Storing a key in a symmetric encryption for password manager using cryptography module

Asked 8 months ago Active 1 month ago Viewed 172 times

0

1

I am making a password manager and generator. The user creates the login information, and then I hash their password. After that, I generate the password and encrypt the password with a key stored unprotected. My Question is how do I protect the key?

I need an easy way to do it since I am just a beginner and working on this password manager as a practice. However, I can't find what I need, I also intend to put this into a GUI which I'll work on when I know how to store the keys safely.

Here is the python file:

```
else:
    with open('salt.txt','wb')as saltfile:
        salt = bcrypt.gensalt()
        saltfile.write(salt)

#Hashes the item
def hashPass(item):
    global passwordOut
    hashed = bcrypt.hashpw(item,salt)
    passwordOut = hashed

# Random Password Generator
def setPassword(length=30,char=string.ascii_letters+string.digits+string.punctuation):
    global generatedPassword
    generatedPassword= ''.join(random.choice(char) for x in range(length))
    return generatedPassword

if os.path.isfile('mykey.key') == True:
    #Opens the key
    with open('mykey.key', 'rb') as mykey:
        print('True')
        key = mykey.read()
        f = Fernet(key)

elif os.path.isfile('mykey.key')== False:
    print('False')
    # Generates a kay
    key = Fernet.generate_key()
```

encryption key-management python key-generation

Share Improve this question Follow

edited Sep 8 at 16:08
saurabh
615 1 2 12

asked Mar 28 at 1:31
Mr_Westhead
1

2 Answers

Active Oldest Votes

1

You don't usually use the password to encrypt, you use it to derive a key. This way you can generate a key with high entropy even if the underlying password is low entropy.

It allows you to create a key with the exact size as needed, no padding or truncating needed. It's better to use PBKDF2 and generate a 64-byte password than asking the user to type a password with exact 64 chars.

- User types his password (PASS): `SuperSecurePassword9000`
- You use PBKDF2 (or BCrypt) and generate a key (DKEY): `5c8fe66a...b8bfc6f8`
- You generate a random key (RKEY): `f01903e3...d2e67a0f`
- You encrypt RKEY with DKEY and store it encrypted (EKEY): `aabbccdd...11223344`

This encrypted key (EKEY) is stored. It cannot be derived back to the RKEY, and there are no traces of DKEY nor PASS anywhere. You only store EKEY.

Now the user unlocks the database. He types PASS, you derive DKEY from it, and decrypt EKEY using DKEY, getting RKEY. You can now use symmetric encryption to decrypt the database.

If the user wants to change its password, you need to re-encrypt only EKEY:

- User types the current password (PASS1)
- You derive DKEY as always, and decrypt EKEY, giving back RKEY on plain text.
- You ask for the new password (PASS2)
- You derive the new DKEY2 from PASS2
- You encrypt RKEY with DKEY2, giving you EKEY2
- You store EKEY2 and it's done

Deriving the key from the password gives you 2 advantages: you can control the security of the key by changing the number of iterations on the key derivation function to be as much as you want, and you can change the password of a very large database by changing only the encryption key.

Share Improve this answer Follow

answered Oct 8 at 20:56

ThoriumBR
46.7k 12 113 130

0

Look what the other password managers do. They encrypt the whole file where you store passwords with a single password. When user opens this file in your password manager, they ask user for a password. User provides password. They decrypt the file using this password. Now the content is decrypted and password manager can do whatever they want: Add new password entries, modify entries, delete entries. When user saves the password file, password manager

CPU and RAM. For instance, use Argon2. It provides all these features: it requires a salt, it has tunable parameters for CPU and RAM. Set these parameters to such values, that derivation takes relatively long time, but is still acceptable, e.g. so that it takes 1-2 seconds. This will make your encrypted file more resistant to brute-forcing.

Share Improve this answer Follow edited Mar 28 at 13:50 answered Mar 28 at 1:55

mentallurg
7,164 4 22 39

I shall try this thank you, so what you're saying is the password that the user creates is the key not a separate key? – Mr_Westhead Mar 28 at 2:12

Yes. A single key is sufficient to encrypt the whole file (database) with all passwords. – mentallurg Mar 28 at 3:12

@Mr_Westhead: See update. – mentallurg Mar 28 at 3:42

1 @Mr_Westhead To be clear, you should never try to encrypt anything using a password directly (usually this won't even be possible, without padding or truncating the password). Instead, you encrypt using *a key derived from a password*, using a strong password hashing / key-derivation algorithm. – CBHacking Sep 8 at 21:45