IMPROVED LOOK-AHEAD RE-SYNCHRONIZATION WINDOW FOR HMAC-BASED ONE-TIME PASSWORD

Alireza Beikverdi, Ian K. T. Tan

Faculty of Computing and Informatics
Multimedia University, Cyberjaya 63100 Selangor, Malaysia
alireza.beikverdi@gmail.com, ian@mmu.edu.my

Keywords: One Time Password, HOTP, De-synchronization, Efficient, Re-synchronization Parameter.

Abstract

With the abundance of mobile wireless devices ranging from notebooks to smart phones, it has become convenient for One-Time Passwords (OTP) mechanism to be used for authentication. OTP are generated for single use purposes only and are generally generated on demand and have a limited usable shelf life. Upon usage, the password will be invalidated on both the client and the server side of the authentication system. A popular and standardized OTP system is the Hashed Message Authentication Code (HMAC) Based OTP (HOTP). In the HOTP system, the OTP is generated on the client side by firstly generating an encryption key that is derived from a shared secret key and incrementing a counter value. The final value generated is then truncated to the number of digits as per required by the OTP. On the server side, the same computation is generated and the OTP generated are compared for authentication. Signal interruptions in wireless environments may cause the need to regenerate a new OTP on the client side and hence causes the dynamic counter value to be desynchronized with the server. In the event that a mismatch occurs during the authentication process, the server increases the counter value within a look-ahead window for *s* times. This resynchronization parameter, *s*, determines the tolerance level of being desynchronized. However, larger *s* values (better tolerance) come with a trade-off of higher computational needs and can be a source of malicious attacks. This paper introduces an improved method to the HOTP standard in order to increase the computational efficiency for a larger *s* window. The introduced method doubles the *s* window size through negligible computational overheads. Furthermore, the method described in this paper can be easily implemented in the current standard implementation of the HOTP standards.

1 Introduction

The Internet has evolved from having protocols that utilize communications in plain text, such as the File Transfer Protocol (FTP) and Telnet, to highly encrypted communication protocols such as Secure Shell (SSH) and Hypertext Transfer Protocol Secure (HTTPS). The need for highly secured communications stems from the exploding number of users adopting online services such as social networking, e-commerce and online banking coupled with the proliferation of wireless devices to access the services. The sites that provide these services will require user authentications. The most tried and tested method for online authentication is through the use of a user login identity with their corresponding password. In a study by Florencio and Herley [1], it was found that many users will tend to use common passwords for a collection of sites and will in total have only a handful of passwords for the many sites that require authentication. In the same study, it was also found that most users will have relatively short password length where most have passwords length that are equal or less than 8 characters long. Using same passwords for various sites would expose the users to multiple site security breaches if any one of their sites were compromised while having short password length would make it vulnerable to attacks.

To provide for a more secure online environment, innovations in the area of applied cryptography and password authentication have been rapid. Authentication mechanisms such as single sign-on and the usage of one time passwords (OTP) have seen tremendous adoption rates in areas such as online email login authentication [2] and also Mobile Internet Banking. OTPs are dynamically generated for a single use. The use of OTP is able to address some of the problems of using the traditional fixed password mechanism. Due to the nature of OTP, users will not be having the same passwords for the different sites that they obtain their services from and the length of the password is of little significance with the use of OTP. The latter is because even if it is vulnerable to attacks, the password is only useful for a single use and hence any attacks on the password will be of no practical consequence.

One time passwords have been in used in highly secured environments such as governmental and military sites. Traditionally, users were provided with an offline token that will generate a new numeric personal identification number (PIN) in regular intervals. With the ubiquitous Internet, the generation of an OTP can be done by personal computers, mobile internet devices and also smart phones. These lead to the creation of standards such as the Time Based One Time Password [3] (TOTP) and the HMAC Based One Time Password (HOTP) [4].

The TOTP standard utilizes the system clock to generate the appropriate OTP on both the client and server side with an implementation of some tolerance for the client and server's clock to be slightly desynchronized. On the HOTP standard, it uses a counter value that was initially synchronized between the client and the server component. The client will increment the counter each time a new OTP is requested while the server will increment the counter each time it is asked to authenticate a client.

In the HOTP, if a client generates a new OTP and does not actually use the OTP for authentication, it will nevertheless increment the counter value on the client side. This is particularly evident in the wireless environment where signal disconnection can cause a generated OTP not to be used and smart phones or touch screen devices where users accidentally request for an OTP that will not used.

If there are no authentication requests on the server side, then the counter value on the server side will not be incremented. This causes the counter values between the client and the server to be desynchronized. When this happens, the authentication will fail and in order to address this issue, the HOTP standard provides for the server side to do a look ahead by incrementing the counter value. The number of times the server is allowed to look ahead is limited by the resynchronization value *s*. If a successful authentication happens within the resynchronization window, the counters will be resynchronized.

The trade off of a larger resynchronization window is higher computational needs and is a target for malicious attacks on an OTP authentication server in the form of a Denial of Service (DOS) attack. The contribution of this paper is in the extension of the resynchronization window with negligible overheads and hence will provide for a larger *s* value without additional risk on a DOS attack.

This paper is organized as follows; in section 2, we provide a brief background to the different OTP schemes that are currently being used and also their drawbacks. Section 3 discusses more specifically about the HOTP standard and its limitations. Our proposed method and efficiency discussion are provided in section 4 and 5 respectively. In section 6, we conclude.

2 Background

One time passwords have been proposed since 1981 for insecure communications [5]. One time password can be broadly divided into two categories based on the participating role of the server in the password generation. These are categorized as challenge-response and unidirectional OTP.

2.1 Challenge-Response

In a challenge-response system, the authentication server sends a token (the challenge) *i*, to the client when a request for authentication is received. Based on the token value *i*, coupled with a shared secret key, the client generate a unique one time password using specific hash function algorithm. This one time password is sent to the server for client authentication. The server will also generate a password from the same shared secret key and the *i* value using the same hash function. This is compared to the one time password it receives from the client and if it matches, the authentication is successful. The one time password can only be used once and if subsequent authentication is requested, even from the same client, the server will send a fresh challenge (token) to the client.

As an extension to the basic challenge-response system where both the client and server needs to have a shared secret key and the exchange of a token *i*, Jablon [6] proposed the use of the simple password key exchange method (SPEKE) which is similar to the much known and studied Diffie-Hellman key exchange method. This is as a replacement to the challenge token *i*, which can be vulnerable to spoofing attacks [7].

A major drawback of a challenge-response system is that there is a need for a direct communication between the authenticating server and the client. This limits the range of devices that can be used to generate the one time passwords. Standalone devices cannot be used and for connected devices, it increases the integration complexity for enterprise networks and connected devices.

2.2 Uni-directional

In a uni-directional method, the authenticating server and the client will both generate the passwords separately based on a counter value. This counter value needs to be synchronized during the setup of the authentication system between the client and the server. Thereafter, the server will not need to communicate with the client and the only communication is in the direction of the client to the server when the password is sent to the server for authentication. The server will compare the password sent by the client with the password it itself generated.

Uni-directional scheme is suitable for standalone devices such as OTP generating tokens. As it doesn't have any network connection, it is relatively safe from tempering and also interception attacks. With the popularity of this method, two standards were proposed, the Time-based One Time Password (TOTP) [3] and the HMAC-based One Time Password [4].

TOTP uses the system clock and increases the counter value by periodically. The period for the counter value to increase is to be agreed upon between the client and the authenticating server. It can be set based on needs and typically for highly secured environment, this counter value can be set to increase every minute.

In the HOTP scheme, the counter value is increased each time the client generates a new password with the token. On the server side, it will increase the value each time an authentication request is received. As the client token can

generate passwords without the user actually using the generated password to attempt an authentication request, it is evident that de-synchronization can occur. We will discuss the HOTP standards, how it attempts to limit the de-synchronization and the limitations in the next section.

3 HMAC-Based OTP

The generation of the OTP using the HOTP standard is based on a unique shared (with the authenticating server) symmetric key and an increasing counter. For the generation of the OTP, the HOTP standard uses the HMAC-SHA-1 algorithm [8].

The HMAC-SHA-1 function takes in two inputs, namely the shared symmetric key (K) and a counter value of 8 bytes long (C). It produces a 160 bit (20 byte) output (HS).

$$HS = \text{HMAC-SHA1}(K, C) \quad (1)$$

As the OTP is not expected to be 20 characters long, it is truncated. To compute an integer OTP of length n, the last 4 bits of the generated 20 bytes (HS) is used to define an offset number between 0 and 16. This number will be used as the offset to select the first byte of the consecutive 4 bytes from the 20 bytes generated earlier. The selected 4 bytes is then converted to a decimal number (D).

$$D = \text{Truncate}(HS) \quad (2)$$

The 4 bytes (32 bit), when converted to a decimal number, can be very large, namely of size 2^{32}. In order to obtain an integer of length n, the final OTP is computed using the modulus of 10^n of the generated decimal number.

$$\text{HOTP}(K, C) = D \bmod 10^n \quad (3)$$

As a short example of the above, we reiterate the example that was used in the Patent for HOTP [9]. Assuming that after the HMAC-SHA-1 function is completed (Equation 1), the 20 byte output is as follows (represented in hexadecimal and each byte separated by `|`).

```
1f|86|98|69|0e|02|ca|16|61|85|50|ef|7f|1
9|da|8e|94|5b|55|5a
```

The last byte has the value of `0x5a` where the value of last four bits is `0xa`. This is the offset number. Hexadecimal `0xa` is 10 in decimal. Using this offset value to determine the 4 bytes that we will use from the 20 byte output. The value of the four bytes starting from 10 is `0x50ef7f19`; this is 1357872921 in integer. In order to obtain a final HOTP of length 6 (n),

$$\text{HOTP} = 1357872921 \bmod 10^6 = 872921$$

Once the OTP has been generated, the counter value, C, will be incremented by one on the client side. When this OTP is used for the authentication request to the server, the server will generate the password using the same method. The server will then authenticate by checking is the received password and the password it generated match. Successful authentication is when the passwords match.

In a situation where a user generates passwords without ever using the generate passwords for authentication, the counter value on the client will increase while the counter value on the server will remain. When this client later request for authentication, the generated OTP will not match that of the servers and hence it will be unsuccessful. In order to address this issue, the HOTP standard provides for the resync-protocol, which involves a server look-ahead window of size S (resynchronization parameter).

In the resync-protocol, when the server is not able to match the passwords, it will proceed to regenerate a new OTP using an incremented counter. This is repeated until the password regenerated matches that of the password sent by the client or when it has been repeated for S times. If a match is found within the look-ahead window, the server counter will then be synchronized to the clients counter value. If on the other hand, the resynchronization parameter, S, is reached, the authentication is deemed as unsuccessful.

The resynchronization protocol requires significant resources and time for regenerating passwords continuously for S times. If the value of the resynchronization parameter is too large, this can be a target for denial of service (DOS) attacks.

Liao [10] et. al. attempted to address this counter de-synchronization issue by encrypting the counter value together with the OTP that is sent. It uses the shared secret key for the encryption of the counter value on the client side and the decryption of the counter value on the server side. However, their method relies solely on the shared secret key, which once broken, will render their method as good as any shared secret key encryption. The method has eliminated the second important input other than the shared symmetric key which is central to the overall OTP setup.

Our approach is not to totally eliminate the counter de-synchronization issue but to extend (double) the resynchronization parameter, S, without the corresponding increase in the resource or time required for the regeneration of the passwords.

4 Proposed Method

The basis of our proposed method is for the server to determine whether the counter value is odd or even. The rest of this paper will use the word parity to refer to the oddness and evenness of the number. Once it is determined, the counter value on the server side can be incremented by 2 instead of 1. This would mean that for the same S value, our proposed method only needs to generate half as many passwords before it reaches the resynchronization parameter limit.

In the standard HOTP, the client will generate the password. The user will then use this password to request for authentication on the server side. Ignoring any encryption to the password during the transmission of the password from client to server, the server will store the received password for processing. The following pseudo code provides for the server authentication process.

```
Data: s is the resynchronization parameter
Input: secret key K, counter C
Output: Password validation from server
i ← 0
while i ≤ s do
   svr_pass ← HOTP(K,C)
   if svr_pass = clnt_pass then
      login successful
   else
      if i ≥ s then
         reset svr_counter
         login failed
      end if
      else
         svr_counter ← svr_counter + 1
   end if
      i ← i + 1
end
```

Figure 1: Pseudo-Code for HOTP Server Authentication

Our proposed method will utilize the parity of the password to reflect the parity of the counter that is used to generate it. This will involve modification to both the client and server side of the OTP system.

4.1 Client-Side Password Generator

The generated password, which is an integer number, will need to have the same parity to that of the counter value. If it is not, then the password is simply incremented by 1 to ensure that it has the same parity.

```
clnt_pass ← HOTP(K,C)
if clnt_pass mod 2 ≠ clnt_counter mod 2
        clnt_pass ← clnt_pass + 1
end if
```

Figure 2: Pseudo-Code for Proposed HOTP Generation

This is the password that is sent to the server to request for authentication.

4.2 Server-Side Password Generator for Comparison

The server, upon receiving the password will check for the parity with its own counter. If it is not the same, then it will know that the counter values are desynchronized. It will immediately increase its counter value by 1 and hence will have the same parity value as the password received.

```
Data: s is the resynchronization parameter
Input: secret key K, counter C
Output: Password validation from server in the efficient
scheme
if clnt_pass mod 2 ≠ svr_counter mod 2
   svr_counter ← svr_counter + 1
end if
i ← 0
while i ≤ s do
   svr_pass ← HOTP(K,C)
   if clnt_pass mod 2 ≠ svr_counter mod 2
      svr_pass ← svr_pass + 1
   end if
   if svr_pass = clnt_pass
      login successful
   else
      if i ≥ s then
         reset svr_counter
         login failed
      end if
      else
         svr_counter ← svr_counter + 2
      end if
         i ← i + 1
   end
```

Figure 3: Pseudo-Code for Proposed HOTP Server Authentication

The server will then proceed to generate the corresponding password for comparison with the received password from the client. An additional process is also required after the generation of the password prior to matching the passwords. As per the client side, the password generated has to be the same parity as the counter value. If it is not, the password received from the client will have been incremented by 1. The server needs to do the same computation prior to comparing the passwords. The rest of the process is the same with the exception that instead of incrementing by 1 per iteration within the look-ahead window, the server counter will be incremented by 2. This will ensure that the parity of the password is correct and hence save on unnecessary iteration cycles.

With the additional few operations in the client and server side, this proposed method is more efficient when there is de-synchronization between the client and server counters.

5 Algorithm Analysis

Our proposed algorithm incorporates the parity of the counter value into the generated OTP. In other words, the OTP will have the same parity as the counter value. By knowing the parity of the counter, the number of checks can be reduced by half through incrementing the counter by two (instead of one) for each checking procedure. From an alternative angle, it would mean that for the same number of checks, the re-synchronization look-ahead window can be doubled.

5.1 Overheads and Complexity

On additional overheads, the client-side OTP generation introduces a check for parity of the generated numeric password and increment it accordingly if it is not.

On the server side, the code complexity of the standard HOTP (Figure 1) and our proposed enhancement (Figure 3) is governed by the `for` loop. The additional code within the `for` loop iteration is independent of the iteration variable, `s`, and hence does not increase the complexity of the code.

The additional lines of code are basic conditional statements with simple integer value additions. These will not incur any significant penalty on the program execution.

5.2 Weakened Security

On the concerns of weakened security due to the possible knowledge of the parity of the counter, this will only assist in brute force attacks as the maximum number of brute force attempts will be halved.

The overall security of the proposed system is still bound by the symmetric encryption of the shared secret key together with the dynamic counter value. Due to the nature of OTP being sheltered from replay if a spoofing attempt is successful, the knowledge of the parity of the spoofed password will still require a brute force attack to determine the symmetric key. This is deemed impractical if the shelf life of the OTP is short.

6 Conclusion

This paper proposes an efficient, easy to implement method in order to increase the resynchronization parameter in HMAC-based one time password.

In the usage of HOTP, it is highly likely that there will be occasional de-synchronization between the client and the server especially in a wireless environment due to signal noise and also the wireless mobile device characteristics. The resynchronization parameter, *S*, provides for some resynchronization possibilities but is limited due to possible DOS attacks on the system.

With our proposed method, we can increase the resynchronization parameter, *S*, without additional increase in resources. This provides for a larger de-synchronization error that can be corrected.

Our proposed method can be easily incorporated into any implementation of the standard HOTP. As our future work, we will be integrating this to current HOTP implementation on smart phones (as the OTP generating token). Our future investigation will include an empirical measurement of the overheads of our proposed method with the analysis.

References

[1] D. Florencio, C. Herley. “A large-scale study of web password habits,” In *Proceedings of the 16th international conference on World Wide Web* (WWW '07). ACM, New York, NY, USA, pp. 657-666, (2007)

[2] D. Raywood. “Google adds two-factor authentication to Gmail via SMS one time password”, SC Magazine, (September 21, 2010). [online].

[3] D. M'Raihi, S. Machani, M. Pei, J. Rydell. “RFC 6238: TOTP: Time Based One Time Password Algorithm”, (May 2011).

[4] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, O. Ranen. “RFC 4226: HOTP: An HMAC Based One Time Password Algorithm”, (December 2005).

[5] L. Lamport. “Password authentication with insecure communication”, Communications of the ACM volume 24 (11), pp. 770-772 (1981).

[6] D.P. Jablon. “Strong password-only authenticated key exchange”, ACM SIGCOMM Computer Communication Review, volume 26 (5), pp. 5-26, (October 1996).

[7] N. Haller. “The S/KEY one-time password system”, In Proceedings of the Internet Society Symposium on Network and Distributed Systems, pp. 151-157, (1994).

[8] H. Krawczyk, R. Canetti, M. Bellare. “RFC2104: HMAC: Keyed-hashing for message authentication”, (February 1997).

[9] N. Popp, D. M'Raihi, L. Hart, "One time password," U.S. Patent 8087074, December 27, 2011.

[10] S. Liao, Q. Zhang, C. Chen, Y. Dai, Y. “A unidirectional one-time password authentication scheme without counter desynchronization”, Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on , volume 4, pp.361-364, (8-9 August 2009).