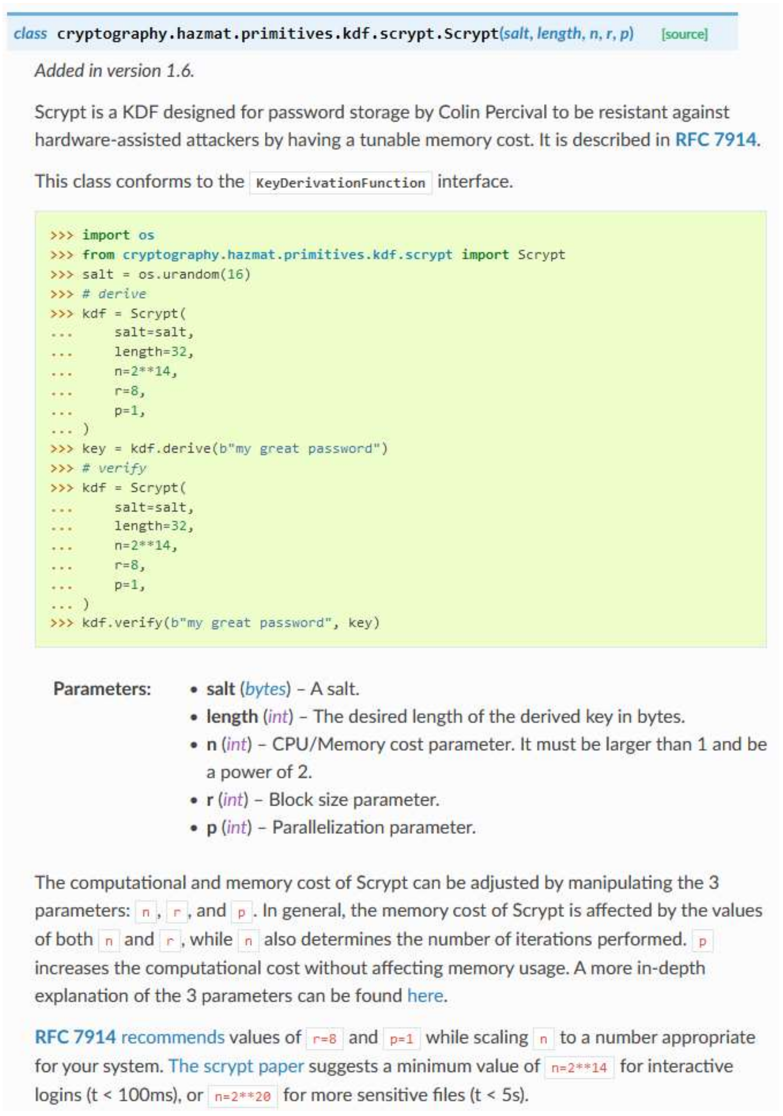
COSC 3364 – Principles of Cybersecurity
Lab 04
Two-Factor Authenticator

1. Develop a two-factor authenticator program to perform credential creation and login verification. Valid usernames will be only lowercase letters and numbers, between four to twelve characters in length. Valid passwords will be comprised of at least one uppercase letter, one lowercase letter, one number and one special character with a minimum of eight characters in length. During credential creation the password will be required to confirm the password with a duplicate password entry. The program will store the username, hashed password, and salt value in respective lists. The key derivation function for password storage will be Scrypt. Upon account creation the program will provide a TOTP token required for login access.

OUTPUT Screenshots below

Scrypt



class **cryptography.hazmat.primitives.kdf.scrypt.Scrypt**(*salt, length, n, r, p*) [source]

Added in version 1.6.

Scrypt is a KDF designed for password storage by Colin Percival to be resistant against hardware-assisted attackers by having a tunable memory cost. It is described in RFC 7914.

This class conforms to the `KeyDerivationFunction` interface.

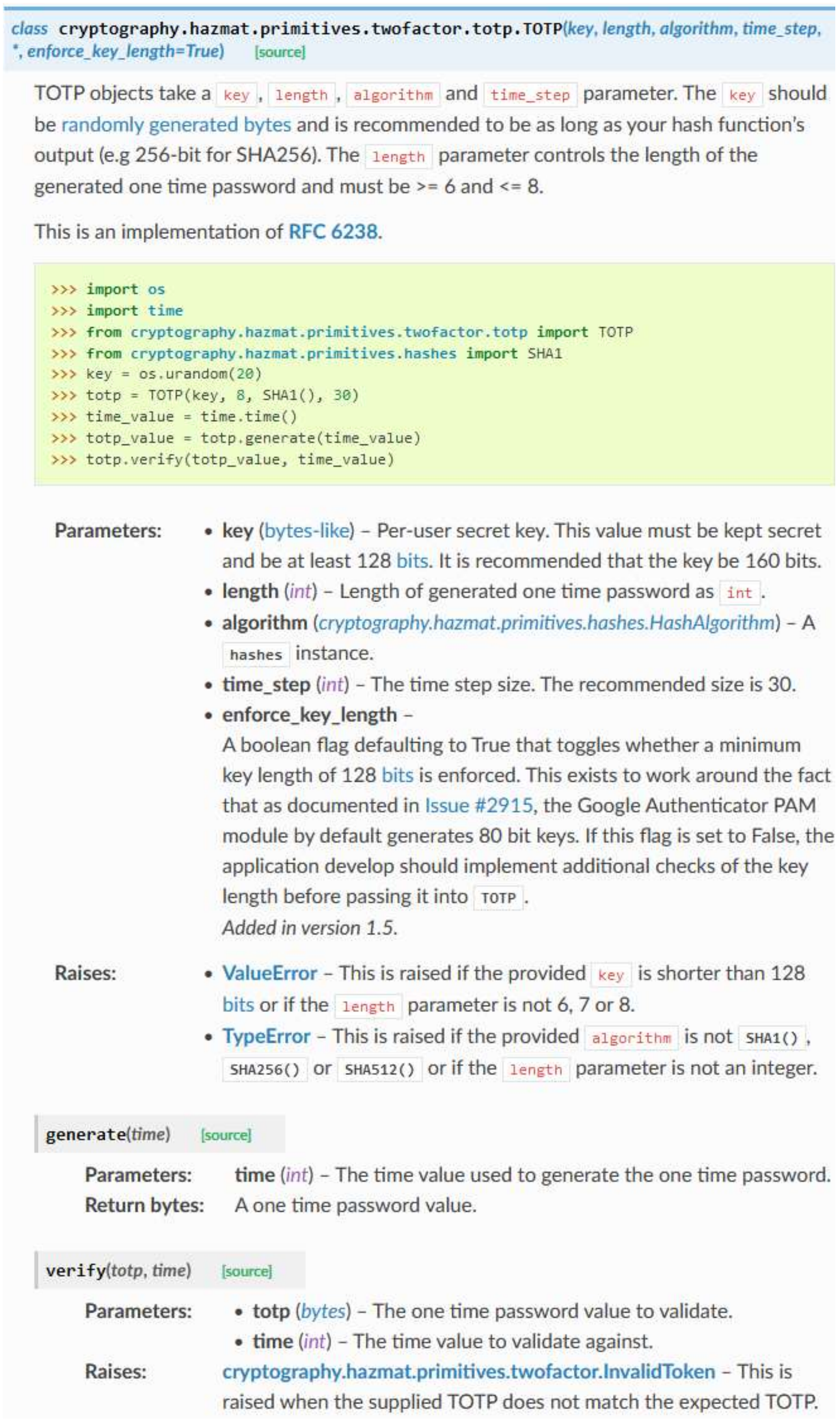
```
>>> import os
>>> from cryptography.hazmat.primitives.kdf.scrypt import Scrypt
>>> salt = os.urandom(16)
>>> # derive
>>> kdf = Scrypt(
...     salt=salt,
...     length=32,
...     n=2**14,
...     r=8,
...     p=1,
... )
>>> key = kdf.derive(b"my great password")
>>> # verify
>>> kdf = Scrypt(
...     salt=salt,
...     length=32,
...     n=2**14,
...     r=8,
...     p=1,
... )
>>> kdf.verify(b"my great password", key)
```

Parameters:
- **salt** (*bytes*) – A salt.
- **length** (*int*) – The desired length of the derived key in bytes.
- **n** (*int*) – CPU/Memory cost parameter. It must be larger than 1 and be a power of 2.
- **r** (*int*) – Block size parameter.
- **p** (*int*) – Parallelization parameter.

The computational and memory cost of Scrypt can be adjusted by manipulating the 3 parameters: `n`, `r`, and `p`. In general, the memory cost of Scrypt is affected by the values of both `n` and `r`, while `n` also determines the number of iterations performed. `p` increases the computational cost without affecting memory usage. A more in-depth explanation of the 3 parameters can be found here.

RFC 7914 recommends values of `r=8` and `p=1` while scaling `n` to a number appropriate for your system. The scrypt paper suggests a minimum value of `n=2**14` for interactive logins (t < 100ms), or `n=2**20` for more sensitive files (t < 5s).

TOTP



class **cryptography.hazmat.primitives.twofactor.totp.TOTP**(*key, length, algorithm, time_step, *, enforce_key_length=True*) [source]

TOTP objects take a `key`, `length`, `algorithm` and `time_step` parameter. The `key` should be randomly generated bytes and is recommended to be as long as your hash function's output (e.g 256-bit for SHA256). The `length` parameter controls the length of the generated one time password and must be >= 6 and <= 8.

This is an implementation of **RFC 6238**.

```
>>> import os
>>> import time
>>> from cryptography.hazmat.primitives.twofactor.totp import TOTP
>>> from cryptography.hazmat.primitives.hashes import SHA1
>>> key = os.urandom(20)
>>> totp = TOTP(key, 8, SHA1(), 30)
>>> time_value = time.time()
>>> totp_value = totp.generate(time_value)
>>> totp.verify(totp_value, time_value)
```

Parameters:

- **key** (bytes-like) – Per-user secret key. This value must be kept secret and be at least 128 bits. It is recommended that the key be 160 bits.
- **length** (*int*) – Length of generated one time password as `int`.
- **algorithm** (*cryptography.hazmat.primitives.hashes.HashAlgorithm*) – A `hashes` instance.
- **time_step** (*int*) – The time step size. The recommended size is 30.
- **enforce_key_length** –
  A boolean flag defaulting to True that toggles whether a minimum key length of 128 bits is enforced. This exists to work around the fact that as documented in Issue #2915, the Google Authenticator PAM module by default generates 80 bit keys. If this flag is set to False, the application develop should implement additional checks of the key length before passing it into `TOTP`.
  Added in version 1.5.

Raises:

- **ValueError** – This is raised if the provided `key` is shorter than 128 bits or if the `length` parameter is not 6, 7 or 8.
- **TypeError** – This is raised if the provided `algorithm` is not `SHA1()`, `SHA256()` or `SHA512()` or if the `length` parameter is not an integer.

generate(*time*) [source]

Parameters: **time** (*int*) – The time value used to generate the one time password.
Return bytes: A one time password value.

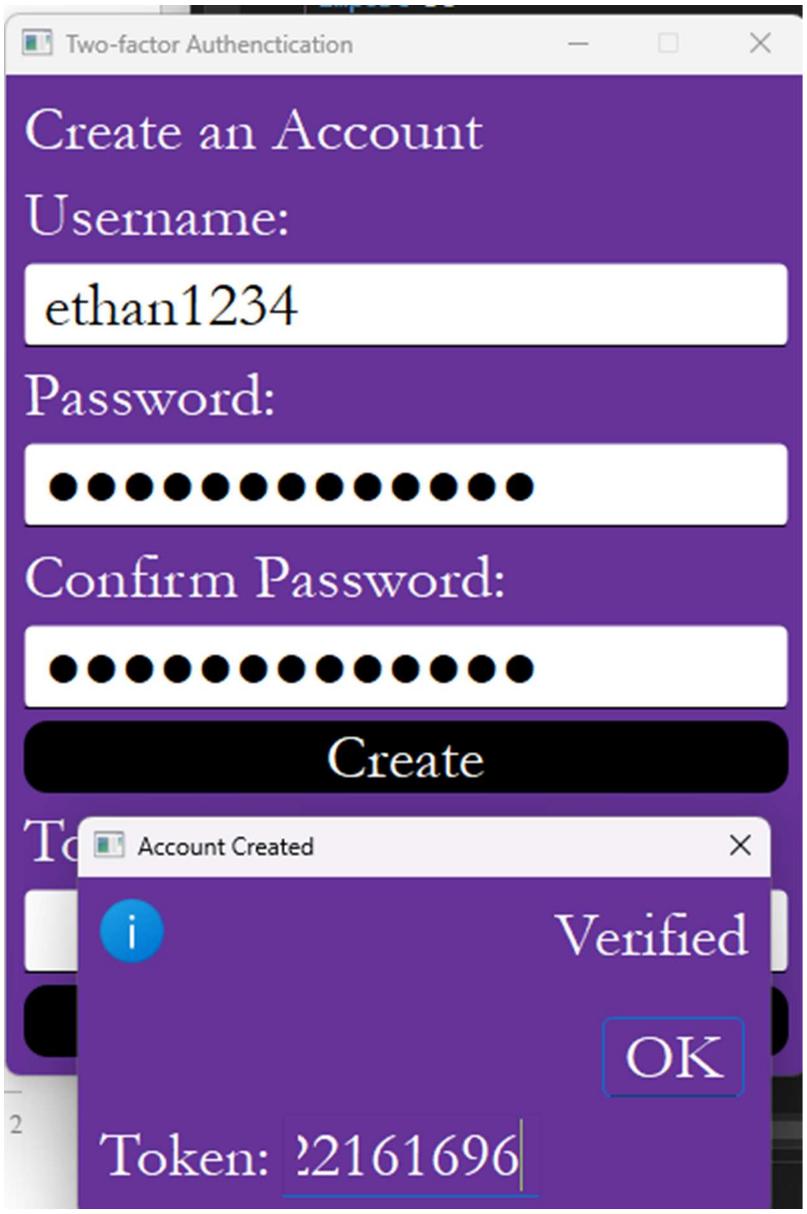
verify(*totp, time*) [source]

Parameters:

- **totp** (*bytes*) – The one time password value to validate.
- **time** (*int*) – The time value to validate against.

Raises: **cryptography.hazmat.primitives.twofactor.InvalidToken** – This is raised when the supplied TOTP does not match the expected TOTP.

Output Screenshots



Two-factor Authenctication

Create an Account

Username:

ethan1234

Password:

●●●●●●●●●●●●●

Confirm Password:

●●●●●●●●●●●●●

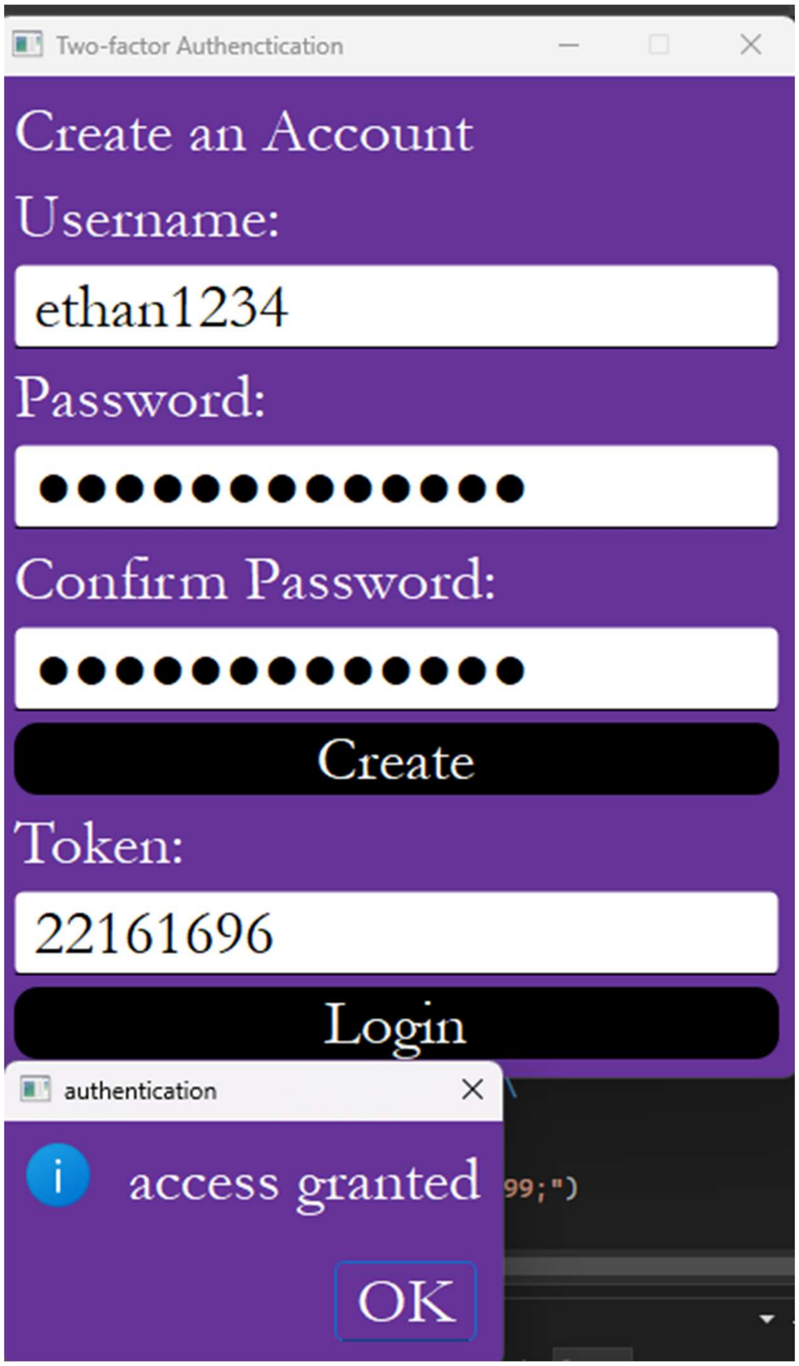
Create

Account Created

Verified

OK

Token: 22161696



Two-factor Authenctication

Create an Account

Username:

ethan1234

Password:

●●●●●●●●●●●●●

Confirm Password:

●●●●●●●●●●●●●

Create

Token:

22161696

Login

authentication

access granted

OK