# Krptn

*Release v0.4*

**Krptn Project**

**May 13, 2024**

# KRPTN

# DOCUMENTATION CONTENT

## 1.1 Krptn

**Quick Install:**

```
pip install krptn
```

Note: we don't have pre-built extensions for all platforms. Please see the *installation section* in this documentation for more info.

If after reading this, you like our project, please consider staring on GitHub[1]!

### 1.1.1 What problem do we solve?

We all love Django and other web frameworks! However, their primary focus is creating websites - not securing them. One example is Django's built-in authentication system. While it hashes the password, it does not encrypt user data for you. Encryption is left to the developer...

Wouldn't it be nice if encryption would also be handled by the IAM? Perhaps it could be handled in a zero knowledge model, such that, without the user entering credentials, not even the database administrator can read it?! This is exactly what we do! Please see our Github Repo[2], our homepage[3], or continue reading here, in our documentation, for more information.

To prove that such is possible, we have a Flask[4] and Django[5] example on GitHub.

---

[1] https://github.com/krptn/krypton
[2] https://github.com/krptn/krypton
[3] https://www.krptn.dev/
[4] https://github.com/krptn/flaskExample
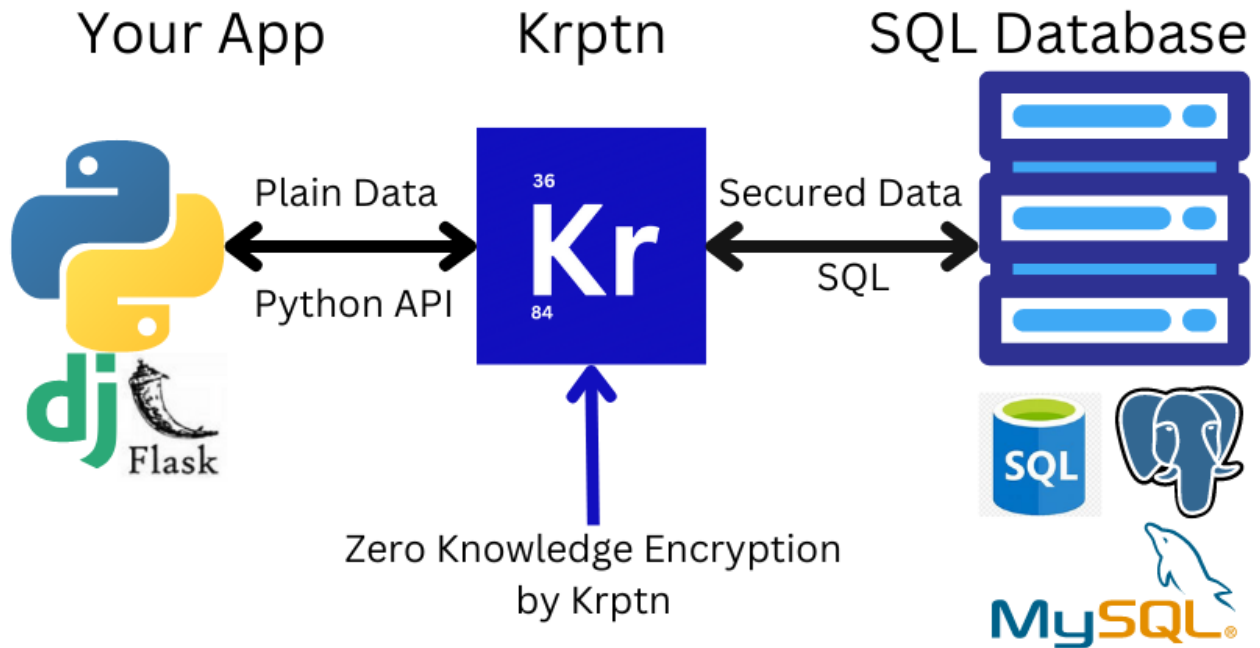[5] https://github.com/krptn/djangoExample

# Krptn's Usage Model



## 1.1.2 What is this?

A user authentication and access management system (IAM) with Zero Knowledge security[6]. It is available as a Python extension module. We, however, have certain limitations[7].

How do we achieve this?

- All data is encrypted (any data can be requested by the developer to be secured)

- Only the appropriate users' credentials can unlock the cryptosystem (this protects you from server-side attacks)

This gives you Zero Knowledge security[8] (one of the most secure available) without ever needing to even notice it! It protects you from server side attacks.

---

[6] https://www.krptn.dev/news/zero-knowledge/
[7] https://www.krptn.dev/news/limitations/
[8] https://www.krptn.dev/news/zero-knowledge/

### 1.1.3 Features

- Secure Storage of Data

- User Authentication

- FIPS Validated Cryptography (via OpenSSL 3)*

- Secure memory wiping (except on PyPy)

- FIDO Passwordless*

- Integration with popular web frameworks

* FIDO (passwordless) does not use FIPS validated resolvers

**Example usage of the Crypto Class**

```python
from krypton import basic
# Create an instance of Crypto - a class for encrypting and storing sensitive data.
myCrypto = basic.Crypto()
pwd = "Perhaps provided by the user"
# It supports C.R.U.D. operations:
id = myCrypto.secureCreate("Example data", pwd) #id is an integer
print("The data is:")
print(myCrypto.secureRead(id, pwd)) # prints Example data
```

### 1.1.4 Installation

**Recommended:** install using PIP:

```
pip install krptn
```

Because we do not have pre-built extensions for all platforms, you may need to *build from source*.

### 1.1.5 User Auth

See *User Auth*.

To use FIDO/WebAuthn with User Auth, please see Krptn's FIDO Documentation[9].

### 1.1.6 Integration with web frameworks

- *Django*
- *Flask*

---

[9] https://docs.krptn.dev/README-FIDO.html

### 1.1.7 Crypto Class

*Crypto Class*

### 1.1.8 Key Management System

This module uses a custom Key Management System for AES-256 Keys. See *KMS* for more information.

**Note:** we have considered using HSM as a key management system. We, however, have decided that we will not integrate HSMs because it would be difficult to maintain Zero Knowledge security.

Of course, all data is securely encrypted even if it is not via an HSM!

If you want, you can encrypt the SQL database using HSM managed keys for additional security.

### 1.1.9 Use custom databases

Here is an example of how to set the database to be used:

```python
import krypton
krypton.configs.SQLDefaultCryptoDBpath = "sqlite+pysqlite:///Path/example.db"
krypton.configs.SQLDefaultKeyDBpath = "sqlite+pysqlite:///Path/key.db"
```

To see what these settings strings should contain please see *Databases*

### 1.1.10 Settings

*Configurations*

### 1.1.11 Planned

- APIs for other languages
- Premium features

## 1.2 Configuration

**Note:** A change in settings (with the excpetion of database changes) will not result in decryption and re-encryption of data, only to apply the changes. Instead, the changes are applied when new data is encrypted, or old data is modified - thereby slowly phasing out the old configuration. For example, when changing password hash iterations, the change takes effect when the user resets their password.

Any database changes will result in Krptn assuming that all data has been migrated to the new database and is ready to use. Krptn will stop using the old database but will finish any started user operations in there.

Simple, pythonic configuration:

```
1  import krypton
2
3  krypton.configs.defaultAlgorithm = "AES256GCM" # Sets which symmetric cipher to use in
   ↪the KMS and Crypto Class(currently only AES256GCM is supported)
4
5  krypton.configs.defaultIterations = 500000 # Number of iterations for PBKDF2
6
7  krypton.configs.defaultPasswordResetIteration = 550000 # Number of iterations to use
   ↪with PBKDF2 for password recovery codes.
8
9  krypton.configs.defaultCryptoperiod = 2 # Approx. number of years for the cryptoperiod
   ↪of a key
10
11 krypton.configs.defaultSessionPeriod = 15 # Number of minutes before a user Session is
   ↪destroyed.
```

> **Warning:** When seting iteration counts for PBKDF2, make sure that it is not too low. A low value could make a brute-force attack against the encryption in the database easy, if it is leaked. However, note that a high value slows down your server. Depending on your needs, an appropriate value needs to be found.
>
> For your information, as of 2023 January, Django, by default, uses 480,000 iterations[10]. As of 2022 December, NIST[11] recommends a minimum of 1,000. For especially critical data, or for very powerful systems or systems where user-perceived performance is not critical, NIST states that 10,000,000 iterations may be appropriate.

### 1.2.1 Databases

For the following settings please see *Databases*

```
1  krypton.configs.SQLDefaultCryptoDBpath = # for DB used by Crypto Class
2  krypton.configs.SQLDefaultKeyDBpath =   # for DB used by Key Management System
3  krypton.configs.SQLDefaultUserDBpath = # for DB used by User Authentication System
```

### 1.2.2 FIDO Auth & MFA

These configuration options must be set for FIDO (passwordless Auth), and TOTP (time-based one-time passwords) to work.

```
1  ## For both TOTP and FIDO
2  krypton.configs.APP_NAME = "ExampleApp" # name of your app
3
4  ## The below are only needed for FIDO
5  krypton.configs.HOST_NAME = "example.com" # hostname, as seen by the user's browser
6  krypton.configs.ORIGIN = "https://example.com/" # again, as seen by the user's browser
```

---

[10] https://github.com/django/django/blob/main/django/contrib/auth/hashers.py
[11] https://csrc.nist.gov/publications/detail/sp/800-132/final

## 1.3 Custom Databases

> **Warning:** While all data saved in these databases is encrypted where necessary, please make sure that user accounts, user privileges, backup, etc. are properly configured in the database. Just because the data is encrypted, an unauthorized user can still delete it.

> **Note:** Any database configuration changes will result in Krptn assuming that all data has been migrated to the new database and is ready to use.

Internally, these strings are passed to SQLAlchemy[12] to create an engine[13]. To add extra connection parameters, please refer to SQLAlchemy's and/or your chosen database's SQL Driver documentation.

Please set these strings at:

```
1  krypton.configs.SQLDefaultCryptoDBpath = # for DB used by Crypto Class
2  krypton.configs.SQLDefaultKeyDBpath =  # for DB used by Key Management System (you most↵
   ↪likely don't need this)
3  krypton.configs.SQLDefaultUserDBpath = # for DB used by User Authentication System
```

### 1.3.1 Microsoft SQL Server

You need to install pyodbc[14] and Microsoft ODBC Driver for SQL Server[15]

The string that you need to pass to this extension should look like this:

```
1  "mssql+pyodbc://user:password@host:port/dbname?driver=odbc driver e.
   ↪g:ODBC+Driver+18+for+SQL+Server"
```

If you are only doing development, you may add the following to prevent installing an SSL certificate:

```
1  &Encrypt=no
```

To use windows authentication, please remove `user:password` from the string.

### 1.3.2 MySQL

Please install mysqlclient[16].

```
1  "mysql+mysqldb://user:password@host:port/database"
```

---

[12] https://www.sqlalchemy.org/
[13] https://docs.sqlalchemy.org/en/14/core/engines.html
[14] https://pypi.org/project/pyodbc/
[15] https://docs.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-ver16
[16] https://pypi.org/project/mysqlclient/

### 1.3.3 SQLite

```
"sqlite+pysqlite:///Path/example.db"
```

### 1.3.4 PostgreSQL

Please install psycopg2[17].

```
"postgresql+psycopg2://user:password@host:port/databse"
```

# 1.4 User Authentication

---

**Note:**

- To use Authentication in a supported web framework please see *integrations*.

---

> **Warning:**
>
> - This does not protect you against brute force attacks - make sure to enable rate limiting on your host.
> - Usernames are not encrypted.
> - User classes are not thread-safe. Please create a new object to use in each thread!
> - Krptn does not verify the security of the password (e.g: complexity), please do this yourself!
> - User names cannot be longer then 450 characters

Here is an example usage of creating a new user:

```python
from krypton.auth.users import userModel

model = userModel.standardUser(None)
model.saveNewUser("Test_UserName", "Test_Password")
```

> **Warning:** Please be carefull when setting credentials. The reasons are the following:
>
> - If you lose your credentials, and have not enabled password reset, you will permanently loose access to your account and data. To enable password reset, please read this document or *skip to the part about password resets*.
> - The encryption of your data is derieved from your credentials. Therefore, weak password equates to easily cracked encryption.
>
> All that said, don't panic :-); just enable password resets and validate user passwords for length, complexity, etc. . .

To retreive the user and set user data as key-value pairs:

---

[17] https://pypi.org/project/psycopg2/

```
1  model = userModel.standardUser(userName="Test_UserName")
2  sessionKey = model.login(pwd="Test_Password") # See below what sessionKey is
3  model.setData("test", "example") # test is the key and example is the value
4  data = model.getData("test") # Gives b"example". Would raise ValueError on error.
5  model.deleteData("test")
```

**Note:** Do make sure that the key in setData does not start with _ - those are reserved for Krypton internals.

You can also use `model.encryptWithUserKey` with `model.decryptWithUserKey`, or `shareSet` with `shareGet`, if you want other users to read it. Please study the *Data Sharing* section of this document.

**Warning:** In `setData` only the stored values are encrypted. Keys are plaintext!! Avoid storing sensitive data in keys!

### 1.4.1 User Sessions

Session keys can be used to restore a session after the user object has been destroyed. For example, in a webserver, the session key could be stored in a cookie, so that the model can be retrieved on each request.

To restore a session:

```
1  model = userModel.standardUser(userName="Test_UserName")
2  model.restoreSession(sessionKey)
```

To set session expiry please see *Configuration*.

### 1.4.2 Sign out of all sessions

```
1  model.revokeSessions()
```

### 1.4.3 Change Username

In case you want to change the user's username, you can simply do this by calling the `changeUserName` method.

```
1  model.changeUserName("NewName")
```

### 1.4.4 MFA

To avoid getting locked out, you may want to read *Password Reset* section of this document.

Before using MFA, make sure that the required *configuration values* are set.

### TOTP

To enable:

```
secret, qr = model.enableMFA()
# Secret is a shared secret and qr is a string, that when converted to QR code can be
→scanned by authenticator apps.
# If QR Codes are not supported by the app, you can tell the user enter secret instead.
# You MUST discard these once the user enabled MFA.
```

When logging in:

```
model.login(pwd="pwd", mfaToken="123456")
```

If a wrong code is provided, Krptn will impose a 5 second delay to slow brute force attacks. However, please note that is not enough to fully protect you. Therefore, it is necessary to employ a proper rate limiting solution on your webserver.

To disable TOTP (user must be logged in):

```
model.disableMFA()
```

---

**Note:** On a failed login attempt, we will impose a 5 second delay to slow down brute force attacks. This is not available for purely password based authentication, so please do impose rate limiting protection on your server.

---

### FIDO Passwordless

See *FIDO Docs*.

## 1.4.5 Data Sharing

Using these methods, you can grant another user access to some of the user's data.

While deploying these methods, all data remains encrypted using the user's credentials. No data is ever plaintext in a database! We use `Elliptic-curve Diffie-Hellman` to share a common encryption key between the users, and we encrypt the data with the common key. Each user has their own Elliptic Curve key, with the private key encrypted with the user's credentials.

---

**Warning:** One thing to note: if the original user used to set/encrypt the data is deleted. All other users will loose access. It is important that the other users create their own copy if they want to retain it.

---

### Sharing

```
model = userModel.standardUser(None)
model.saveNewUser("Test_UserName", "Test_Password") # Note: if a user with the same
→username exists an ValueError will be raised.

model2 = userModel.standardUser(None)
model2.saveNewUser("Test_UserName2", "Test_Password")

```

```
7   # Save value "data" with key "test" and allow access to user "Test_UserName"
8   user2.shareSet("test", "data", ["Test_UserName"])
9   value = model.shareGet("test") # returns b"data". Raises ValueError on error.
10  user2.shareDelete("test") # deletes the data - can only be done by the user who shared it
```

---

**Note:** Do make sure that the key in shareSet does not start with _ - those are reserved for Krypton internals.

If the same user has `shareSet` to another user multiple times with the same name, it is undefined which one will be returned when the recieving user calls `shareGet`. To avoid such conflicts, make sure to `shareDelete` data, or use a different name.

---

As you can see above, `shareSet` requires you to pass a unique name for the data (`"test"` in this case), the data (`"data"` in this case), and a list of usernames who can access it (`["Test_UserName"]` above).

### Encryption

When possible, it is preferred to use `shareSet` and `shareGet` but when required you can directly use only Krptn's encryption capabilities. E.g: if you want to use another database to store this data.

```
1   model = userModel.standardUser(None)
2   model.saveNewUser("Test_UserName", "Test_Password")
3
4   model2 = userModel.standardUser(None)
5   model2.saveNewUser("Test_UserName2", "Test_Password")
6
7   r = model.encryptWithUserKey("data")
8   model.decryptWithUserKey(r) # Returns b"data"
9
10  ## Here is the tricky part:
11
12  r = model.encryptWithUserKey("data", ["Test_UserName2"]) # Allow Test_UserName to␣
    →decrypt the data
13  model2.decryptWithUserKey(r[0][1], r[0][2], "Test_UserName") # Returns b"data"
```

In the case that an incorrect data, or key is provided, a `ValueError` will be raised.

`encryptWithUserKey` needs the following parameters: `data`, `otherUsers` (optional). `data` is the plaintext to encrypt and `otherUsers` is a list of usernames of users who can also decrypt the data.

`encryptWithUserKey` returns a list of tuples in the following format: `(username, data, salt)`. `username` is the name of the user to who we need to provide `data` and `salt`.

When decrypting, call `decryptUserKey`, on the user object corresponding to `username`, passing `data` as the first argument and `salt` as the second argument. It will return the plaintext.

Therefore, by using this method, you can grant another user access to some of the user's data, simply by allowing that user to decrypt the data.

### 1.4.6 Password Reset

To enable password reset you need to obtain recovery codes, that you can use to unlock the account.

```
keys = model.enablePWDReset() # keys is a list of OTPs that can be used to unlock the
  ↪user account
model.logout() # This is not needed but you can reset the password of a locked out user.
sessionKey = model.resetPWD(keys[0], "newPWD") # Note: you cannot use keys[0] again, use
  ↪the next one in the list.
# Note: when you call resetPWD the model will automatically login, you may want to logout
model.logout()
```

If a wrong code is provided, Krptn will impose a 5 second delay to slow brute force attacks. However, please note that is not enough to fully protect you. Therefore, it is necessary to impose a proper rate limiting solution on your webserver.

You may notice in the previous code block the `resetPWD` returns a `sessionKey`. This session key is the same as returned from the `model.login` method.

If the OTPs get compromised you can revoke them and generate new ones:

```
model.disablePWDReset() # Revoke
keys = model.enablePWDReset() # Generate. This also revokes all codes but we already did
  ↪so previously.
```

## 1.5 Migrating to Krptn

**Note:** Before attempting migration please read *User Auth*.

Multiple methods exist to migrate from your existing IAM to Krptn. We will explore the following two options:

- Creating a script to move all users at once
- Migrating users one-by-one as they login

### 1.5.1 Moving all users

To achieve this, you will have to set an initial password for the users. Copy all data into Krptn, and force the user to change their password on the next login.

### 1.5.2 Migrating Users One-By-One

When a user logs in, you can check whether they are saved in Krptn. If not, create them using the User Auth API. Since you still have the user's old password (they entered it when logging in), you can avoid creating an initial password. After that, you can copy the user's data into Krptn's user object.

# 1.6 WebAuthn/FIDO Passwordless: both server and client setup guide

**Note:** *we have originally published this article on our* homepage[18] *and only later added it to the documentation also.*

For a live demo, there is a working version[19] on GitHub.

Please note that this is a tutorial, not a full documentation. To use this in a production environment, please read our FIDO docs[20] also!



## 1.6.1 What is WebAuthn?

Passwords are the root cause of 80% of data breaches! It is so common, yet so many fall for it. . . It's simply time to fix this mess. So, the world is deploying a solution: WebAuthn.

WebAuthn allows you to login to websites using a gesture. For example, a fingerprint, an external authenticator, etc.

The server creates a challenge that the authenticator device signs using the credential. When the server verifies the signature, it can authenticate the user. Usually, the authenticator will require the user to verify their identity using a fingerprint, a pin code or other gesture.

Examples of authenticators include external hardware security keys and your device's trusted platform module.
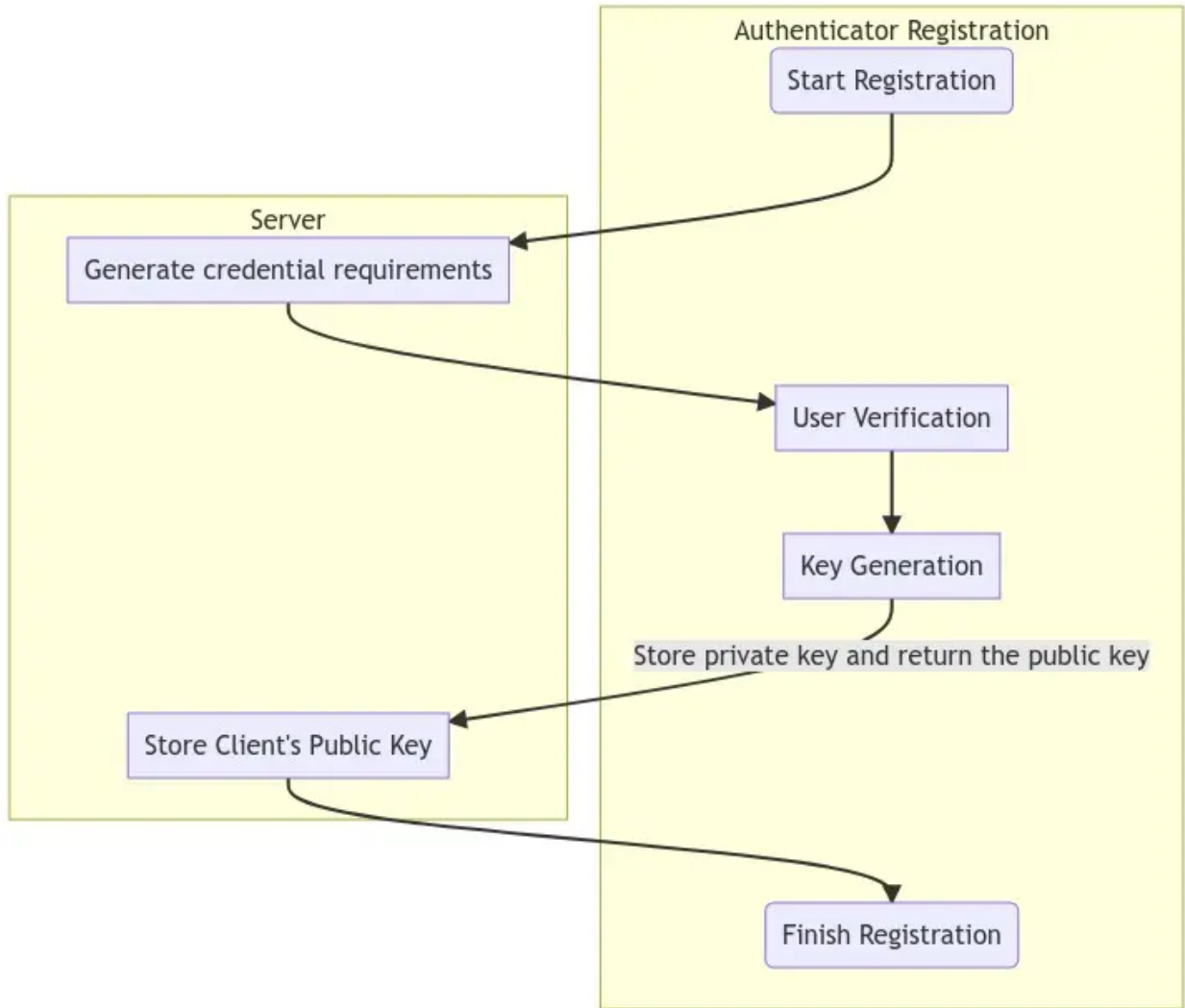
Here is the WebAuthn flowchart. Don't worry: you don't need to understand this to use it in your website :-)!
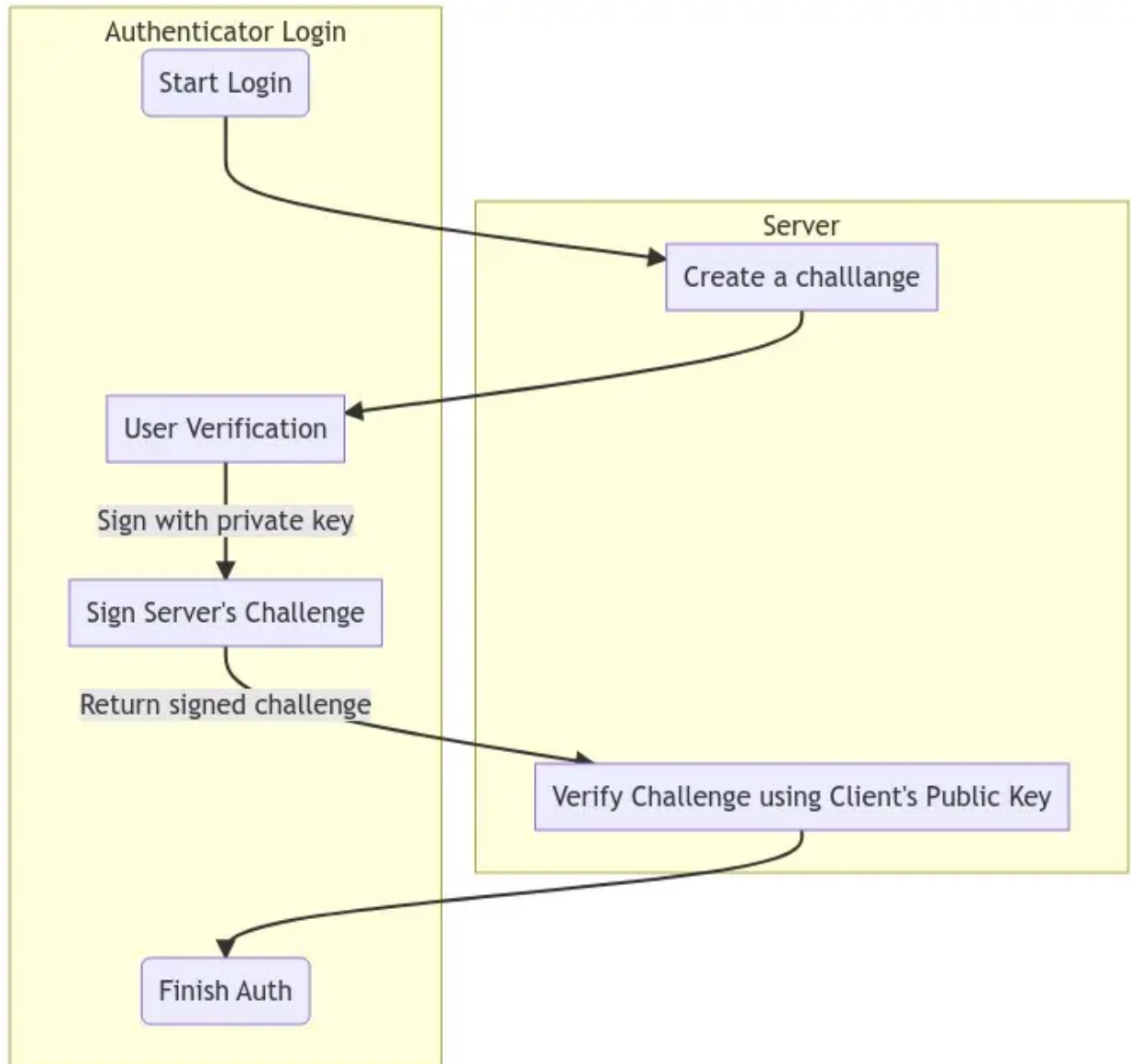
Here is the flowchart for registration:

---

[18] https://www.krptn.dev/news/webauthn/
[19] https://github.com/krptn/flaskExample
[20] https://docs.krptn.dev/README-FIDO.html

Here is the flowchart to login:

Because of this mechanism, it is impossible to phish WebAuthn credentials, as they are never released from the authenticator. Only the signed challenge is released, which is not enough to obtain the credentials. Because a large portion of cyberattacks come from breached passwords, this can massively improve your security.

## 1.6.2 Setup your environment

Before proceeding, please install Python on your computer. Depending on your system, you may have Python already installed. If not, you can always get the latest version from the Python Homepage[21]!

Next, use the Python package manager (pip) to install Krptn:

```
pip install krptn
```

If pip cannot find the wheels, you may need to build Krptn from source. The installation section[22] of our documentation

---

[21] https://python.org/
[22] https://docs.krptn.dev/README-BUILD.html

contains instructions.

### 1.6.3 Create the client

First, we need some external JS:

```
<script async src="https://cdn.jsdelivr.net/gh/herrjemand/Base64URL-ArrayBuffer@latest/
↪lib/base64url-arraybuffer.js"></script>
```

#### Registration

We need a way to register the credential in the user's browser, so that the challenge can be signed.

Secondly, we need to obtain the credential's options from the server. In the next sections, we will discuss how we generate these:

```
const response = await fetch('/fidoReg', {cache: 'no-cache'});
const options = await response.json();
```

It is important that the response is decoded:

```
options.user.id = base64url.decode(options.user.id);
options.challenge = base64url.decode(options.challenge);
if (options.excludeCredentials) {
    for (let cred of options.excludeCredentials) {
        cred.id = base64url.decode(cred.id);
    }
}
```

And the moment we were all waiting for! We can register the credential with the browser:

```
const cred = await navigator.credentials.create({
    publicKey: options,
});
```

Unfortunately, our job is not done… We need to upload the browser's response to the server. After this, the server will store the credential in the database!

We need to prepare the browser's response first:

```
if (cred.response) {
    const clientDataJSON =
        base64url.encode(cred.response.clientDataJSON);
    const attestationObject =
        base64url.encode(cred.response.attestationObject);
    credential.response = {
        clientDataJSON,
        attestationObject,
    };
}
```

Finally, we are ready to upload the response and finalize the registration.

```
await fetch('/fidoFinishReg', {
    body: JSON.stringify(credential),
    cache: 'no-cache',
    method: 'POST',
    headers: {'Content-Type': 'application/json'}
});
```

Again, these mysterious server endpoints will the discussed!

### Login

While according to the official WebAuthn website, the user does not need to enter the password, because of Krptn's Zero Knowledge approach, we will require the user to provide the password anyway.

```
const email = "EMAIL";
const pwd = "PASSWORD";
```

First, we need to request the FIDO (WebAuthn) challenge from the server:

```
const query = {};
query.email = email;
const repsonse = await fetch('/getFidoLogin', // Mysterious endpoints will be discussed
    {cache: 'no-cache',
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify(query)}
);
const options = await repsonse.json();
```

Decode:

```
options.challenge = base64url.decode(options.challenge);
for (let cred of options.allowCredentials) {
    cred.id = base64url.decode(cred.id);
}
```

Next, we can request the browser to sign the challenge, thereby proving the user's identity:

```
const cred = await navigator.credentials.get({
    publicKey: options
});
```

This response will be uploaded to the server. But first, we need some base64!

```
const credential = {};
credential.fido = 1;
credential.id = cred.id;
credential.type = cred.type;
credential.rawId = base64url.encode(cred.rawId);

if (cred.response) {
    const clientDataJSON =
        base64url.encode(cred.response.clientDataJSON);
```

```
    const authenticatorData =
        base64url.encode(cred.response.authenticatorData);
    const signature =
        base64url.encode(cred.response.signature);
    const userHandle =
        base64url.encode(cred.response.userHandle);
    credential.response = {
        clientDataJSON,
        authenticatorData,
        signature,
        userHandle,
    };
}
```

Finally, we can upload the response to the server and finish the authentication:

```
credential.pwd = pwd;
credential.email = email; // These are required by Krptn
authToken = await fetch('/fidoFinishLogin', {
    body: JSON.stringify(credential),
    cache: 'no-cache',
    method: 'POST',
    headers: {'Content-Type': 'application/json'}
});
```

### 1.6.4 Create the server

The following section assumes that the users have already been created in Krptn. You can quickly create them as discussed in the User Auth docs[23].

This also allows you to take advantage of other extensive security features in Krptn.

In order for WebAuthn to work, you need to set certain configuration options:

```
1  import krypton
2
3  krypton.configs.APP_NAME = "ExampleApp" # name of your app
4  ## The below are only needed for FIDO
5  krypton.configs.HOST_NAME = "example.com" # hostname, as seen by the user's browser
6  krypton.configs.ORIGIN = "https://example.com/" # again, as seen by the user's browser
```

HOST_NAME can be set to `localhost` and ORIGIN can be set to `https://localhost` for development.

---

[23] https://docs.krptn.dev/README-USER-AUTH.html

**Registration**

Inside the `/fidoReg` endpoint:

```
1  from krypton.auth import users
2
3  model = users.standardUser(email)
4  key = model.login(...)
5  ... # Standard Krptn login procedure
6  options = model.beginFIDOSetup()
```

This `options` needs to be the response sent to the browser.

Inside `/fidoFinishReg`:

```
1  model.completeFIDOSetup(request_json_string)
```

Of course, you will have to store the user model in the session.

This is best achieved by setting `key`, as returned from `model.login` in a cookie, so that on each request, you can restore the session:

```
1  model = users.standardUser(username_from_cookie)
2  model.restoreSession(key_from_cookie)
```

**Login**

Inside `getFidoLogin`:

```
1  model = users.standardUser(email_slash_username)
2  options = model.getFIDOOptions()
```

`options` needs to be provided in response to the request.

Inside `fidoFinishLogin`:

```
1  model = users.standardUser(name_email)
2  key = user.login(password, fido=fidoChallangeFromBrowser)
```

As mentioned, `key` can be set to keep the user authenticated in the session. Please see our User Auth docs[24] for more information.

## 1.6.5 Pulling it all together

Depending on which web framework you are using, the client and server side needs to be glued together differently. We have an example where it is glued together with Flask on GitHub[25].

Last, but certainly not least, after creating a GUI where the user can enter the email and password, you are ready!

---

[24] https://docs.krptn.dev/README-USER-AUTH.html
[25] https://github.com/krptn/flaskExample

## 1.6.6 Full Code and wrap up

---

**Note:** In order for this section to make sense, please read *User Auth* first.

---

To see WebAuthn with Krptn implemented in action, you can have a look at our Flask example[26] on GitHub.

First make sure that the required *configuration options* for FIDO are set (app name, origin).

Currently, we only support passwordless as a second (or third) authentication factor. The password still has to be enabled.

For security reasons, we can only have one FIDO credential registered. To remove the FIDO credential:

For convention,`model` will be the current user's user model (that is, `standardUser` object). It is your task to retrieve the model using *user sessions*.

```
model.removeFIDO()
```

### Register

```
options = model.beginFIDOSetup()
```

The above code generates options for FIDO. Please send these to the client's browser. In the browser, please run the following JS:

```
<!---
    This code was taken from Google's WebAuthn Glitch Tutorial: https://glitch.com/edit/#!
→/webauthn-codelab-start?path=README.md%3A1%3A0
    This code was changed to work with Krypton's Auth Backends. These include changing
→auth URLs, loading JSON data.

    Here is the original copyright notice:

    Copyright 2019 Google Inc. All rights reserved.

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        https://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License
--->
<script async src="https://cdn.jsdelivr.net/gh/herrjemand/Base64URL-ArrayBuffer@latest/
→lib/base64url-arraybuffer.js"></script>
<script>
    async function register() {
```

(continues on next page)

---

[26] https://github.com/krptn/flaskExample

```javascript
        const response = await fetch('/fidoReg', {cache: 'no-cache'}); // /fidoReg
→should return FIDO options as generated above
        const options = await response.json();

        options.user.id = base64url.decode(options.user.id);
        options.challenge = base64url.decode(options.challenge);

        if (options.excludeCredentials) {
            for (let cred of options.excludeCredentials) {
                cred.id = base64url.decode(cred.id);
            }
        }

        const cred = await navigator.credentials.create({
            publicKey: options,
        });

        const credential = {};
        credential.id = cred.id;
        credential.rawId = base64url.encode(cred.rawId);
        credential.type = cred.type;

        if (cred.response) {
            const clientDataJSON =
                base64url.encode(cred.response.clientDataJSON);
            const attestationObject =
                base64url.encode(cred.response.attestationObject);
            credential.response = {
                clientDataJSON,
                attestationObject,
            };
        }
        localStorage.setItem('KryptonFIDOcredId', credential.id);
        return await fetch('/fidoFinishReg', { // See below what /fidoFinishReg should do
            body: JSON.stringify(credential),
            cache: 'no-cache',
            method: 'POST',
            headers: {'Content-Type': 'application/json'}
        });
    }
</script>
```

Please see our tutorial[27] for more details on the above code.

Inside `/fidoFinishReg` (or however you rename it):

```python
import json
model.completeFIDOSetup(json.dumps(request.get_json()["credentials"])) # Of course,
→depending on your web framework this will differ
```

---

[27] https://www.krptn.dev/news/webauthn/

### Login

First of all, we need to obtain our FIDO options:

```
1 options = model.getFIDOOptions()
```

These will need to be transmited to the browser, and the result (returned from the browser) of the authentication should be passed to `login` function:

```
1 model.login(pwd='MyPWD', fido=fidoResponse) # fidoResponse, is the stringified JSON from
  ↪the browser.
```

On failure, a `krypton.auth.users.bases.UserError` will be raised and `model.FIDORequired` will be set to `True`.

To obtain authentication result in the browser:

```
<!---
    Some of this code was taken from Google's WebAuthn Glitch Tutorial: https://glitch.
↪com/edit/#!/webauthn-codelab-start?path=README.md%3A1%3A0
    This code was changed to work with Krypton's Auth Backends. These include changing
↪auth URLs, loading JSON data.

    Here is the original copyright notice:

    Copyright 2019 Google Inc. All rights reserved.

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        https://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License
--->
<script async src="https://cdn.jsdelivr.net/gh/herrjemand/Base64URL-ArrayBuffer@latest/
↪lib/base64url-arraybuffer.js"></script>
<script>
    async function doFido() {
        const email = document.getElementsByName('email')[0].value; // Replace with your
↪password form
        const pwd = document.getElementsByName('password')[0].value; // Replace with
↪your password form
        const query = {}
        query.email = email;

        // To the below request, please return the response from model.getFIDOOptions()
        // Don't forget to replace your endpoint
        const repsonse = await fetch('/getFidoLogin', // Replace endpoint with yours
            {cache: 'no-cache',
```

(continues on next page)

```
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(query)}
    );

    const options = await repsonse.json();

    options.challenge = base64url.decode(options.challenge);

    for (let cred of options.allowCredentials) {
        cred.id = base64url.decode(cred.id);
    }
    const cred = await navigator.credentials.get({
        publicKey: options
    });

    const credential = {};
    credential.fido = 1;
    credential.id = cred.id;
    credential.type = cred.type;
    credential.rawId = base64url.encode(cred.rawId);

    if (cred.response) {
        const clientDataJSON =
            base64url.encode(cred.response.clientDataJSON);
        const authenticatorData =
            base64url.encode(cred.response.authenticatorData);
        const signature =
            base64url.encode(cred.response.signature);
        const userHandle =
            base64url.encode(cred.response.userHandle);
        credential.response = {
            clientDataJSON,
            authenticatorData,
            signature,
            userHandle,
        };
    }
    const finalCredentials = JSON.stringify(credential);
    // Please pass the stringified JSON `finalCredentials` as the `fido` parameter to␣
→the `login` function.
    // You still need to provide the user's password to the funcion also.
    }
</script>
```

### 1.6.7 Copyright

Throughout the tutorial, we used some code from Google's tutorial on FIDO. This code is in the client side where we decode/encode the credentials.

Here is the original, Google's, copyright notice:

```
Copyright 2019 Google Inc. All rights reserved.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    https://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License
```

## 1.7 Flask Integration

It is important the you first read *User Auth API*. The User Auth API needs to be used in Flask.

In this example we will use WebAuthn (FIDO).

For Flask Integration, please see the example implementation on GitHub[28].

## 1.8 Django Integration

**Warning:** Because Krptn uses a Zero Knowledge[29] security model, the Django Admin cannot change the user's password, or any other attributes. Any attempt to do so will fail. To avoid tempting administrators to try anyway, it may be a good idea to remove these forms from the admin site.

**Note:** In order for this to make sense please read *User Auth* first. It it also useful to have a knowledge of the Django webframework.

You can check our example implementation[30] on GitHub.

As you will notice, Krptn does not integrate with Django's Authentication but rather serves as a replacement. However, this comes with the limitation that (whitout extra programming) you cannot login with a Krptn account into the Django admin site. For that, you need to create a Django account.

---

28 https://github.com/krptn/flaskExample
29 https://www.krptn.dev/news/zero-knowledge/
30 https://github.com/krptn/djangoExample

Also, Krptn accounts, because of our strict security model, will have issues with Django's permissions. Avoid using built-in Django permision management and instead do permision checks manually. The only exception to this is the `login_required` decorator, but this is only valid, if you have your custom login page configured (`/accounts/login`).

### 1.8.1 Middleware

In order to use Krptn's user model as `request.user` in your view, you need to install the middleware.

Please add Krptn middleware to the END OF THE LIST.

For exmaple:

```
1  MIDDLEWARE = [
2      'django.middleware.security.SecurityMiddleware',
3      'django.contrib.sessions.middleware.SessionMiddleware',
4      'django.middleware.common.CommonMiddleware',
5      'django.middleware.csrf.CsrfViewMiddleware',
6      'django.contrib.auth.middleware.AuthenticationMiddleware',
7      'django.contrib.messages.middleware.MessageMiddleware',
8      'django.middleware.clickjacking.XFrameOptionsMiddleware',
9      'krypton.auth.django.middleware.kryptonLoginMiddleware' ## <-- Like here
10 ]
```

Inside your views:

```
1  def aRandomView(request):
2      request.user.
3      # request.user is a krypton.auth.django.users.djangoUser object
4      # djangoUser object has the same interface as a standardUser objects
```

### 1.8.2 Forms

Krptn requires custom forms for user management (e.g.: creation, password reset, etc..). You need to configure the forms. Because everyone has a wide variety of needs regarding user creation, there is no single form to use. You need to create these forms according your needs.

We will breifly discuss how to create these forms.

#### Create User

```
1  from django import forms
2  from krypton.auth.django import users
3
4  class UserCreationForm(forms.Form):
5      Password = forms.CharField(widget=forms.PasswordInput)
6      userName = forms.CharField(label = "User Name")
7      age = forms.CharField(label = "Age")
8      def save(self, commit=True):
9          user = users.djangoUser(None)
10         token = user.saveNewUser(pwd=self.cleaned_data["Password"], name=self.cleaned_
   data["userName"])
```

(continues on next page)

```
11          user.setData("Age", self.cleaned_data["age"])
12          return token, user.id
```

*Do not forget to set* `token` *and* `user.id` *as cookies in any view that handles authentication!! Otherwise, the middleware will have issues!*

```
1   response.set_cookie("_KryptonUserID", UserId)
2   response.set_cookie("_KryptonSessionToken", token, 15*60) # set token for 15 minutes
```

*The cookies have to have the same name as in the above example.*

Again, you will need to customise this form to include fields that you need.

As you can see we use Krptn's *User Auth* inside the forms `save` method.

### Login

This form depends on whether you are using MFA and whether you use FIDO or TOTP MFA.

In this example, we will use TOTP.

```
1   class LoginForm(forms.Form):
2       userName = forms.CharField(label = "User Name")
3       password = forms.CharField(widget=forms.PasswordInput)
4       totp = forms.IntegerField(label = "TOTP")
5       def save(self, commit=True):
6           user = users.djangoUser(self.cleaned_data["userName"])
7           token = user.login(pwd=self.cleaned_data["userName"], mfaToken=str(self.cleaned_
    →data["totp"]))
8           return token, user.id
```

*Do not forget to set* `token` *and* `user.id` *as cookies in any view that handles authentication!! Otherwise, the middleware will have issues!*

```
1   response.set_cookie("_KryptonUserID", UserId)
2   response.set_cookie("_KryptonSessionToken", token, 15*60) # set token for 15 minutes
```

*The cookies have to have the same name as in the above example.*

### Other forms

There are plenty of other possible forms. For example, enabling MFA, password resets, etc.. However, we will not discuess them.

In case of any doubt, you can check our example[31] on GitHub or reach out to us.

---

[31] https://github.com/krptn/djangoExample

## 1.9 Crypto Class

> **Warning:** Crypto Class is not thread-safe. Please create a new object to use in each thread!

Here is a simple usage example:

```python
from krypton.basic import Crypto

cryptoObject = Crypto()
id = cryptoObject.secureCreate("data", "pwd") # returns an integer
print("Reading data:")
print(cryptoObject.secureRead(id, "pwd")) # Prints data

print("Updating data:")
cryptoObject.secureUpdate(id, "New Data", "pwd")
print(cryptoObject.secureRead(id, "pwd")) # Prints New Data

print("Deleting:")
cryptoObject.secureDelete(id, "pwd")
```

## 1.10 Key Management System

> **Note:** KMS is not thread-safe. Please create a new object to use in each thread!

This module uses a custom Key Management System for AES-256 keys.

You need to identify the key with a `name` and a `password`.

```python
from krypton.basic import KMS
obj = KMS()
key = obj.createNewKey("KeyName", "password")
keyAgain = obj.getKey("KeyName", "password")
## Note getKey raises a krypton.basic.KeyManagementError
# if the cryptoperiod of the key has expired as
# specified in the configs. To get the key
# anyway, add force=True to the parameters.
obj.removeKey("KeyName", "password")
```

## 1.11 Building Krptn

In case there aren't any pre-built wheels on PyPI for your platform, you may need to build from source.

### 1.11.1 Building from source

*Note:* apart from x86 on Windows, only 64-bit environments are supported.

To build Krptn, you will need:

- A C++ compiler
- Build tools required by OpenSSL (please see OpenSSL's guide for information[32])

After cloning the repo from GitHub[33] (and checking out your version using git tags), please build and install OpenSSL 3, which is included as a git submodule:

- Install openssl in the `kr-openssl-install/` and place configurations in `kr-openssl-config/` directories.
    - Therefore, in the configure script, you need `--prefix` and `--openssldir` set.
- As Krptn uses the FIPS module, please set `enable-fips` also.

For complete instructions on building OpenSSL, please see OpenSSL's installation instructions[34].

For example (Windows example):

```
perl Configure --prefix="C:\Users\markb\source\repos\krypton\kr-openssl-install" \
  --openssldir="C:\Users\markb\source\repos\krypton\kr-openssl-config" \
  enable-fips
```

You need to both build and install OpenSSL:

```
make
make install
```

To install the extension:

```
pip install .
```

## 1.12 CLI Interface

Currently Krptn only exposes one CLI option: `--clean`.

This commands cleans the default database used by Krptn. It will not affect any custom databases set by the user.

Because it erases all content from the database, to avoid errors, please close any instances of Krptn before running this command.

Here is an example usage:

```
python -m krypton --clean
```

---

[32] https://github.com/openssl/openssl/blob/master/INSTALL.md#prerequisites
[33] https://github.com/krptn/krypton
[34] https://github.com/openssl/openssl/blob/master/INSTALL.md

## 1.13 Common Issues

### 1.13.1 I get SQLAlchemy errors about missing columns/tables

This error occures if Krptn is upgraded and the new version uses a different database schema. Generally, there will be tools provided to fix such errors. Please check the corresponding release on GitHub.

However, if you are using a development or pre-release version you will have to erase your database. If you have not set a custom database, this can easily be achieved from *Krptn's CLI*:

```
python -m krypton --clean
```

### 1.13.2 I get an error about failing to load the FIPS provider

Krptn uses OpenSSL's FIPS module for cryptography. Generally, this allows US government agencies to use this software. However, the downside is that it requires self-tests to be loaded. This error happens when the self tests could either not be performed, or there is an error in the configuration.

If you get this error, please open an issue on GitHub[35] and attempt the following fix:

1. Locate your Krptn installation by running `pip show krptn`.

2. Navigate to the location inidicated by pip.

3. There is a folder named `kr-openssl-install`. Inside that, locate:

    - on linux, `libssl.so.3` and `libcrypto.so.3`;

    - on windows, `libcrypto-3.dll` and `libssl-3.dll` (or for x64 `libssl-3-x64.dll` and `libcrypto-3-x64.dll`);

    - on Mac, `libcrypto.dylib` and `libssl.dylib`.

4. With this directory:

    - On Windows, add it to `PATH`;

    - On GNU/Linux, call `/sbin/ldconfig <directory-path>`, or set `LD_LIBRARY_PATH` environment variable to point to it;

    - On MacOS, set `DYLD_LIBRARY_PATH` to point to it.

5. Inside `kr-openssl-install` locate the openssl executable.

6. Inside `kr-openssl-install` locate the fips shared library (either `fips.so`, `fips.dll`, or `fips.dylib`).

7. In the path where pip inidcated packages are installed, there is another folder named `kr-openssl-config`.

    - In that directory, locate `openssl.cnf` and delete it;

    - Create an empty `fipsmodule.cnf` or empty it if it already exists.

8. Call the Openssl executabe `openssl fipsinstall -module {path to fips lib} -out {your path to kr-openssl-config/fipsmodule.cnf}`.

9. Retry the task you were attempting.

10. Open an issue on GitHub[36] about this so that we can prevent such bugs in the future.

---

[35] https://github.com/krptn/krypton
[36] https://github.com/krptn/krypton

### 1.13.3 Other issues

If you have any other issues, we recommend doing the following:

1. Checking that your Krptn install is the latest version.

2. Opening an issue on GitHub[37] about your problem.

## 1.14 Security Policy

### 1.14.1 Supported Versions

Only the most recent version is supported; however, we are still interested in learning about security vulnerabilities in previous versions.

### 1.14.2 Reporting a Vulnerability

Email security vulnerabilities to security@krptn.dev.

Please make sure the following information is clearly stated:

- What components are affected?

- How to exploit it?

- Minimal reproducible example

- Recommendations on fixes, if any

### 1.14.3 Vulnerability Publishing

Any published vulnerabilities will be available under the Security tab of affected GitHub repositories. To view them, click on the tab and select advisories under the reporting section.

Important vulnerabilities will also appear under our news on our homepage[38].

### 1.14.4 GPG

Depending on the severity of the vulnerability, you might want to encrypt it with GPG/PGP before sending it.

Our key is uploaded to `pgp.mit.edu`, `keys.openpgp.org` and should be available on `keys.gnupg.net` (please use whichever you prefer), with a key id of `7126492594E7DCA0`. It is also copy-pasted into the bottom of this file.

(Please substitute "server" with your preferred server from the above mentioned)

```
gpg --keyserver server --recv-keys 7126492594E7DCA0
```

The following EC (nistp384, encrypt & sign) key was created on 2022-06-30 and expires on 2025-06-29:

---

[37] https://github.com/krptn/krypton
[38] https://www.krptn.dev/news/

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mG8EYr2dPRMFK4EEACIDAwTc05svlFaiiWGCRxJ1FLdMJXXm1zRBn7XUnOm2AOUQ
SvOu0rpdnuLxDKMPFgBEerUk/wUkWC4SKA0UsaVRAFlaG8nwKTFJgWIMVjE6oNG2
qc5+pANQmtq3/pr2ktbANfa0aktyeXB0b25TdXBwb3J0IChUaGlzIGtleSBjYW4g
YmUgdXNlZCB0byBzZW5kIHNlbnNpdGl2ZSBpbmZvcm1hdGlvbiB0byBLcnlwdG9u
IFN1cHBvcnQpIDxzdXBwb3J0QGtycHRuLmRldj6IuQQTEwkAQRYhBHVHhK+OZb3q
W3WB6XEmSSWU59ygBQJivZ09AhsDBQkFo5qABQsJCAcCAiICBhUKCQgLAgQWAgMB
Ah4HAheAAAoJEHEmSSWU59ygzxsBgLqSADOF2ptNHftcQ1ZtPAY33gvLfN2rB6rF
fWH7GttqgWtd9Zcka+bgiqkrscdyXQGA2/fwKjxYu495FIMhnN555omRCKcLTRpi
/yoPKgRSqW7nFhW8llmJDY7QHYtg6i46iQIzBBABCAAdFiEEOT9Dt7NJPaH87PQJ
BwS8y6FfaRQFAmLP6hwACgkQBwS8y6FfaRQYMhAAgXTqvudNXG38eCJ6UMfdQug9
vEW1PL9f7Yq8mTa6hU2yJ8JZ1OH/YAeLPCIHBEnKC2XMYQ9X1hFKEpF4it3/KBVx
ZuRnIGY59Wp0cat8l6sRYLlIe4iw+fcTbhtEk8qCpbIXh09OzmwfbWFnX61mVdQC
7vgtZxWqZUfSJgRKpY/NGrENvvoyEQHqIZFeqJE5Bnnd2rj3QC7dyTx8eiQ+a3ZG
7Zw3TpXlc99a0t7L5upkvzZ7eKO6woh1NNN7VgXkHQ+GMCwEKsqdqT3jHt1uYimO
7CXrJPMtMnGEsvjMBeqRcVK8ge7je14x9p5j9nUEdXWo2g8oBC/4O87nEPzA78w2
CaYd/62OudLjV04mp4knKbHxKbBlZCtnGyOs/5NgpdM8jQWKsOnepbyGTtPKIhMQ
LEPM+JCIswtRtav+0QJQztrFQDFb174rl3gmNtseA60+SULmF77caboaqC5trDfn
YZHs3Zl800C55wY8v+tj2UgedWGYNKn4HUux3SAmNmVat/JE33bqYfsLrbcqV5z3
gK68nWjVfPAkLklQxywryXOFSfEnP9zBShiNjUmEWEY3KGYOqmo4q3X8Mo7mYRm6
bv+BT8ojmw8tDGBUNlcHqOqq/Q7nDyq782TKzbo2MgTCnpUNS4JLLV6is9/3RI1G
wX2jpX2YfzMOUOS9GUq4cwRivZ09EgUrgQQAIgMDBAloh4vRykF24nRdlsbB9aP5
RyGREr4VxxmR0LN+NcH2pUCaH8sYOK1Q75ki2Wgj8AdJ0uU5dyCWYHXQ7z7Ww60Z
4aN2u4d7KENBIxlUUMKD3zW12AHpabmB9ysrtSBfAgMBCQmIngQYEwkAJhYhBHVH
hK+OZb3qW3WB6XEmSSWU59ygBQJivZ09AhsMBQkFo5qAAAoJEHEmSSWU59yg25sB
gI8TP2MU1h5rIp1E7C1tJPahkq9YYVvhFX0JQ2xG2G/YuLVaLtqO+N+CoslnuNuA
OAGAqso7qKBIvidjN4jDdppBxPpEwYco+GLqjL29RwWz3MsdAfBtmGhEL3mUcqDq
dMF9
=x4k7
-----END PGP PUBLIC KEY BLOCK-----
```

## 1.15 API Reference

### 1.15.1 krypton.basic

### 1.15.2 krypton.auth.users.bases

### 1.15.3 krypton.auth.users.userModel

### 1.15.4 krypton.auth.users.userModelBaseAuth

### 1.15.5 krypton.auth.users.userModelMFAAuth

### 1.15.6 krypton.auth.factors

### 1.15.7 krypton.auth._utils

### 1.15.8 krypton.base