



OTP (One Time Password) Demystified

**orouit**, 26 May 2013

CPOL



4.86 (32 votes)

Rate this:

This article shows how an OTP generator works.

[Download source - 18.6 KB](#)

Introduction

At the beginning of 2004, I was working with a small team of Gemplus on the EAP-SIM authentication protocol. As we were a bit ahead of the market, our team was reassigned to work with a team of Verisign on a new authentication method: OTP or One Time Password.

At this time, the existing one time password was a token from RSA that was using a clock to synchronize the passwords.

The lab of Versign came with a very simple but I should say very smart concept. The OTP that you may be using with your bank or Google was born.

This is this algorithm and authentication method I describe in the following two articles. In this article, I present a complete code of the OTP generator. It is very similar to the Javacard Applet I wrote in 2004 when I started to work on this concept with the Versign labs.

The One Time Password Generator

This OTP is based on the very popular algorithm HMAC SHA. The HMAC SHA is an algorithm generally used to perform authentication by challenge response. It is not an encryption algorithm but a hashing algorithm that transforms a set of bytes to another set of bytes. This algorithm is not reversible which means that you cannot use the result to go back to the source.

A HMAC SHA uses a key to transform an input array of bytes. The key is the secret that must never be accessible to a hacker and the input is the challenge. This means that OTP is a challenge response authentication.

The secret key must be 20 bytes at least; the challenge is usually a counter of 8 bytes which leaves quite some time before the value is exhausted.

The algorithm takes the 20 bytes key and the 8 bytes counter to create a 8 digits number. This means that there will obviously be duplicates during the life time of the OTP generator but this doesn't matter as no duplicate can occur consecutively and an OTP is only valid for a couple of minutes.

Why is the OTP a very strong authentication method?

There are few reasons why this is a very strong method.

- The key is 20 digits
- A password is a couple counter/password, only valid once and a very short time
- The algorithm that generates each password is not reversible
- With an OTP token, the key is hardware protected
- If the OTP is received on your phone, the key always stays at the server

Those few characteristics make the OTP a strong authentication protocol. The weakness in an authentication is usually the human factor. It is difficult to remember many complex passwords, so users often use the same one all across the internet and not really a strong one. With an OTP, you don't have to remember a password, the most you would have to remember would be PIN code (4 to 8 digits) if the OTP token is PIN protected. In the case of an OTP sent by a mobile phone, it is protected by your phone security. A PIN is short but you can't generally try it more than 3 times before the token is locked.

The weakness of an OTP if there is one, is the media used to generate or receive the OTP. If the user loses it, then the authentication could be compromised. A possible solution would be to protect this device with a biometric credential, making it virtually totally safe.

The code of the OTP generator follows:

Hide Shrink  Copy Code

```
public class OTP
{
    public const int SECRET_LENGTH = 20;
    private const string
MSG_SECRETLENGTH = "Secret must be at least 20 bytes",
MSG_COUNTER_MINVALUE = "Counter min value is 1";
```

```

public OTP()
{
}

private static int[] dd = new int[10] { 0, 2, 4, 6, 8, 1, 3, 5, 7, 9 };

private byte[] secretKey = new byte[SECRET_LENGTH]
{
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41, 0x42, 0x43
};

private ulong counter = 0x0000000000000001;

private static int checksum(int Code_Digits)
{
int d1 = (Code_Digits/1000000) % 10;
int d2 = (Code_Digits/100000) % 10;
int d3 = (Code_Digits/10000) % 10;
int d4 = (Code_Digits/1000) % 10;
int d5 = (Code_Digits/100) % 10;
int d6 = (Code_Digits/10) % 10;
int d7 = Code_Digits % 10;
return (10 - ((dd[d1]+d2+dd[d3]+d4+dd[d5]+d6+dd[d7]) % 10) ) % 10;
}

/// <summary>
/// Formats the OTP. This is the OTP algorithm.
/// </summary>
/// <param name="hmac">HMAC value</param>
/// <returns>8 digits OTP</returns>
private static string FormatOTP(byte[] hmac)
{
int offset = hmac[19] & 0xf ;
int bin_code = (hmac[offset] & 0x7f) << 24
| (hmac[offset+1] & 0xff) << 16
| (hmac[offset+2] & 0xff) << 8
| (hmac[offset+3] & 0xff) ;
int Code_Digits = bin_code % 10000000;
int csum = checksum(Code_Digits);
int OTP = Code_Digits * 10 + csum;

return string.Format("{0:d08}", OTP);
}

public byte[] CounterArray
{
get
{
return BitConverter.GetBytes(counter);
}
}

```

```

set
{
    counter = BitConverter.ToUInt64(value, 0);
}
}

/// <summary>
/// Sets the OTP secret
/// </summary>
public byte[] Secret
{
set
{
    if (value.Length < SECRET_LENGTH)
    {
throw new Exception(MSG_SECRETLENGTH);
    }

    secretKey = value;
}
}

/// <summary>
/// Gets the current OTP value
/// </summary>
/// <returns>8 digits OTP</returns>
public string GetCurrentOTP()
{
HmacSha1 hmacSha1 = new HmacSha1();

hmacSha1.Init(secretKey);
hmacSha1.Update(CounterArray);

byte[] hmac_result = hmacSha1.Final();

return FormatOTP(hmac_result);
}

/// <summary>
/// Gets the next OTP value
/// </summary>
/// <returns>8 digits OTP</returns>
public string GetNextOTP()
{
// increment the counter
++counter;

return GetCurrentOTP();
}

/// <summary>
/// Gets/sets the counter value

```

```

/// </summary>
public ulong Counter
{
    get
    {
        return counter;
    }

    set
    {
        counter = value;
    }
}

```

The methods **FormatOTP()** and **checksum()** are the heart of the OTP algorithm. Those methods transform the result of the hmacsha into an 8 digits OTP.

The attached code also contains an implementation of the HMAC SHA algorithm. It is of course possible to use the standard hmacsha of the .NET Framework but the code I provide in fact used a demo in a prototype of smart card that was running a .NET CLR. At the time I wrote this code, the cryptography namespace was not yet implemented by the card.

This way, you can also see how a hmacsha algorithm is implemented.

The OTP Server and Authentication Protocol

There are usually 2 ways to perform an authentication with an OTP. I'm going to describe the real case of an authentication to an online banking site. I just want to be explicit with something. You cannot use what I'm going to describe in this post to hack into a banking site! On the contrary after reading this you should understand why using an OTP as a second factor authentication is extremely secure.

The OTP by itself is already very secure for at least the 2 following reasons:

- You can't play it twice
- You can't go back to the source.

The second characteristic is very important in term of security. An OTP depends on 2 parameters:

- A secret key
- A counter

Even if a hacker intercepts millions of OTP the algorithm is not reversible which means that even if you know the key you can't go back to the counter that was used to generate the OTP. So without the key and the counter, it is virtually impossible even with millions of OTP to find a pattern to guess the key and the current counter value.

Like many security protocols, the strength of the OTP is given by the quality of the cryptography algorithm used, in this

case HMACSHA1 which is a proven challenge response algorithm. An other HMAC algorithm can be used in place of HMACSHA as encryption algorithm have to become stronger when CPU power is increasing. This can be done by increasing the size of the key or by redesigning the algorithm itself.

OTP are usually used to perform authentication or to verify a transaction with a credit card. In the case of a transaction an OTP is sent to the mobile phone of the user, for an authentication if is possible to use either a secure token or to request an OTP to be send to the user phone.

Using an OTP sent to a phone

This is usually the authentication method used when a transaction is verified with an OTP. The bank system sends you an OTP and you then have few minutes to enter this OTP. This mechanism doesn't need any synchronization process as the OTP is originally generated by the server and send to a third party device. The server expects that you type the correct OTP within generally 2 mns. If you fail to do it, you just ask a new OTP and then enter it within the given time.

When a system supports both authentication methods, it means that the back-end has 2 different keys and counters; one pair for the OTP token and one pair for the OTP transmitted by SMS.

Using an OTP token

The original product I worked on when we implemented one of the first versions of the OTP in a Javacard was using an OTP token with a screen or a mobile phone with a card applet to generate the OTP. In this model both the server and the authentication token have to generate an OTP that must be synchronized.

The process is the following: The user generates an OTP with his token, type it and press OK. The server receives the OTP generated by the token, it increments the counter and generates a new OTP.

This is where there is a possible synchronization issue.

Synchronization issues

If the user enters the correct OTP, then the server when it increments the counter and calculate the OTP, the authentication will be successful.

Now there could be few scenarios that could lead to a desynchronization of the server counter and the authentication mechanism won't work. In some cases it could be possible to resynchronize automatically the counter but in some cases the user would have to resynchronize the server counter using a specific procedure.

Few scenarios of desynchronization could arise:

1. The user accidentally press the generate button of his token and doesn't perform an authentication.
In this case the counter of the token would be ahead of the server counter by few steps.
2. The user enters an OTP without generating it from the token. In this case the counter of the server would be ahead of the token counter.
3. The user generates an OTP with the token but types a wrong OTP.

If the OTP given by the user doesn't match the one of the server, the server can try to auto-resynchronize itself by trying few counters around the expected counter. In our server we would use 10 values around the nominal counter value. If the synchronization cannot be done, the server would retain the current counter value in order not to desynchronize the server further.

However the server would have to implement a strategy to inform that the server and token are totally desynchronized and a manual synchronization must be performed.

Manual synchronization process

The server can propose a manual synchronization process to the user. The OTP numbers are only 8 digits generated by the hash of 8 bytes counter and formatting a 20 bytes result. This means that it is possible to get twice the same OTP for 2 different counter values. So attempting synchronization with only one OTP value is not reliable. A manual resynchronization process needs the user to enter 2 consecutive OTP, and then the server can try to find the requested sequence as the probability to get the same sequence of 2 OTPs for different counter values is extremely low if not zero.

Getting the Source

You can get the source code of the project from the ZIP files attached to the article or you can follow it on github where it will be updated regularly as this is a public repository.

Points of Interest

OTP is a popular and quite simple authentication method; I hope those articles will help you understand how it works behind the scenes.

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

Share



About the Author



orouit



Architect Consistel - Singapore
Singapore

Software Architect, COM, .NET and Smartcard based security specialist.

I've been working in the software industry since I graduated in Electrical and Electronics Engineering. I chose software because I preferred digital to analog.

I started to program with 6802 machine code and evolved to the current .NET technologies... that was a long way.

For more than 20 years I have always worked in technical positions as I simply like to get my hands dirty and crack my brain when things don't go right!

After 12 years in the smart card industry I can claim a strong knowledge in security solutions based on those really small computers!

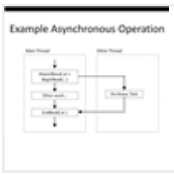
I've been back into business to design the licensing system for the enterprise solution for Consistel using a .NET smart card (yes they can run .NET CLR!)

I'm currently designing a micro-payment solution using the NXP DESFire EV1 with the ACSO6 SAM of ACS. I can then add a full proficient expertise on those systems and NFC payments.

This technology being under strict NDA by NXP I cannot publish any related article about it, however I can provide professional consulting for it.

You can contact me for professional matter by using the forum or via my [LinkedIn profile](#).

You may also be interested in...



Multithreading Demystified



Fitting Static Code Analysis Into Continuous Integration



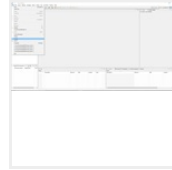
Demystifying OAuth



How-To Intel® IoT Technology Code Samples: Access control in C++



Static Keyword Demystified



How-To Intel® IoT Technology Code Samples: Robot arm in C++

Comments and Discussions

You must [Sign In](#) to use this message board.

Search Comments

Go

☐ Profile popups

Spacing

Relaxed



Layout

Normal



Per page





























































50


















Update

First Prev Next

Onetime passwords in PHP	Member 11073352	31-Oct-15 8:49
Getting the same OTP same time	farhan azhar	20-May-15 3:43
Regarding OTP Generator	Member 11569305	31-Mar-15 2:26
How to run this program?	hate code	2-Nov-14 23:10
Re: How to run this program?	orouit	10-Nov-14 23:54
Re: How to run this program?	B K Singh 2187	14-Nov-14 2:04

 Re: How to run this program? 	 hate code	18-Nov-14 17:30
 Re: How to run this program? 	 orouit	2-Mar-15 15:14
 checksum 	 rana faiz	28-Oct-14 4:18
 Re: checksum 	 orouit	11-Nov-14 0:10
 My vote of 5 	 Sibeesh KV	7-Oct-14 20:15
 OTP in WordPress using PHP 	 Member 11073352	10-Sep-14 12:54
 Re: OTP in WordPress using PHP 	 orouit	17-Sep-14 19:04
 Can I generate a OTP without use token 	 rana faiz	31-Aug-14 10:36
 Re: Can I generate a OTP without use token 	 orouit	9-Sep-14 4:39
 Re: Can I generate a OTP without use token 	 rana faiz	21-Oct-14 7:43
 Re: Can I generate a OTP without use token 	 orouit	22-Oct-14 22:22
 Thanks for a great article! 	 Zoe Kat	15-May-14 3:30
 Time generated OTP 	 Member 10698183	25-Mar-14 6:14
 OTP,SHA-1,Hmacsha1 classes 	 Member 10424362	5-Jan-14 9:49
 Re: OTP,SHA-1,Hmacsha1 classes 	 orouit	14-Jan-14 21:39
 Re: OTP,SHA-1,Hmacsha1 classes 	 Member 10424362	14-Jan-14 22:03
 question.. 	 Member 10424362	19-Dec-13 2:07
 Re: question.. 	 orouit	14-Jan-14 21:36
 Nice article! 	 valagappan	16-Sep-13 10:14
 Re: Nice article! 	 orouit	14-Oct-13 17:45

 From 8 digit to 6 digit 	 Try_Catch	18-Jun-13 17:15
 My vote of 5 	 Mihai MOGA	13-Jun-13 21:40
 My vote of 5 	 Dr Bob	2-Jun-13 9:54
 Right on time and a question 	 Frankidoze	15-May-13 9:39
 Re: Right on time and a question 	 orouit	16-May-13 7:33
<div> Last Visit: 31-Dec-99 19:00 Last Update: 20-Apr-16 0:37 <div>Refresh</div> <div>1</div> </div>		

 General
 News
 Suggestion
 Question
 Bug
 Answer
 Joke
 Praise
 Rant
 Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

[Permalink](#) | [Advertise](#) | [Privacy](#) | [Terms of Use](#) | [Mobile](#)
Web02 | 2.8.160418.1 | Last Updated 26 May 2013

 Select Language | Layout: [fixed](#) | [fluid](#)

Article Copyright 2013 by orouit
Everything else Copyright © [CodeProject](#), 1999-2016