if there is a way to generate all the passwords from a small amount of information, then they cannot be totally independent. The technique described here uses a pseudo-random function to generate OTP's from an initial secret key. Finding a relationship between any two OTP's is equivalent to breaking the PRF.

Our implementation uses three rounds of DES [8] (triple-DES) as the PRF.[1] This function is believed to be a PRF, and it is resistant to differential [2] (and linear [6]) cryptanalysis. This property insures that the outputs of the algorithm cannot be related in polynomial time and/or space even if the inputs are very similar. Our implementation utilizes this strength of triple-DES to produce independent OTP's. For the remainder of the paper, we will assume that triple-DES is a good PRF.

Before OTP's can be generated, a user must register with the authentication server (AS). It is assumed that the AS has some means of authenticating the initial registration.[2] The AS maintains a table with certain information about each user, such as an identification number (ID) and a random key that is generated on his behalf. This information, along with a couple of number, i and n, and some other data are stored in the authentication server's table. $i-1$ represents the number of OTP's already used and n represents the total number of OTP's for a user. n is optional; it is included in our implementation for reasons that are explained later. The following is a simplified table for 3 users.

ID #	Secret Key	i	n	...
459332	da54f8cd703b75dc	1	500	...
459181	e0bf9bd6b0dfcee6	55	500	...
458932	b5b3c2c8d36bf2ab	1	450	...
...	...	...	...	...

User 459332 has never authenticated, and he may authenticate 500 times. On the other hand, user 459181 has authenticated 54 times. The users are not aware that a secret key has been assigned to them.

For some applications, it may be desirable to protect the entire table with a master key. Rather than constantly encrypting and decrypting the table, the master key is included in the calculation of every OTP. Thus, to calculate the i^{th} OTP for a user, the authentication server computes:

$$X = f(MK, K_{user}, i)$$

[1] We have also implemented the system with a keyed version of MD5 [9] because of export restrictions on triple-DES. In practice, any PRF will due.

[2] This process may take place off-line or require that users go somewhere in person.

where f is a suitable PRF, MK is the master key for the authentication server's table, and K_{user} is the secret key associated with the user. Thus, X is a function of the master key, the secret key of the user and the OTP number, i. Any change in the input will result in an X' that is unrelated to X. The AS then computes

$$OTP = g(X)$$

where g is a function that converts any string of bits into a list of small, human-readable passwords. In our implementation, we borrowed code from S/KEY for the function g. We used 3-DES for the function f. The secret key for each user is 192 bits long and consists of three random DES keys. The first key is exclusively or'ed with the master key before 3-DES is applied. Thus,

$$X = DES(DES(DES(i, K^1_{user} \oplus MK), K^2_{user}), K^3_{user})$$

This formula will produce a unique X for each value of i. Also, as long as the secret keys for each user are different, the probability that it will produce the same X for two users for the same value of i is negligible. Finally, without the master key, it is infeasible to compute X.

3.3 Using the OTP's

This section discusses several applications of the one-time password scheme described above.

3.3.1 Internet billing

Our one-time password scheme is being used to authenticate users to a billing server on the Internet. There are several serious considerations to any service that is offered in such an insecure environment. It can be assumed that there is eavesdropping on all communications, that workstations and user accounts may be compromised, and that user keystrokes can be monitored. It is impossible to store any long-term secret in such an environment. Therefore, any public-key system where the private key is stored in a password protected file (such as PGP™[12]) is inadequate.

In our billing system, the users register on the phone with a credit card. There are various safeguards in place, and how this is achieved is not the subject of this paper. The billing server (formerly the AS) generates 350 OTP's for the user by default, or more if requested. A booklet containing a numbered list of OTP's is sent to the user by some trusted out of band mechanism.[3] When the user needs to authenticate, he is prompted for OTP, k, which he looks up in the booklet. After the OTP number k is entered, the billing server computes the k^{th} OTP and compares. If they match, then authentication succeeds, otherwise it fails.

The scheme presented here is vulnerable to over the shoulder attacks. In a public computer room at a university, it may be possible for someone to copy passwords from the current page of a user's booklet while he is entering his OTP. To counter this, the OTP numbers are not requested sequentially. In fact, the OTP's are guaranteed to be far apart in the booklet. This is accomplished as follows. When the user registers, he is assigned a number, j, that such that $\frac{n}{2} < j < n$ and j is relatively prime to n. j is also chosen to be as closer to $\frac{n}{2}$ than to n if possible. The billing server table from the previous example looks like this.

ID #	Secret Key	i	j	n
459332	da54f8cd703b75dc	1	331	500
459181	e0bf9bd6b0dfcee6	55	281	500
458932	b5b3c2c8d36bf2ab	1	241	450
...	...	...	...	...

PGP is a trademark of Phil Zimmerman.

[3]E.g. registered mail.

To calculate the OTP number for the next authentication, the billing server computes

$$k = i * j \ mod \ n$$

k is always between 1 and n. Also, as i goes from 1 to n, k equals each value between 1 and n exactly once. This results from the relative primality of j and n. For example, the user 459181 will be prompted for passwords 281, 62, 343, 124, etc. There are 50 OTP's on each page, so these OTP's will appear on pages 6, 2, 7, 3. etc. Thus, it is unlikely that an intruder will be able to copy down an OTP that will be prompted for in the near future.

After user 459181 enters OTP number 281, the authentication server computes

$$OTP'_{281} = g(f(MK, K_{459181}, 281))$$

and checks to see if OTP'_{281} matches what the user entered. OTP number 62 is calculated as

$$OTP'_{62} = g(f(MK, K_{459181}, 62)).$$

The corresponding value of k is used in the i^{th} OTP calculation. After each successful authentication, the value of i is incremented until it reaches n. At that point the user is removed from the table, and he must register again. Unsuccessful authentication attempts are logged.

S/KEY can be used in this mode as well. A user prints out a list of OTP's and uses them in the same manner as described above. However, due to the dependence of the OTP's on each other, they must be entered in the correct order. In S/KEY, if a user accidentally enters the wrong OTP from his list, he compromises all of the passwords between the one he enters and the correct one. Also, there is no way to defend against over the shoulder attacks by jumping around the password list.

3.3.2 Limited access

The scheme described above is especially useful when a company wishes to allow limited access to a business partner. For various reasons, one company may wish to grant access to specific machines, services or files to several individuals outside of their organization. To do this, a machine can be dedicated as a special authentication server. Then, each of the privileged users is given a small list of OTP's. In this manner, a user can be allowed to log into a machine a maximum of ten times. The one-time password technique described above allows for a flexible authentication scheme.

3.3.3 A passive attack

There is a passive attack on any authentication scheme where the user types in an OTP manually [10]. We illustrate how an eavesdropper, Eve, can use information from an active session to beat a legitimate user, Bob. Eve situates herself so that she can read any packet between Bob and the authentication server. Assume that Bob must type in 6 short words from a known dictionary, and that Bob types in these words at normal typing speed. Say that the OTP is HOW LOON CRY SOFT PAR MEND. Eve sets up several simultaneous authentication attempts to the authentication server. Immediately after Bob has typed HOW LOON CRY SOFT PAR M, Eve automatically sends candidate OTP's to the authentication server by trying all possible values for the last OTP word from the dictionary. In all likelihood, Eve will authenticate before Bob finishes typing the OTP. This attack can be easily prevented by only allowing one authentication attempt per user at a time.

3.4 Replication of the authentication server

As explained earlier, both S/KEY and Secure ID are limited in their abilities to support multiple authentication servers. However, using the independent one-time password scheme described in this paper, replicating the AS is easy.

The following example demonstrates how two authentication servers, AS_1 and AS_2 are used. The generalization to n authentication servers is obvious. Say that user 459181 registers with 500 OTP's. Half of the OTP's are used to authenticate to AS_1 and the other half are used for AS_2. This is accomplished as follows. The n in the previous examples represents the maximum value of i, and the modulus that determines the next OTP number. These two roles are now split into two variables, n_1 and n_2. The former represents the maximum value of i for a user at the AS, while the latter represents the modulus. AS_1 stores the following for user 459181.

ID #	Secret Key	i	j	n_1	n_2
459181	e0bf9bd6b0dfcee6	1	281	250	500

AS_2 stores the following for the same user:

ID #	Secret Key	i	j	n_1	n_2
459181	e0bf9bd6b0dfcee6	251	281	500	500

Thus, there are 500 OTP's associated with this user. 250 of these correspond to each AS. Given the j value of 281, AS_1 will prompt for OTP's 281, 62, 343, 124, etc. When i reaches 250, the user will be removed from AS_1's table. AS_2 will prompt for 31 (251 * 281 mod 500), 312, 93, 374, etc. As 281 and 500 are relatively prime, no number appears in both lists. Given the OTP number, the user key, and the master key, the actual OTP is easily computed by each AS. The user is given one list without knowing which OTP's are associated with which AS. Therefore, the two authentication servers must use the same master key.

3.5 Persistent authentication with OTP's

Traditional authentication systems such as S/KEY and Secure ID only authenticate once per session. If the authentication succeeds, there is little to protect the user from a hijacked connection. We implemented a prototype authentication system that uses the OTP scheme described above for persistent authentication. That is, authentication is repeated every t seconds, where t is a configurable system parameter. The persistent authentication is transparent to the user as long as the connection is legitimate.

Unlike the Internet billing application above, persistent authentication requires computing on the client side. This can only be achieved with a secure client machine or with a tamper resistant smart card (see Section 3.6). Our implementation assumes a secure client machine; it was designed to map directly into a smart card implementation. In our implementation, the authentication server has a table containing user ID's, a secret key for each user (3 DES keys), and a number, i. These are the same 3 data items in the earlier examples. However, for persistent authentication, we also assume that each user is in possession of his secret key. These keys must be distributed off-line. Ideally, they reside in a smart card. Also, there is no master key.

For the initial authentication, the client software calculates the first OTP,

$$g(f(K_{user}, 1))$$

and sends it to the AS. The AS performs the same calculation to verify the OTP. We use triple-DES for the function, f, as before. Subsequent OTP's are generated by incrementing the number in the PRF. Next, the client forks a process that sleeps, but wakes up every t seconds and sends the next OTP to the server. The server sets a timer, and if the next OTP is not received in time, kills the connection.

If the OTP is received, and it is correct, then the server acknowledges it with the next OTP. Thus, the client and the server mutually authenticate every t seconds, and two OTP's are used each time. If either side does not send the correct OTP at the right time, the connection is terminated. Thus, if an intruder hijacks the connection, and he is not in possession of the user's secret key, his connection is killed when the current time interval expires.

3.6 Authentication with smart cards

The independent one-time password scheme described here is ideally suited for smart card applications. As described in the previous section, each smart card contains a secret key (3 DES keys, in our example). We assume that the tamper resistant nature of smart cards means that there is no way to obtain any information about the key without destroying it in the process. A virtually unlimited number of OTP's can be computed on both the client and server side using a PRF, such as triple-DES. The OTP's are independent, and thus, none of the shortcomings associated with dependent OTP's applies. Many interesting applications can be designed using smart cards, along with independent OTP's.

4 Conclusions

The independent OTP scheme described in this paper has been implemented. We are currently using the authentication system described in Section 3.3.1 in our billing server. The advantages offered by the new technique for generating OTP's are easy replication of the authentication server, persistent authentication for the lifetime of a connection, and a natural mapping to smart card applications. The calculation of each OTP requires one application of a pseudorandom function, as opposed to the many iterations of S/KEY.

Acknowledgements

The author thanks Bill Aiello for very helpful comments and explanations of pseudorandom functions, R. Venkatesan for suggesting triple-DES as a PRF, Milt Anderson and Yacov Yacobi for helpful comments, Phil Servita and Neil Haller for pointing out the passive attack on some OTP schemes and a possible solution, and Ali Bahreman for integrating the code into a working prototype.

References

[1] Steve Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communication Review*, 19(2):32–48, April 1989.

[2] E. Biham and A. Shamir. *A Differential Cryptanalysis of the Data Encryption Standard.* Springer-Verlag, 1993.

[3] CERT. Cert advisory CA-95:01, January 1995.

[4] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pages 464–479, 1984.

[5] Neil Haller. The s/key(tm) one-time password system. *Symposium on Network and Distributed System Security*, pages 151–157, February 1994.

[6] M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor, *Advances in Cryptology — Eurocrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397, Berlin, 1994. Springer-Verlag.

[7] Robert Morris and Ken Thompson. Password security: A case history. *CACM*, 22(11):594–597, November 1979.

[8] National Bureau of Standards. Data encryption standard. *Federal Information Processing Standards Publication*, 1(46), 1977.

[9] R. Rivest. The md5 message digest algorithm. *RFC 1321*, April 1992.

[10] Phil Servita. Personal communication, 1995.

[11] J.G. Steiner, B.C. Neuman, and J.I. Schiller. Kerberos: An authentication service for open network systems. In *Usenix Conference Proceedings*, pages 191–202, Dallas, Texas, February 1988.

[12] P. Zimmerman. Pgp user's guide. December 4, 1992.