

DAUTH: Secure Offline Verification of One-Time Passwords

Abstract

A one-time password (OTP) is a single-use value that authenticates a user for access to a protected resource. In most implementations the use of OTPs requires access to an on-line authentication service to verify the value supplied by the user. We propose a scheme, called Disconnected Authentication (DAUTH) that allows OTPs to be verified without access to a server. The DAUTH scheme allows OTPs to be used for local authentication without exposing the long-term secret used to generate the OTP values.

Karl Ackerman, Piers Bowness, John Brainard, Bill Duane
RSA Security Bedford, MA 01730, USA
e-mail: {kackerman, pbowness, jbrainard, bduane}@rsasecurity.com

TABLE OF CONTENTS

1. INTRODUCTION	PAGE 1
1.2 Organization	PAGE 1
2. REQUIREMENTS AND ASSUMPTIONS	PAGE 1
2.1 Assumptions	PAGE 1
2.2 Security Requirements	PAGE 2
2.3 Implementation Requirements	PAGE 2
3. DAUTH DESIGN	PAGE 2
4. DAUTH CRYPTOGRAPHY	PAGE 3
4.1 Functions	PAGE 3
4.2 System Constants	PAGE 4
4.3 Data Stored on the Server	PAGE 4
4.4 Local Secret Encryption	PAGE 4
4.5 Verifier Generation	PAGE 5
4.6 OTP Verification	PAGE 5
5. DAUTH IMPLEMENTATION	PAGE 6
5.1 Cryptographic Algorithms	PAGE 6
5.2 Dayfile Structure	PAGE 7
5.3 Dayfile Header Record	PAGE 8
5.4 Dayfile Entry Record	PAGE 8
5.5 Password Encryption	PAGE 9
APPENDIX A. ESTIMATING DAUTH SECURITY	PAGE 9
A.1 Estimation Parameters	PAGE 9
A.2 Computing OTP Entropy	PAGE 10
A.3 Computing Verification Time	PAGE 10
A.4 Computing Attack Time	PAGE 11
A.5 Results	PAGE 11
A.6 Conclusions	PAGE 13

1 Introduction

1.1 OTP Background

One-Time passwords, such as those produced by the RSA SecurID[®] token, are typically used to protect remote access to network resources. In a typical OTP implementation, each user carries a hand-held device or token that produces a sequence of single-use values that are used to authenticate the user for access to a protected resource. The OTP values are generated by combining a static secret value with a dynamic value such as the time, an event counter, or a random challenge. The static secret is stored in the token and is shared with an on-line server which performs the verification.

OTPs provide significant security advantages over static passwords, in that they are not vulnerable to “sniffing” attacks, where the cleartext password is viewed on the network, then replayed at a later time. OTP usage also prevents “dictionary” attacks where likely passwords are chosen from a pre-compiled list.

The verification of OTP values normally requires access to an authentication server. This requirement makes the use of OTP for protecting local access problematic, especially in the case of mobile users who may not always have a direct connection to a server. The DAUTH mechanism allows a local system to download a set of short-term verifiers from the authentication service. The short-term verifiers are used, if the server is unavailable, to verify OTP values presented by the user.

1.2 Organization

This document describes three major aspects of disconnected authentication. Section 3 discusses the design principles of DAUTH. Section 4 discusses the cryptographic methods used in generating OTP verifiers and protecting local secrets. Finally, section 5 illustrates some implementation issues based on the use of DAUTH in the RSA SecurID for Microsoft[®] Windows[®] product.

2 Requirements and Assumptions

To develop a useful system for off-line verification of OTPs, it is helpful to consider the environment in which such a system is used. This leads us to a set of assumptions about the environment, and some requirements for security and usability.

2.1 Assumptions

In developing a system for off-line use of OTPs, we make a number of assumptions about the environment in which they are used. The assumptions include:

- The authentication server is completely trusted. It is assumed to reside in a physically secure environment and must be trusted to hold long-term secrets.

- The local system is only partially trusted. It is not assumed to be physically secure. It may be trusted with short-term secrets only.
- Users will be off-line for a maximum period of time, specifically a small number of weeks. We can require users to “re-synchronize” with the authentication server periodically.

2.2 Security Requirements

We wish to allow for local authentication, without sacrificing the security advantages provided by the OTP. To accomplish this, the system must meet the following security requirements:

- A compromise in the local environment must not lead to a compromise of the secrets stored on the server.
- To prevent replay attacks cleartext OTP values must not be available in the local environment.
- Dictionary attacks against the verifier must be significantly more difficult than comparable attacks against a static password.

2.3 Implementation Requirements

In addition to the security requirements listed above, a DAUTH scheme must meet a number of practical requirements if it is to be successfully deployed.

- Verification of OTP values in an offline setting should not add a significant delay to the user. Verification times should be at most one or two seconds in the typical case.
- The DAUTH scheme must be usable with existing OTP generators. Requiring a wholesale redeployment is not practical.

3 DAUTH Design

The simplest way to verify an OTP when offline is to download the long-term secret from the server. This violates our first security requirement since the local environment is not trusted to protect long-lived secrets.

An alternative approach requires generating the OTP values indirectly, from a short-term secret that is derived from the long-term secret. This approach meets all of our security requirements, and it is a viable approach for new OTP

designs, but it is not backward-compatible with existing OTP implementations that generate the OTP value directly from the long-term secret.

The DAUTH scheme downloads a list of verification values, one for each of the set of possible OTP values that may be used while offline. The security requirement against replay attacks precludes downloading the actual OTP values. The verification values are derived from the actual OTP values by hashing. The verifier computes the hash on the entered OTP value and checks the result against the list of verifiers.

OTP values entered by users are typically fairly short strings, consisting of a static PIN (Personal Identification Number) of six digits or characters and a dynamic OTP value of six or eight decimal digits. The resulting value does not have sufficient entropy to resist a dictionary attack by an attacker with a moderate level of resources.

The use of a simple hash does not protect against dictionary attack. Pre-computed dictionary attacks are prevented by including a random salt value in the data to be hashed, along with the OTP value. The salt is included, in clear-text, in the list along with the corresponding verifier value.

The salt value is known to the verifier and, by our assumptions, to the attacker as well. This means that the salt value does not protect against post-download dictionary attacks. To protect against these attacks another random value is added, in addition to the salt, to the input to the hash.

The second random value, called the “pepper”, is not known to the local verifier. The local verifier must try all possible pepper values when verifying an entered OTP value. A method for determining the length of the random pepper value is given in Appendix A.

4 DAUTH Cryptography

DAUTH uses a number of cryptographic mechanisms to compute the OTP verifier values, transfer them to the local environment, and perform the local verification. The implementation details of these mechanisms may be varied according to the needs of the application.

4.1 Functions

To simplify our description of the DAUTH design, we reference a set of primitive functions. The details of these functions are not described here. The choice of the particular functions used is left to the implementer.

OTPGEN(S, T) computes the OTP value for secret S and index value T .

The index value may be a time, an event counter, or a random challenge.

RANDOM(min, max) generates a random integer r such that $\min \leq r \leq \max$.

HASH(a, b, c, ...) computes a 256-bit cryptographic hash value on the concatenation of its arguments.

E(P, K) encrypts plaintext value P , with 128-bit key K .

E⁻¹(C, K) decrypts ciphertext value C , with 128-bit key K .

BITS(x, m, n) selects a substring from x starting with m^{th} most significant bit and ending with the n^{th} most significant bit.

4.2 System Constants

We define a number of constant values used in computing the verifier values. These values may be varied on a per-implementation basis.

S_{max} = Maximum salt value

P_{max} = Maximum pepper value

Version = version number for this verifier format

4.3 Data Stored on the Server

The server must store a number of values specific to the user and the verifier. These values include:

S_U The long-term secret for the user.

S_V A secret value shared between the server and the local verifier. This value is used to authenticate the server to the local verifier.

S_{LOCAL} A secret value specific to the local verifier. It may be a local password, a file encryption key, or some other data from the local environment.

4.4 Local Secret Encryption

For each batch of verifiers we encrypt the local secret value with a randomly generated key.

$K_R = \text{RANDOM}(0, 2^{128})$

$$ES = E(S_{LOCAL}, K_S)$$

ES is delivered to the local environment.

4.5 Verifier Generation

For each index value T to be used in the desired offline time period, the server first computes the corresponding OTP value:

$$OTP_T = OTPGEN(S_U, T)$$

Then a random salt value (S_T) between 0 and the maximum salt is generated:

$$S_T = \text{RANDOM}(0, S_{max})$$

Next we generate a random pepper value (P_T) between 0 and the maximum allowed pepper value:

$$P_T = \text{RANDOM}(0, P_{max})$$

Next we compute a hash of the OTP, along with the salt and pepper values, the verifier secret, and a version number.

$$H = \text{HASH}(S_T \parallel P_T \parallel \text{OTP}(T) \parallel T \parallel S_V \parallel \text{Version})$$

The 256-bit hash value is divided into two 128-bit components, V_T and K_T :

$$V_T = \text{BITS}(H, 0, 127)$$

$$K_T = \text{BITS}(H, 128, 255)$$

The K_T value is used to encrypt the random key:

$$EK_{RT} = E(K_R, K_T)$$

All of the V_T , S_T , and EK_{RT} values are delivered to the local environment.

4.6 OTP Verification

In the local environment, the local verifier first finds the the S_T value corresponding to the current index. Next the entered OTP value is successively hashed with each of the allowed pepper values. The least significant 128 bits of each computed hash is compared with the V_T value for the same time. If the two values match, the most significant 128-bits of the hash are used to decrypt the random key. The random key is then used to decrypt the local secret. The entire process

is described in the pseudocode of Algorithm 1.

```

for  $p = 0$  to  $P_{max}$  do
  H = HASH( $S_T$  || p || U || T ||  $S_V$  || Version);
   $V'_T =$  BITS(H, 0, 127);
  if  $V'_T = V_T$  then
     $K_T =$  BITS(H, 128, 255);
     $K_R =$  D( $E_{K_{RT}}$ ,  $K_T$ );
     $S_{LOCAL} =$  D(ES,  $K_R$ );
  end
end

```

Algorithm 1: Offline OTP Verification Algorithm

5 DAUTH Implementation

This section describes the implementation of DAUTH in the RSA SecurID for Microsoft Windows product. The descriptions here focus on the implementation of the cryptography described in 4. Some of the implementation details regarding integration with the Windows environment have been omitted for the sake of clarity.

The RSA SecurID token is a time-based OTP generator, so the index values are sequential increments of Coordinated Universal Time (UTC.) The user's Windows password is used as the local secret and is presented to the operating system for local access after a successful OTP verification. The use of time as the index requires checking the V_T values for a range of times around the current time rather than just a single value, to allow for clock drift in the local environment.

5.1 Cryptographic Algorithms

The DAUTH implementation in the RSA SecurID for Windows product uses two key cryptographic algorithms, as listed below:

- The hash is computed using NIST's SHA-256 algorithm.
- Encryption and decryption uses RSA Security's RC5 algorithm.

5.2 Dayfile Structure

The local verifier requests a set of OTP verifiers for a specified time range, where the range may be as much as one month. The request is accepted only after a successful on-line authentication. The server computes the requested list of OTP values and stores the results in a structure called a dayfile. The dayfile consists of a single header record, followed by an entry record for each verifier value. The high-level organization of the dayfile is shown in Table 1.

Dayfile Header
Dayfile Entry for StartTime
Dayfile Entry for StartTime + 1
...
Dayfile Entry for EndTime - 1
Dayfile Entry for EndTime

Table 1. Dayfile Organization

5.3 Dayfile Header Record

The dayfile header contains the times corresponding to the first and last OTP verifier values in the dayfile. It contains the time interval between sequential OTP values, the maximum pepper value, the format version. It also contains the encrypted user password for local access. The fields of the header record are described in Table 2.

Field Name	Description
T_{START}	UTC Time corresponding to first entry
T_{END}	UTC Time corresponding to last entry
Interval	Time interval between OTP values
P_{max}	Maximum allowed pepper value
ES	Encrypted login password
Version	Format version number

Table 2. Dayfile Header Fields

5.4 Dayfile Entry Record

Each entry record contains the information needed to verify a single OTP value. The data include the corresponding UTC time, the random salt value, the hashed verifier value and the encrypted password decryption key. The fields in the entry record are described in Table 3.

Field Name	Description
T	UTC Time used to compute OTP
S_T	Salt value used in key derivation
V_T	Verifier value derived from OTP
EK_{RT}	Encrypted password decryption key

Table 3. Dayfile Entry Fields

5.5 Password Encryption

Before the Windows password is encrypted, it is copied into a password block structure. The password block is organized as shown in Table 4.

Length	Random Pad	Password	Random Pad	Hash
--------	------------	----------	------------	------

Table 4. Password Block Layout

The password block includes an initial length field, random padding, and a SHA-1 message digest value, in addition to the actual password. The random pad values prevent codebook attacks since the same password will never encrypt to the same ciphertext twice. The inclusion of the message digest allows verification of a successful decryption. Table 5 illustrates the layout of the password block.

Field Name	Description
Length	Total length of password block
RandomPad	8-byte random pad value
Password	Windows password
Hash	SHA-1 hash of the entire block

Table 5. Password Block Fields

The entire block is encrypted, using RC5 in CBC mode.

A Estimating DAUTH Security

The “pepper” bits are used to make off-line dictionary attacks against the contents of the dayfile more difficult. To determine the number of pepper bits to use, we must consider the processing power of the verifier, the processing power available to a hypothetical attacker, the entropy contained in a single OTP value, and the maximum allowed time to process a single verification. By combining these values we can estimate the security level for a specific pepper length.

A.1 Estimation Parameters

The parameters of the security estimation are shown in Table 6.

Parameter	Description
P_{max}	Maximum pepper value.
R_V	Processing rate of verifier, in hashes per second.
R_A	Processing rate of attacker, in hashes per second.
E_{OTP}	Entropy contained in a single Windows password.
W	Number of OTP values to test (Window).
A	Attacker advantage factor.

Table 6. Security Estimation Parameters

A.2 Computing OTP Entropy

The entropy contained in a single OTP value depends on the length and character set used. A typical OTP consists of a PIN which may be either numeric or alphanumeric, concatenated with a numeric code. The PIN is typically 4-6 digits or characters, while the code is usually 6 or 8 digits.

For numeric PINs with N digits and codes of length C digits the entropy is:

$$E_{OTP} = 10^{N+C}$$

For alphanumeric PINs with N characters (case-insensitive) and codes of length C digits the entropy is:

$$E_{OTP} = 36^N * 10^C$$

The computed entropy values for some common OTP configurations are shown in Table 7.

OTP Configuration	E_{OTP}
4 Digit PIN, 6 Digit Code	$2^{33.22}$
6 Digit PIN, 6 Digit Code	$2^{39.86}$
4 Character PIN, 6 Digit Code	$2^{40.61}$
6 Character PIN, 6 Digit Code	$2^{50.95}$
4 Digit PIN, 8 Digit Code	$2^{39.86}$
4 Character PIN, 8 Digit Code	$2^{47.26}$

Table 7. OTP Entropy Table

A.3 Computing Verification Time

To compute the expected verification time, we compute the expected number of pepper values to be tried, multiply by the number of OTP values to test, then

divide by the computation rate at the verifier. The expected number of OTP verifiers can be approximated by $W/2$ where W is the maximum range of index values to be tested per authentication attempt.

$$T_{VERIFY} = \frac{P_{max} * (W/2)}{R_V}$$

With a typical W value of 5, and an estimated verifier performance of 2^{15} SHA-256 hashes per second, we can compute the expected verification times, for differing numbers of pepper bits. The results are shown in Table 8.

P_{max}	T_{VERIFY} - seconds
2^{10}	0.1
2^{12}	0.3
2^{14}	1.3

Table 8. Typical Verification Times

A.4 Computing Attack Time

We assume that an attacker has significantly more computational resources than a single verifier. This advantage value, A , may be tuned to a specific threat model, but we use a value of 5000. Given the ever-growing quantities of CPU power available on-line, this is a conservative, but plausible estimate. The expected time for an attacker to perform a dictionary attack against a single OTP verifier is equal to the maximum pepper value times the OTP entropy divided by 2, divided by the computation rate of the verifier times the attacker's advantage factor.

$$T_{ATTACK} = \frac{P_{max} * (E_{OTP}/2)}{R_V * A}$$

If we use 10 pepper bits ($P_{max} = 2^{10}$), and an OTP with a 6 digit numeric PIN and a 6-digit code we compute, again using 2^{15} for R_V :

$$T_{ATTACK} = \frac{2^{10} * 2^{38.86}}{2^{15} * 5000} \approx 3,118,212 \text{ seconds} \approx 866 \text{ hours}$$

A.5 Results

Combining the results of the preceding sections we can compute the following table of verification and attack times for common OTP configurations and useful pepper sizes. The results are shown in Table 9.

OTP Configuration	P_{\max} - bits	T_{verify} - seconds	T_{attack} - hours
4 Digit PIN 6 Digit Code	10	0.1	9
	12	0.3	34
	14	1.3	138
6 Digit PIN 6 Digit Code	10	0.1	866
	12	0.3	3,465
	14	1.3	13,859
4 Character PIN 6 Digit Code	10	0.1	1,467
	12	0.3	5,867
	14	1.3	23,470
6 Character PIN 6 Digit Code	10	0.1	1,901,236
	12	0.3	7,604,945
	14	1.3	30,419,781
4 Digit PIN 8 Digit Code	10	0.1	872
	12	0.3	3,489
	14	1.3	13,955
4 Character PIN 8 Digit Code	10	0.1	148,336
	12	0.3	593,343
	14	1.3	2,373,371

Table 9. Security Estimates Table

A.6 Conclusions

What is the smallest attack time acceptable for a secure implementation? This is dependent on the details of the implementation. Ideally, the length of the expected attack should be longer than the expected lifetime of the local secret being protected in the dayfile. In the RSA SecurID for Windows product, this secret is a Windows password. A password change interval of 3 months is a common practice, so the attack time should be greater than this (2,160 hours).

Looking at table 9 we can draw a few useful conclusions about pepper sizes and OTP configurations:

- To make a secure implementation with the shortest values for both PIN length and code length requires a pepper value that makes verification take too long. This configuration is not recommended for use with DAUTH.
- Using alphanumeric PINs instead of numeric significantly increases the attack time while keeping the verification time low.
- Adding only two digits or characters to either the PIN or the code can move a usable implementation well above the required level of security. Making either value unreasonably large is not necessary.

It is also interesting to compare the attack time to that of a dictionary attack against a static Windows password. If we estimate the entropy in a Windows password as 2^{37} , a generous estimate, then even the shortest OTP configuration is significantly more secure against the same level of attack. If DAUTH is used whenever a password authentication is required, however, the password may be replaced with a significantly higher entropy string, since the user never needs to enter it. Thus, the security threshold should not be set higher than the current security level of the password.

RSA, SecurID, and RC5 are either registered trademarks or trademarks of RSA Security Inc. in the United States and/or other countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other products and services mentioned are trademarks of their respective owners.



RSA Security Inc.
www.rsasecurity.com

RSA Security Ireland Limited
www.rsasecurity.ie

RSA, RSA Security, *Confidence Inspired* and RCS are registered trademarks or trademarks of RSA Security Inc. in the United States and/or other countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the U.S. and/or other countries. All other products or services mentioned are trademarks of their respective owners.
©2005 RSA Security Inc. All rights reserved.

DAUTH TB 0305