PyOTP - The Python One-Time Password Library

PyOTP is a Python library for generating and verifying one-time passwords. It can be used to implement two-factor (2FA) or multi-factor (MFA) authentication methods in web applications and in other systems that require users to log in.

Open MFA standards are defined in RFC 4226 (HOTP: An HMAC-Based One-Time Password Algorithm) and in RFC 6238 (TOTP: Time-Based One-Time Password Algorithm). PyOTP implements server-side support for both of these standards. Client-side support can be enabled by sending authentication codes to users over SMS or email (HOTP) or, for TOTP, by instructing users to use Google Authenticator, Authy, or another compatible app. Users can set up auth tokens in their apps easily by using their phone camera to scan otpauth:// QR codes provided by PyOTP.

Implementers should read and follow the HOTP security requirements and TOTP security considerations sections of the relevant RFCs. At minimum, application implementers should follow this checklist:

- Ensure transport confidentiality by using HTTPS
- Ensure HOTP/TOTP secret confidentiality by storing secrets in a controlled access database
- Deny replay attacks by rejecting one-time passwords that have been used by the client (this requires storing the most recently authenticated timestamp, OTP, or hash of the OTP in your database, and rejecting the OTP when a match is seen)
- Throttle brute-force attacks against your application's login functionality
- When implementing a "greenfield" application, consider supporting FIDO U2F/WebAuthn in addition to HOTP/TOTP. U2F uses asymmetric cryptography to avoid using a shared secret design, which strengthens your MFA solution against server-side attacks. Hardware U2F also sequesters the client secret in a dedicated single-purpose device, which strengthens your clients against client-side attacks. And by automating scoping of credentials to relying party IDs (application origin/domain names), U2F adds protection against phishing attacks. One implementation of FIDO U2F/WebAuthn is PyOTP's sister project, PyWARP.

We also recommend that implementers read the OWASP Authentication Cheat Sheet and NIST SP 800-63-3: Digital Authentication Guideline for a high level overview of authentication best practices.

Quick overview of using One Time Passwords on your phone

- OTPs involve a shared secret, stored both on the phone and the server
- OTPs can be generated on a phone without internet connectivity
- OTPs should always be used as a second factor of authentication (if your phone is lost, you account is still secured with a password)
- Google Authenticator and other OTP client apps allow you to store multiple OTP secrets and provision those using a QR Code

Installation

```
pip install pyotp
```

Usage

Time-based OTPs

```
totp = pyotp.TOTP('base32secret3232')
totp.now() # => '492039'

# OTP verified for current time
totp.verify('492039') # => True
time.sleep(30)
totp.verify('492039') # => False
```

Counter-based OTPs

```
hotp = pyotp.HOTP('base32secret3232')
hotp.at(0) # => '260182'
hotp.at(1) # => '055283'
hotp.at(1401) # => '316439'

# OTP verified with a counter
hotp.verify('316439', 1401) # => True
hotp.verify('316439', 1402) # => False
```

Generating a Secret Key

A helper function is provided to generate a 16 character base32 secret, compatible with Google Authenticator and other OTP apps:

```
pyotp.random_base32()
```

Some applications want the secret key to be formatted as a hex-encoded string:

```
pyotp.random_hex()  # returns a 32-character hex-encoded secret
```

Google Authenticator Compatible

PyOTP works with the Google Authenticator iPhone and Android app, as well as other OTP apps like Authy. PyOTP includes the ability to generate provisioning URIs for use with the QR Code scanner built into these MFA client apps:

```
pyotp.totp.TOTP('JBSWY3DPEHPK3PXP').provisioning_uri(name='alice@google.com', issuer_name='Secure App')

>>> 'otpauth://totp/Secure%20App:alice%40google.com?secret=JBSWY3DPEHPK3PXP&issuer=Secure%20App'

pyotp.hotp.HOTP('JBSWY3DPEHPK3PXP').provisioning_uri(name="alice@google.com", issuer_name="Secure App", initial_count=0)

>>> 'otpauth://hotp/Secure%20App:alice%40google.com?
secret=JBSWY3DPEHPK3PXP&issuer=Secure%20App&counter=0'
```

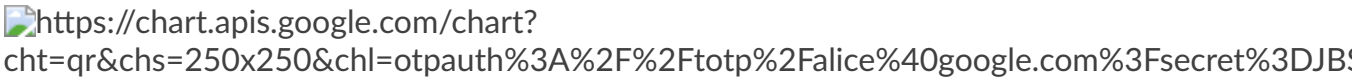
This URL can then be rendered as a QR Code (for example, using https://github.com/neocotic/qrious) which can then be scanned and added to the users list of OTP credentials.

Parsing these URLs is also supported:

```
pyotp.parse_uri('otpauth://totp/Secure%20App:alice%40google.com?
secret=JBSWY3DPEHPK3PXP&issuer=Secure%20App')

>>> <pyotp.totp.TOTP object at 0xFFFFFFFF>

pyotp.parse_uri('otpauth://hotp/Secure%20App:alice%40google.com?
secret=JBSWY3DPEHPK3PXP&issuer=Secure%20App&counter=0'

>>> <pyotp.totp.HOTP object at 0xFFFFFFFF>
```

Working example

Scan the following barcode with your phone's OTP app (e.g. Google Authenticator):



Now run the following and compare the output:

```
import pyotp
totp = pyotp.TOTP("JBSWY3DPEHPK3PXP")
print("Current OTP:", totp.now())
```

Links

- Project home page (GitHub)
- Documentation (Read the Docs)
- Package distribution (PyPI)
- Change log
- RFC 4226: HOTP: An HMAC-Based One-Time Password
- RFC 6238: TOTP: Time-Based One-Time Password Algorithm
- ROTP - Original Ruby OTP library by Mark Percival
- OTPHP - PHP port of ROTP by Le Lag
- OWASP Authentication Cheat Sheet
- NIST SP 800-63-3: Digital Authentication Guideline

For new applications:

- WebAuthn
- PyWARP

API documentation

Table of Contents

- Index
- Module Index
- Search Page