Krptn

Release v0.8

Krptn Project

May 13, 2024

KRPTN

1 Documentation Content **1**

1.1 Krptn . . . 1
1.2 Configuration . . . 4
1.3 Custom Databases . . . 5
1.4 User Authentication . . . 7
1.5 Migrating to Krptn . . . 13
1.6 WebAuthn/FIDO Passwordless: both server and client setup guide . . . 13
1.7 Flask Integration . . . 25
1.8 Django Integration . . . 25
1.9 Crypto Class . . . 28
1.10 Key Management System . . . 28
1.11 Building Krptn . . . 29
1.12 CLI Interface . . . 30
1.13 Common Issues . . . 30
1.14 Security Policy . . . 32
1.15 API Reference . . . 33

Python Module Index **47**

Index **49**

CHAPTER

ONE

DOCUMENTATION CONTENT

1.1 Krptn

Please have a look at our homepage[1] for an overview of the project. Here we only host documentation.

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[2]!

1.1.1 What problem do we solve?

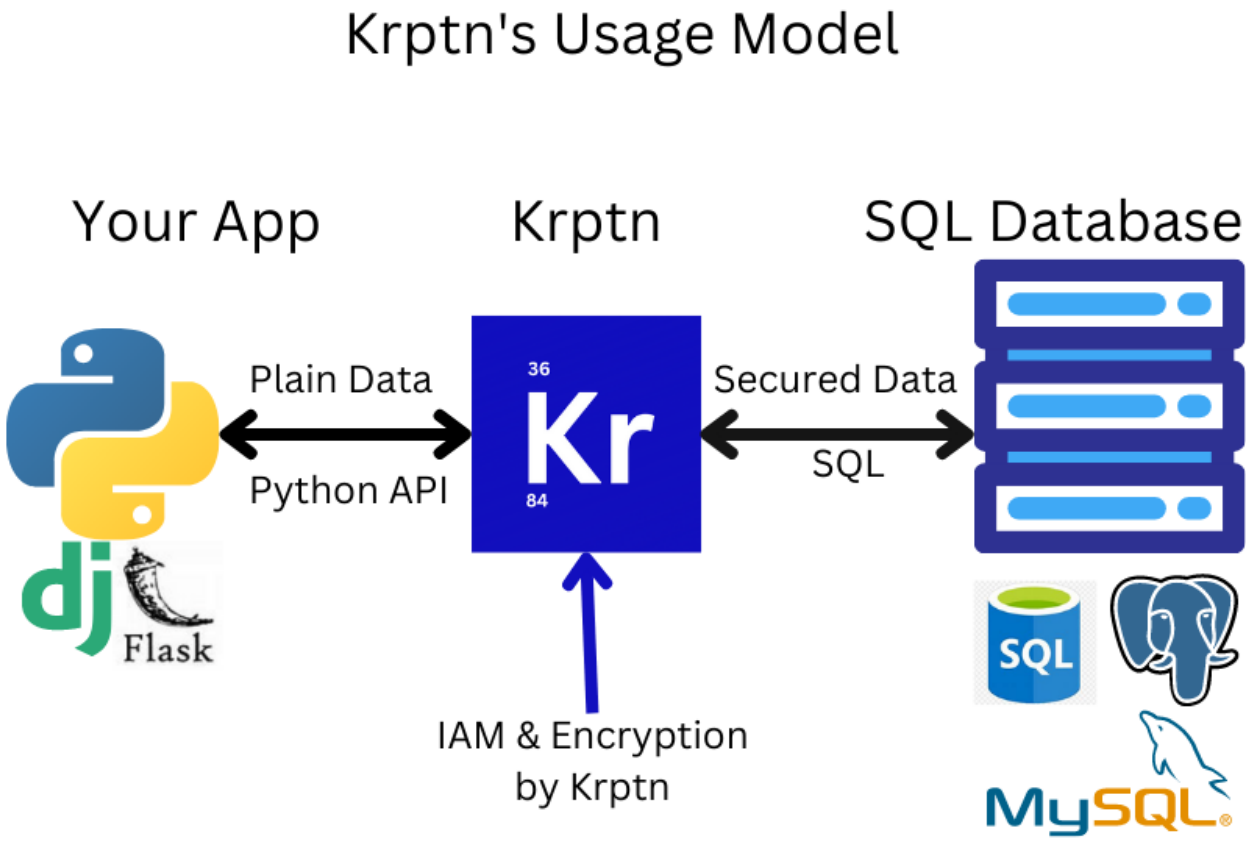
We all love Django and other web frameworks! However, their primary focus is creating websites - not implementing secure storage for user data. Django makes it easy to store data. While it hashes the password, it does not encrypt user data for you. In case of a data breach, malicious actors could access any data from the DB. Encryption is left to the developer…

Wouldn't it be nice if encryption would also be handled? Perhaps it could be handled in a **way that keys are derived from credentials**[3], such that, without the user entering credentials, not even the database administrator can read it?! This is exactly what we do!

Krptn also runs in the same server instance as your web app. So you don't have to host anything new. Just install the extension for Python.

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

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



1.1.2 What is this?

A user authentication and access management system (IAM) with **data encryption at rest derived from credentials**[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 *security from encryption*[8] without ever needing to even notice it! It protects you from server side attacks.

[6] https://www.krptn.dev/news/security-model/
[7] https://www.krptn.dev/news/limitations/
[8] https://www.krptn.dev/news/security-model/

1.1.3 Features

- Secure Storage of Data
- User Authentication
- Uses OpenSSL 3 backend
- Secure memory wiping (except on PyPy)
- FIDO Passwordless
- Integration with popular web frameworks

Example usage of the Crypto Class

We have more sophisticated user authentication *available also.*

```
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 is availble. Though you would typically use *the User Auth API* as it is higher level and more advanced.

1.1.8 Key Management System

Though you would typically use *the User Auth API*, for simple and lower level activities, this module uses a custom Key Management System for symmetric encryption 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 encryption derived from user credentials.

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:

```
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

1.2 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:

```
import krypton

krypton.configs.defaultArgonOps = 3 # Number of iterations for Argon2id

krypton.configs.defaultPasswordResetArgonOps = 4 # Iteration count for Password Reset␣
↪codes

krypton.configs.defaultCryptoperiod = 2 # Approx. number of years for the cryptoperiod␣
```

(continues on next page)

(continued from previous page)

```
↪of a key

krypton.configs.defaultSessionPeriod = 15 # Number of minutes before a user Session is␣
↪destroyed.

krypton.configs.defaultLogRetentionPeriod = 43200 # Number of minutes to store login logs
```

Warning: When seting iteration counts for Argon2id, 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.

1.2.1 Databases

For the following settings please see *Databases*

```
krypton.configs.SQLDefaultCryptoDBpath = # for DB used by Crypto Class
krypton.configs.SQLDefaultKeyDBpath =  # for DB used by Key Management System
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.

```
## For both TOTP and FIDO
krypton.configs.APP_NAME = "ExampleApp" # name of your app

## The below are only needed for FIDO
krypton.configs.HOST_NAME = "example.com" # hostname, as seen by the user's browser
krypton.configs.ORIGIN = "https://example.com/" # again, as seen by the user's browser
```

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[10] to create an engine[11]. To add extra connection parameters, please refer to SQLAlchemy's and/or your chosen database's SQL Driver documentation.

[10] https://www.sqlalchemy.org/
[11] https://docs.sqlalchemy.org/en/14/core/engines.html

Since many database connection URLs are dificult (and sometimes impossible) to type as strings, you can use SQLAlchemy's API to create them. Please read *this section* for more information.

Please set these strings at:

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

1.3.1 Microsoft SQL Server

You need to install pyodbc[12] and Microsoft ODBC Driver for SQL Server[13]

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

```
"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:

```
&Encrypt=no
```

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

1.3.2 MySQL

Please install mysqlclient[14].

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

1.3.3 SQLite

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

1.3.4 PostgreSQL

Please install psycopg2[15].

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

[12] https://pypi.org/project/pyodbc/
[13] https://docs.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-ver16
[14] https://pypi.org/project/mysqlclient/
[15] https://pypi.org/project/psycopg2/

1.3.5 More complex connection strings

In places where more complex connection strings are needed, you can import SQLAlchemy's API to do it.

For example, to connect to MSSQL LocalDB (of course replace server and driver with your own values):

```
from sqlalchemy.engine import URL
connection_string = "DRIVER={ODBC Driver 18 for SQL Server};Server=(localdb)\
↪MSSQLLocalDB;Integrated Security=true'"
connection_url = URL.create("mssql+pyodbc", query={"odbc_connect": connection_string})
```

And then pass it to Krptn:

```
krypton.configs.SQLDefaultUserDBpath = connection_url
```

In the above example, `connection_url` will have the following content: `mssql+pyodbc:///?odbc_connect=DRIVER%3D%7BODBC+Driver+18+for+SQL+Server%7D%3BServer%3D%28localdb%29%5CMSSQLLocalDB%3BInt` This would be truly horrible to type hence SQLAlchemy's API is very handy.

1.4 User Authentication

Note:

- To use Authentication in a supported web framework please see *integrations*.
- There are many different errors that the User Authentication API may raise. To avoid surprises in production environments, we encourage everyone to test what errors are raised on different edge cases (e.g.: incorrect password, missing MFA token).

Warning:

- This does not protect you against brute force attacks - make sure to enable rate limiting on your host.
- User names are not encrypted.
- User objects 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:

```
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 delete the user, call the `.delete()` method:

```
model.delete()
```

1.4.1 Data storage

```
model = userModel.standardUser(userName="Test_UserName") # If user does not exist will␣
↪fail silently
# It will raise an error on model.login as below.
sessionKey = model.login(pwd="Test_Password") # See below what sessionKey is
```

To retreive and set user data as key-value pairs using `data` attribute of the object:

```
model.data.role = "admin" # "admin" would now be stored in the DB
print(model.data.role) # Would fetch "role" from the DB, and decrypt it
del model.data.role # Would delete it from the DB
```

Note: any data set under `model.data` will be encrypted & stored in the database. They will be persisted between new instances of user objects. This a very clean solution.

Alternatively, to retreive and set user data as key-value pairs using functions:

```
model.setData("test", "example") # test is the key and example is the value
data = model.getData("test") # Gives b"example". Would raise ValueError on error.
model.deleteData("test")
```

`model = userModel.standardUser(userName="Test_UserName")` can be replaced by `model = userModel.standardUser(userID=123)`. The `userID` can be obtained from `model.id` for a logged in user.

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

To avoid side channel attacks, `userModel.standardUser(userName="xyz")`{l=python} will fail silently if the user does not exist. An error will be raised on `login` instead.

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 only the stored values are encrypted. Keys are plaintext!! Avoid storing sensitive data in keys!

1.4.2 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.

Session keys are returned from `user.login` and `user.saveNewUser`.

To restore a session:

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

To set session expiry please see the *configurations*.

Sign out of all sessions

```
model.revokeSessions()
```

1.4.3 Logs

To control the retention period of logs, please see the *configurations*.

Once the user is logged in, it is easy to recall the login logs:

```
model.getLogs()
```

This returns a list, in the folowing format:

```
[[time: datetime, success: bool], ...]
```

It is a 2-dimensional list. The first item in the nested list, is always the DateTime object representing the time of the log. The second item in the nested list, is a Boolean representing the success status of the attempt.

As mentioned in ISO/IEC 27002, it is a good idea to display the past login attempts to the user. This way, the user can easily notice an attack.

1.4.4 Change Username

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

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

1.4.5 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*.

1.4.6 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")

```

(continues on next page)

(continued from previous page)

```
# Save value "data" with key "test" and allow access to user "Test_UserName"
user2.shareSet("test", "data", ["Test_UserName"])
value = model.shareGet("test") # returns b"data". Raises ValueError on error.
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 Krptn internals.

If the a user has used `shareSet` to send data to the same user multiple times with the same `name` (`name` as in the key/identifier for the data), you will get an error from the SQL Layer. 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.

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

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

r = model.encryptWithUserKey("data")
model.decryptWithUserKey(r) # Returns b"data"

## Here is the tricky part:

r = model.encryptWithUserKey("data", ["Test_UserName2"]) # Allow Test_UserName to␣
↪decrypt the data
model2.decryptWithUserKey(r[0][1], "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)`. `username` is the name of the user to who we need to provide `data`.

When decrypting, call `decryptUserKey`, on the user object corresponding to `username`, passing `data` as the first argument, and the encryptor's user name 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.

Unsafe Sharing

> **Warning:** Data which is shared using this method is not encrypted - hence any user can access it. This isn't necessarily a problem when storing data which is meant to be public. However, just note that you shouldn't store secret data.

Note: Because this data is shared accross all user's, its `name` must be unique across all users. If you attempt to set a data with a `name` which has already been taken by another user, you will get an error from the SQL layer.

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

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

model.setUnsafe("test", b"TEST_VALUE")

data = user2.getUnsafe("test")

model.deleteData("test")
```

1.4.7 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()
```

You can choose to email these codes to the user (therefore delegating trust to the email account), or any other way to handle this. It is also possible to split the codes in half, email the first half to a primary email, and send the 2nd half to secondary email - this way, for the hacker, they would need to compromise two emails instead of one. Aditionally, you may also decide to email a user when a recovery code is used - to help them prevent attacks.

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*[16] *and only later added it to the documentation also.*

For a live demo, there is a working version[17] 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[18] also!

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

1.6.1 What is WebAuthn?

According to the FIDO Alliance[19], 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.

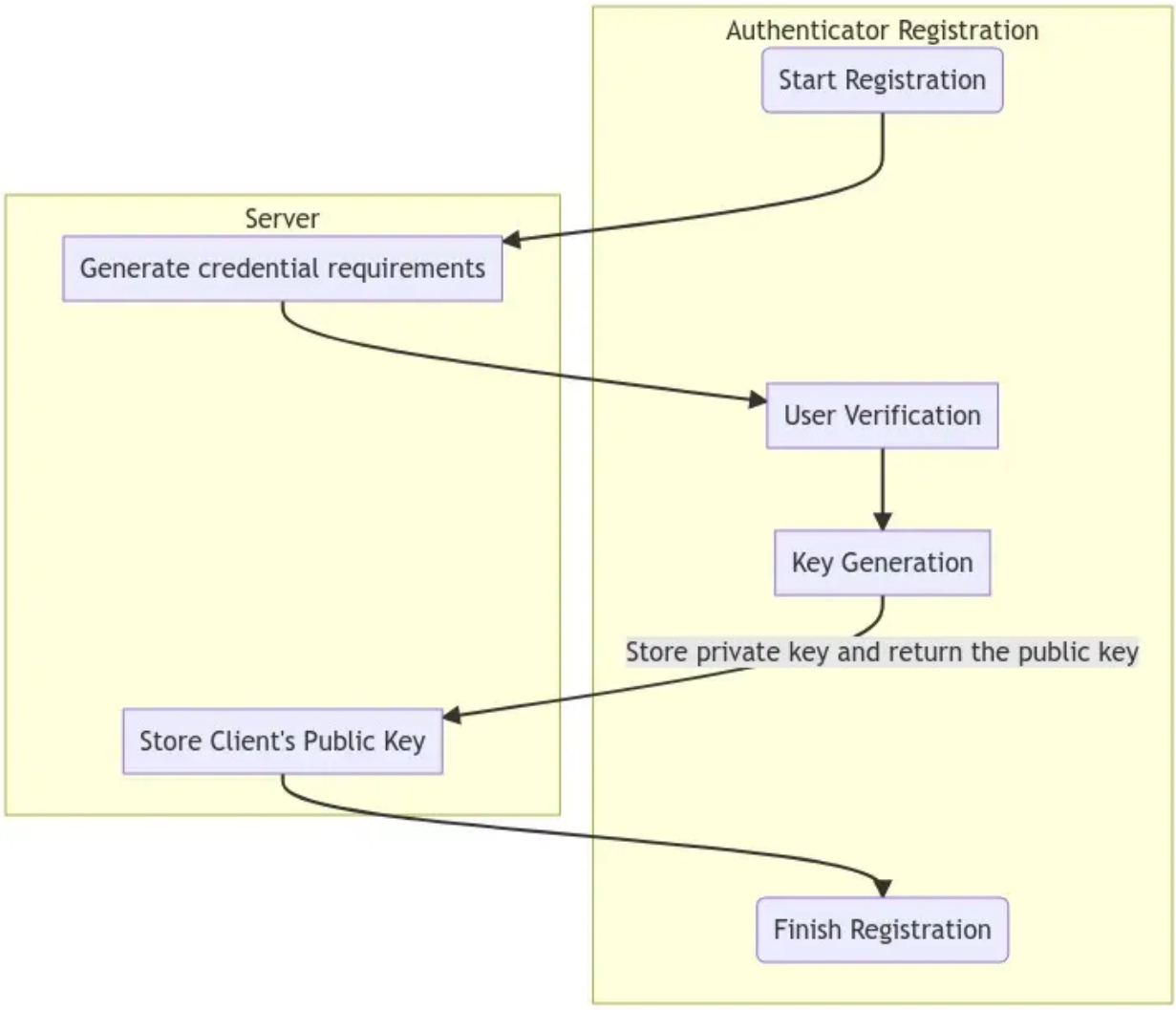
The server creates a challenge that the authenticator device signs using the credential. When the server verifies the signature, it can authenticate the user. Usually, the authenticator will require the user to verify their identity using a fingerprint, a pin code or other gesture.

Examples of authenticators include external hardware security keys and your device's trusted platform module.

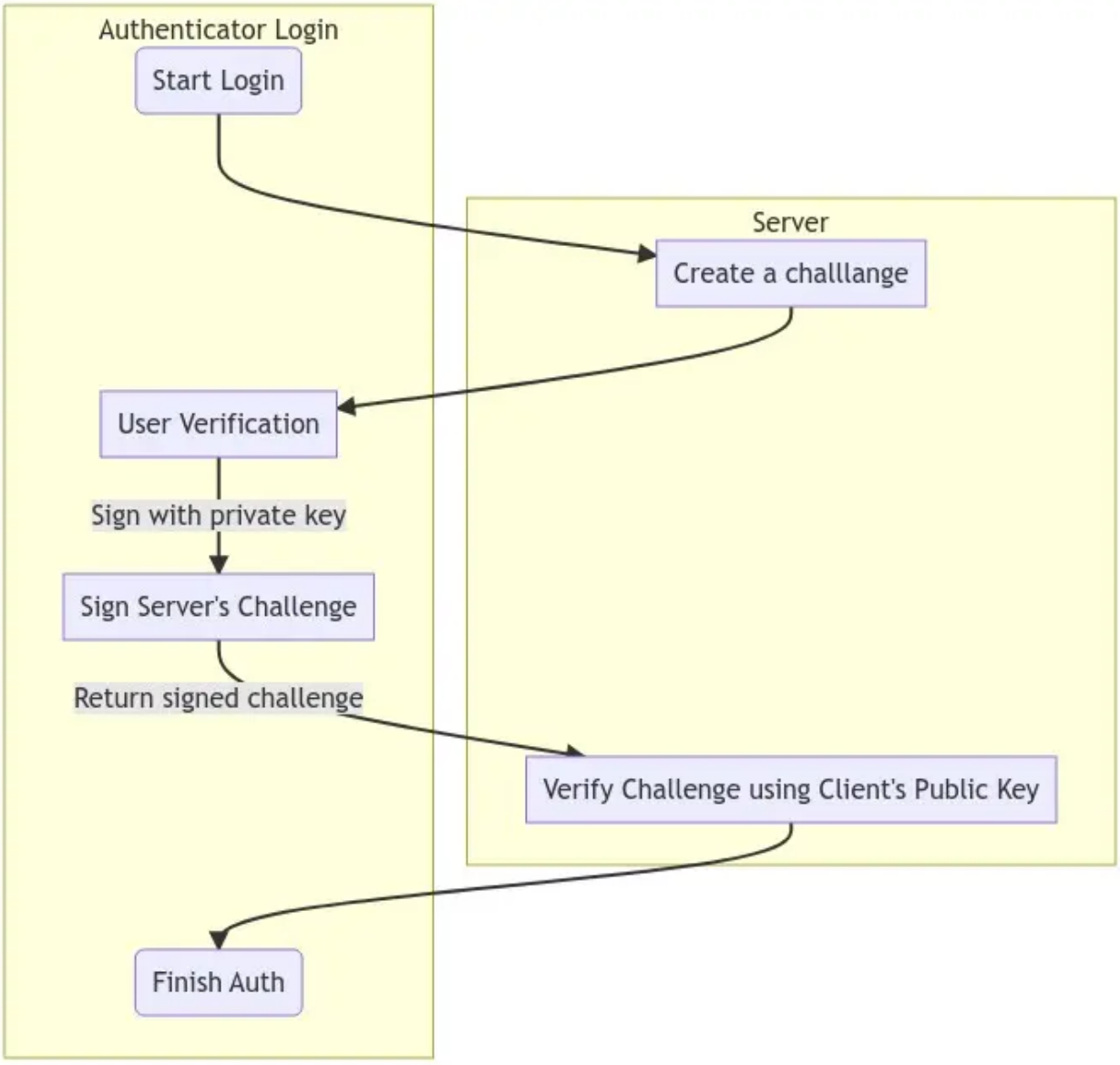
Here is the WebAuthn flowchart. Don't worry: you don't need to understand this to use it in your website :-)!

Here is the flowchart for registration:

[19] https://fidoalliance.org/



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[20]!

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[21] of our documentation

[20] https://python.org/
[21] 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);
```

(continues on next page)

(continued from previous page)

```
    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[22].

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:

```
import krypton

krypton.configs.APP_NAME = "ExampleApp" # name of your app
## The below are only needed for FIDO
krypton.configs.HOST_NAME = "example.com" # hostname, as seen by the user's browser
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.

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

Registration

Inside the `/fidoReg` endpoint:

```
from krypton.auth import users

model = users.standardUser(email)
key = model.login(...)
... # Standard Krptn login procedure
options = model.beginFIDOSetup()
```

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

Inside `/fidoFinishReg`:

```
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:

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

Login

Inside `getFidoLogin`:

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

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

Inside `fidoFinishLogin`:

```
model = users.standardUser(name_email)
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[23] 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[24].

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

[23] https://docs.krptn.dev/README-USER-AUTH.html
[24] 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[25] 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)

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

(continued from previous page)

```
        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[26] for more details on the above code.

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

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

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

Login

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

```
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:

```
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)

(continued from previous 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[27].

1.8 Django Integration

Warning: Because Krptn uses a strict[28] 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[29] 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.

[27] https://github.com/krptn/flaskExample
[28] https://www.krptn.dev/news/zero-knowledge/
[29] 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:

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'krypton.auth.django.middleware.kryptonLoginMiddleware' ## <-- Like here
]
```

Inside your views:

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

If you want to use type annotations, since the default Django HttpRequest is no longer valid, you can do something like this:

```
from krypton.auth.django.types import KrHttpRequest

def aRandomView(request: KrHttpRequest) -> HttpResponse:
    ## Your logic in here
```

After that type analysis in IDEs should work properly.

The only thing that KrHttpRequest does is override the `user` property in `django.http.HttpRequest` to point to Krptn's `djangoUser`.

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 to your needs.

We will briefly discuss how to create these forms.

Create User

```
from django import forms
from krypton.auth.django import users

class UserCreationForm(forms.Form):
    Password = forms.CharField(widget=forms.PasswordInput)
    userName = forms.CharField(label = "User Name")
    age = forms.CharField(label = "Age")
    def save(self, commit=True):
        user = users.djangoUser(None)
        token = user.saveNewUser(pwd=self.cleaned_data["Password"], name=self.cleaned_
↪data["userName"])
        user.setData("Age", self.cleaned_data["age"])
        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!*

```
response.set_cookie("_KryptonUserID", UserId)
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.

```
class LoginForm(forms.Form):
    userName = forms.CharField(label = "User Name")
    password = forms.CharField(widget=forms.PasswordInput)
    totp = forms.IntegerField(label = "TOTP")
    def save(self, commit=True):
        user = users.djangoUser(self.cleaned_data["userName"])
        token = user.login(pwd=self.cleaned_data["userName"], mfaToken=str(self.cleaned_
↪data["totp"]))
        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!*

```
response.set_cookie("_KryptonUserID", UserId)
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[30] on GitHub or reach out to us.

1.9 Crypto Class

Warning: Crypto Class is not thread-safe. Please only have one instance of the class per database. For a thread-safe solution, please use Krptn's proper *User Authentication*.

Here is a simple usage example:

```
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`.

```
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")
```

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

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.

Warning: First, please try to `pip install krptn` to avoid building from source. Only build from source if you need to (if there aren't any pre-built wheels on PyPI for your platform).

1.11.1 Building from source

To build Krptn, you will need:

- A C++ compiler
- Build tools required by VCPKG[31]

Checkout Git Submodules

After cloning the repo from GitHub[32] (and checking out your version using git tags), please checkout all submodules.

Setup Vcpkg

Note: In order for Vcpkg to work on Linux you certain developer tools installed[33].

To use Vcpkg on Windows you need Visual Studio.

To use Vcpkg on MacOS you need Xcode command line tools.

Vcpkg[34] is included as a git submodule. This is a C/C++ package manager which handles Krptn's C dependencies. Please install Vcpkg up by running `./vcpkg/bootstrap-vcpkg.bat{l=shell}` (for Windows) or `./vcpkg/bootstrap-vcpkg.sh{l=shell}` (for Unix-like systems).

Warning: When installing Vcpkg on aarch64 Linux, you may encounter errors since Vcpkg will build from source.

To ensure that you are able to build Vcpkg on aarch64, you need to ensure that you install `cmake`, `ninja`, and `export VCPKG_FORCE_SYSTEM_BINARIES=1{l=shell}`.

Note that Vcpkg requires a new version of `cmake` which is not available in most package managers. You will likely need to build `cmake` from source if you are using aarch64.

Warning: When installing (and using) Vcpkg on `musl` libc based distributions (so Alpine Linux), you need to `export VCPKG_FORCE_SYSTEM_BINARIES=1{l=shell}`.

[31] https://vcpkg.io/
[32] https://github.com/krptn/krypton
[33] https://github.com/microsoft/vcpkg#installing-linux-developer-tools
[34] https://vcpkg.io/

Install C packages

After setting up Vcpkg, you can install Krptn's dependencies from `vcpkg.json` by running `./vcpkg/vcpkg install --triplet {your-triplet}{l=shell}`.

Please substitute `{your-triplet}` with one of the following:

- `x64-windows-static` for Windows x64 builds
- `arm64-windows-static` for Windows ARM64 builds
- `x64-linux` for Linux x64 builds
- `arm64-linux` for Linux aarch64 builds
- `x64-osx` for MacOS Intel x64 builds
- `arm64-osx` for MacOS Apple Silicon builds

If you are using an platform which is not listed above, you need to manually find your triplet. We have not tested any other platforms apart from the above listed.

Install Krptn

```
pip install .
```

This will install Krptn.

1.12 CLI Interface

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

This commands cleans the database used by Krptn. Currently, it is not possible to erase custom databases (only the default database is erased). Please erase custom databases manually.

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
```

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

Older Krptn versions use OpenSSL's FIPS module for cryptography. Generally, this allows US government agencies to use this software - and also prevents us from accidentally using insecure ciphers. 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.

Note: this error can only happen when using old versions of Krptn. If you are using such an old version, **please upgrade to a newer version**. These **old versions are not supported**, and these instructions below will be removed eventually.

If you get this error, upgrade Krptn to a newer version or (if you can't) attempt the following fix:

1. Locate your Krptn installation by running `pip show krptn`.
2. Navigate to the location inidicated by pip.
3. Locate the OpenSSL install folder. By default, it is next to the Krptn installation named `kr-openssl-install`, but you may have changed it at install time. Inside that, locate:
   - on GNU/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 the OpenSSL install directory locate the openssl executable.
6. Inside the OpenSSL install directory 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 a 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.

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[35] about your problem.

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

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

Vulnerabilitis affecting the Krptn module

If your vulnerability affects the code that is installed on peoples devices when they `pip install krptn`, please fill out this form[36].

We also welcome vulnerabilities with no existing exploits. That means, for example, a use of an insecure cipher, that cannot be directly exploited, but is better fixed.

Other vulnerabilites

If your vulnerability is not to do with the Python package (e.g.: XSS vulnerability on our website), please follow the below instructions.

Email security vulnerabilities to security@krptn.dev.

Please make sure the following information is clearly stated:

- What components are affected?
- PoC - if any (please see our below notice)
- Recommendations on fixes, if any

We also welcome vulnerabilities with no existing exploits. That means, for example, a use of an insecure cipher, that cannot be directly exploited, but is better fixed.

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[37].

[36] https://github.com/krptn/krypton/security/advisories/new
[37] https://www.krptn.dev/news/

1.15 API Reference

1.15.1 krypton.basic

Basic security related classes.

class krypton.basic.**Crypto**(*keyDB: ~sqlalchemy.orm.session.Session = <sqlalchemy.orm.scoping.scoped_session object>*)

Crypto Class (see Documentation)

secureCreate(*data: ByteString*, *pwd: ByteString = None*, *_num: int = None*)

Store Encrypted Data

Arguments:
data – Plaintext data

Keyword Arguments:
pwd – Password To Decrypt (default: {None})

_num – Unless you know what this is, not good idea to set! Id to store in DB (default: {None})

Returns:
Integer to be passed to secureRead to return data

secureDelete(*num: int*, *pwd: ByteString = None*) → None

Delete Data set by secureCreate

Arguments:
num – Integer id of entry

Keyword Arguments:
pwd – Password (default: {None})

secureRead(*num: int*, *pwd: ByteString*)

Read data from secureCreate

Arguments:
num – Integer returned from secureCreate

pwd – Password set in secureCreate

Returns:
Plaintext data

secureUpdate(*num: int*, *new: ByteString*, *pwd: ByteString*)

Update Entry Set by secureCreate

Arguments:
num – Integer id of entry

new – New data to set

pwd – Password

class krypton.basic.**KMS**(*keyDB: ~sqlalchemy.orm.session.Session = <sqlalchemy.orm.scoping.scoped_session object>*)

They Key Management System

createNewKey(*name: str*, *pwd: ByteString = None*) → str

Create a new key and store it

Arguments:
name – Name of the Key

Keyword Arguments:
pwd – Password (default: {None})

Raises:
KeyError: If key with same name already exists

Returns:
The key as python bytes

getKey(*name: str*, *pwd: ByteString = None*, *force: bool = False*) → bytes

Get a Key

Arguments:
name – Name of the key to get

Keyword Arguments:
pwd – Password (default: {None})

force – Override Cryptoperiod Compliance errors (default: {False})

Raises:
ValueError: If the key does not exist

KeyManagementError: If the key has expired - set force=True to override

ValueError: If an unsupported cipher is used

ValueError: Wrong passwords were provided or the key was tampered with

Returns:
The key as python bytes

removeKey(*name: str*, *pwd: ByteString = None*) → None

Delete a Key

Arguments:
name – Name of the Key

Keyword Arguments:
pwd – Password (default: {None})

exception krypton.basic.**KeyManagementError**(**args: object*)

Error in Key Management System

For example, compliance issues

Arguments:
Exception – Inherits base Exception class

1.15.2 krypton.auth.users.bases

Contains Abstract Base Classes for user models. You can check this to see the declarations for functions.

exception krypton.auth.users.bases.**UserError**(**args: object*)

Exception to be raised when an error occurs in a user model.

class krypton.auth.users.bases.**user**

Base Class for User Models. You can check this to see whether a method is implemented in user models.

abstract decryptWithUserKey(*data: ByteString*, *sender=None*) → bytes

Decrypt data with user's key

Arguments:
data – Ciphertext

Keyword Arguments:
sender – If applicable sender's user name (default: {None})

Raises:
ValueError: if decryption fails

Returns:
Plaintext

abstract delete()

Delete a user

Returns:
None

abstract deleteData(*name: str*) → None

Delete key-value pair set by setData

Arguments:
name – The key to remove

abstract deleteUnsafe(*name: str*)

setUnsafe

Args:
name (str): Data identification

abstract disableMFA()

The method name says it all.

abstract enableMFA()

The method name says it all.

abstract enablePWDReset()

Enable Password Reset

Arguments:
key – The key needed to reset

abstract encryptWithUserKey(*data: ByteString*, *otherUsers: list[str]*) → bytes

Encrypt data with user's key

Arguments:
data – Plaintext

Keyword Arguments:
otherUsers – List of user nameswho can decrypt it (default: {None})

Returns:
List of tuples of form (user name, ciphertext, salt), check: https://docs.krptn.dev/README-USER-AUTH.html#encryption.

abstract generateNewKeys(*pwd: str*)

Regenerate Encryption keys

Arguments:
pwd – Password

abstract getData(*name: str*) → ByteString

Get value set by setData

Arguments:
name – the key

Raises:
AttributeError: if a value is not set

Returns:
The value

abstract getLogs() → list[list[datetime.datetime, bool]]

getLogs Get the login logs for the user

abstract getUnsafe(*name: str*)

setUnsafe

Args:
name (str): Data identification

abstract logFailure()

logFailure Log a login failure

abstract login(*pwd: str*, *mfaToken: str = None*, *fido: str = None*)

Log the user in

Keyword Arguments:
pwd – Password (default: {None})

otp – One-Time Password (default: {None})

fido – Fido Token (default: {None})

Raises:
UserError: Password is not set

Returns:
Session Key, None if user is not saved

abstract logout()

logout Logout the user and delete the current Session

abstract reload()

Reload encryption keys. Warning: previous keys are not purged!

abstract resetPWD(*key: str*, *newPWD: str*)

Reset Password

Arguments:
key – Key as provided to enablePWDReset

abstract restoreSession(*key: bytes*)

Resume session from key

Arguments:
key – Session Key

abstract revokeSessions()

Revoke all Sessions for this User

Raises:
UserError: If the user does not exist

abstract saveNewUser(*name: str*, *pwd: str*)

Save a new user

Arguments:
name – User Name

pwd – Password

Keyword Arguments:
fido – Fido Token (default: {None})

Raises:
ValueError: If user is already saved

abstract setData(*name: str*, *value: any*) → None

Store user data as a key-value pair

Arguments:
name – key

value – value

abstract setUnsafe(*name: str*, *data: ByteString*)

Args:
name (str): Data identification data (ByteString): data

abstract shareDelete(*name: str*) → None

shareDelete Delete data set by shareSet

Arguments:
name – Name of the data

abstract shareGet(*name: str*) → bytes

Get data set by shareSet

Arguments:
name – The "name of the data"

Raises:
ValueError: if decryption fails

Returns:
Decrypted data

abstract shareSet(*name: str*, *data: ByteString*, *otherUsers: list[str]*) → None

Set data readable by others

Arguments:
name – The "name" of the data

data – The data

otherUsers – List of usernames who should read it

krypton.auth.users.bases.**userExistRequired**(*func*)

User has to be saved in order to run this function

Arguments:
func – function

Raises:
UserError: If user is not saved

Returns:
inner1

1.15.3 krypton.auth.users.userModel

Provides User Models Note for developer's working on Krypton: this only contains user model cryptography.

class krypton.auth.users.userModel.**standardUser**(*userName: str = None*, *userID: int = None*)

User Model for Krypton Check documentation.

decryptWithUserKey(*data: ByteString*, *sender=None*) → bytes

Decrypt data with user's key

Arguments:
data – Ciphertext

Keyword Arguments:
sender – If applicable sender's user name (default: {None})

Raises:
ValueError: if decryption fails

Returns:
Plaintext

deleteData(*name: str*) → None

Delete key-value pair set by setData

Arguments:
name – The key to remove

deleteUnsafe(*name: str*)

setUnsafe

Args:
name (str): Data identification

encryptWithUserKey(*data: ByteString*, *otherUsers: list[int] = None*) → list[tuple[str, bytes, bytes]]

Encrypt data with user's key

Arguments:
data – Plaintext

Keyword Arguments:
otherUsers – List of user names who can decrypt it (default: {None})

Returns:
If otherUsers is None: ciphertext.

If otherUsers is not None: list of tuples (check https://docs.krptn.dev/README-USER-AUTH.html#encryption).

`generateNewKeys`(*pwd: str*)

Regenerate Encryption keys

Arguments:
pwd – Password

`getData`(*name: str*) → ByteString

Get value set by setData

Arguments:
name – the key

Raises:
ValueError: if decryption fails, or if a value is not set

Returns:
The value

`getUnsafe`(*name: str*)

setUnsafe

Args:
name (str): Data identification

`reload`()

Reload encryption keys. Warning: previous keys are not purged!

`setData`(*name: str*, *value: any*) → None

Store user data as a key-value pair

Arguments:
name – key

value – value

`setUnsafe`(*name: str*, *data: ByteString*)

Args:
name (str): Data identification data (ByteString): data

`shareDelete`(*name: str*) → None

shareDelete Delete data set by shareSet

Arguments:
name – Name of the data

`shareGet`(*name: str*) → bytes

Get data set by shareSet

Arguments:
name – The "name of the data"

Raises:
ValueError: if decryption fails or requested data does not exist

Returns:
Decrypted data

shareSet(*name: str*, *data: ByteString*, *otherUsers: list[str]*) → None

Set data readable by others

Arguments:
name – The "name" of the data

data – The data

otherUsers – List of usernames who should read it

1.15.4 krypton.auth.users.userModelBaseAuth

This module contains auth functions for models

class krypton.auth.users.userModelBaseAuth.**AuthUser**

Auth Logic for User Models

changeUserName(*newUserName: str*)

changeUserName Change the user's username

Arguments:
newUserName – The new username (string)

delete()

Delete a user

getLogs()

getLogs Get the login logs for the user

logFailure()

logFailure Log a login failure

login(*pwd: str*, *mfaToken: str = ''*, *fido: str = None*)

Log the user in

Keyword Arguments:
pwd – Password

otp – One-Time Password (default: {""})

fido – Fido Credentials (default: {None})

Raises:
UserError: Password is not set or wrong password is provided.

Returns:
Session Key

logout()

logout Logout the user and delete the current Session

restoreSession(*key*)

Resume sessoin from key

Arguments:
key – Session Key

revokeSessions()

Revoke all Sessions for this User

Raises:
UserError: If the user does not exist

saveNewUser(*name: str*, *pwd: str*) → bytes

Save a new user

Arguments:
name – User Name

pwd – Password

Raises:
ValueError: If user is already saved

1.15.5 krypton.auth.users.userModelMFAAuth

Extended auth logic

class krypton.auth.users.userModelMFAAuth.**MFAUser**

MFA for Krypton Users

beginFIDOSetup()

Being FIDO Registration

completeFIDOSetup(*response*)

Finish FIDO Setup

Arguments:
repsonse – The response from the client

disableMFA()

Disable TOTP based MFA

disablePWDReset()

Disbale PWD and revoke all codes

enableMFA()

Enable TOTP MFA

Returns:
base32 encoded shared secret, QR code string

enablePWDReset() → list[str]

Enable PWD Reset

Returns:
The recovery codes that unlock the account

getFIDOOptions()

Obtain FIDO options before Auth

Returns:
Fido Options as string, { “error”: “No keys availble” } if FIDO is not setup

removeFIDO()

Remove the FIDO Auth from Server

resetPWD(*key: str*, *newPWD: str*)

Reset Password

Arguments:
key – Key as provided to enablePWDReset

1.15.6 krypton.auth.factors

Different Auth Factors available inside krypton.

exception krypton.auth.factors.**AuthFailed**(**args: object*)

Exception to be raised when an error occures in a user model.

class krypton.auth.factors.**fido**

FIDO authentication support.

static authenticate(*cred_id*)

Begin user authentication

Arguments:
cred_id – The user's credential's id

Returns:
verification options, expected challange

static authenticate_verify(*challenge: bytes*, *credential_public_key*, *credentials*)

Complete Authentication

Arguments:
challenge – The expected challange from authenticate

credential_public_key – The user's public key

credentials – The credentials provided by the user

Returns:
True on success, False otherwise

static register(*userID: int*, *userName: str*)

Start FIDO auth registration process

Arguments:
userID – User's ID

userName – The User's username

Returns:
registration options and registration challenge

static register_verification(*credentials*, *challenge*)

Complete registration

Arguments:
credentials – The user's fido credentials, recieved from the browser

challenge – The expected challenge

Raises:
AuthError: registration failure

Returns:
credential id and credential public key

class krypton.auth.factors.**password**

Note: no need to create an object just call the methods directly. Simple password authentication.

1.) Hash the password with PBKDF2 and random salt.

2.) Decrypt the value in the table arg.

3.) Verify that the decryption was successfully authenticated.

4.) Return the encryption key.

static auth(*authTag: str*, *pwd: str*) → bytes

Authenticate against a tag

Arguments:
authTag – Tag

pwd – Password

Returns:
Encryption key if success, False otherwise

static getAuth(*pwd: str*)

Generate authentication tag for later use

Arguments:
pwd – Password

Returns:
Auth tag

class krypton.auth.factors.**totp**

Simple TOTP authentication

static createTOTP(*userName: str*)

Create parameters for TOTP Generate

Arguments:
userName – The username

Returns:
shared secret, base32 encoded shared secret, totp uri

static verifyTOTP(*secret: bytes*, *otp: str*) → bool

Verify TOTP

Arguments:
secret – The Shared secret

otp – The OTP

Returns:
True if success False otherwise

1.15.7 krypton.auth._utils

Utils to help code

krypton.auth._utils.**cleanUpSessions**(*session: scoped_session*, *userID: int = None*)

Cleanup Expired Session or Remove all sessions for a user

Arguments:
session – The database session to use

Keyword Arguments:
userID – The ID for which to delete all sessions (even if not expired) (default: {None})

1.15.8 krypton.base

Loads __CryptoLib and contains wrappers.

krypton.base.**base64decode**(*data: ByteString*) → ByteString

Decode base64

Arguments:
data – Base64 encoded string

Returns:
Base64 decoded bytes

krypton.base.**base64encode**(*data: ByteString*) → str

Base64 Encoding

Arguments:
data – Text to encode

Returns:
Base64 encoded string

krypton.base.**createECCKey**() → tuple[str, str]

create an Eliptic Curve Key

Encoded in P.E.M. format

Returns:
Returns a tuple like (privateKey:str, publicKey:str)

krypton.base.**createTOTPString**(*secret: bytes*, *user: str*) → str

Create a TOTP String that can be scanned by Auth Apps

Arguments:
secret – The shared secret

Returns:
The String to be converted to QR code

krypton.base.**decryptEcc**(*privKey: bytes*, *pubKey: bytes*, *data: ByteString*) → bytes

Decrypt data using public/private keys

Args:
privKey (bytes): Private Key pubKey (bytes): Public Key data (ByteString): Data to decrypt

Returns:
bytes: the decrypted data

krypton.base.**encryptEcc**(*privKey: bytes*, *pubKey: bytes*, *data: ByteString*) → bytes

Encrypt data using public/private keys

Args:
privKey (bytes): Private Key pubKey (bytes): Public Key data (ByteString): Data to encrypt

Returns:
bytes: the encrypted data

krypton.base.**genOTP**() → str

Generate an 20-digit OTP/PIN.

Returns:
The OTP/PIN as python string

krypton.base.**passwordHash**(*text: ByteString*, *salt: ByteString*, *opsLimit: int = 3*, *keylen: int = 32*) → bytes

Argon2id

Arguments:
text – Plain text salt – Salt

Keyword Arguments:
keylen – Len of key to return (default: {32}) opsLimit – Ops Limit for Argon2id

Returns:
The key as python bytes

krypton.base.**seal**(*data: ByteString*, *key: bytes*) → bytes

Encrypt Data for at rest storage

Arguments:
data – Plain text

key – 32-byte key

Returns:
Cipher text

krypton.base.**sleepOutOfGIL**(*seconds: int = 5*) → bool

Sleep for seconds while releasing the GIL.

Keyword Arguments:
seconds – Number of seconds to sleep for (default: {5})

Returns:
True

krypton.base.**unSeal**(*data: bytes*, *key: bytes*) → bytes

Decrypt Data from restEncrypt

Arguments:
data – Cipher text

key – 32-byte key

Returns:
Plain text

krypton.base.**verifyTOTP**(*secret: bytes*, *code: str*) → bool

Verify a 6-digit TOTP

Arguments:
secret – The shared secret

code – The code to verify

Returns:
True is success False otherwise

krypton.base.**zeromem**(*obj: ByteString*) → int

Set the byte/string to x00

WARNING! Improper use leads to severe memory corruption. Ensure you only use it with bytes and string objects. Also, on PyPy this function does nothing to avoid corruption.

Arguments:
obj – Object to do this on (bytes and str are supported!)

Returns:
Result from memset.

PYTHON MODULE INDEX

k

krypton.auth._utils, 44
krypton.auth.factors, 42
krypton.auth.users.bases, 35
krypton.auth.users.userModel, 38
krypton.auth.users.userModelBaseAuth, 40
krypton.auth.users.userModelMFAAuth, 41
krypton.base, 44
krypton.basic, 33

INDEX

A

auth() (*krypton.auth.factors.password static method*), 43
authenticate() (*krypton.auth.factors.fido static method*), 42
authenticate_verify() (*krypton.auth.factors.fido static method*), 42
AuthFailed, 42
AuthUser (*class in krypton.auth.users.userModelBaseAuth*), 40

B

base64decode() (*in module krypton.base*), 44
base64encode() (*in module krypton.base*), 44
beginFIDOSetup() (*krypton.auth.users.userModelMFAAuth.MFAUser method*), 41

C

changeUserName() (*krypton.auth.users.userModelBaseAuth.AuthUser method*), 40
cleanUpSessions() (*in module krypton.auth._utils*), 44
completeFIDOSetup() (*krypton.auth.users.userModelMFAAuth.MFAUser method*), 41
createECCKey() (*in module krypton.base*), 44
createNewKey() (*krypton.basic.KMS method*), 33
createTOTP() (*krypton.auth.factors.totp static method*), 43
createTOTPString() (*in module krypton.base*), 44
Crypto (*class in krypton.basic*), 33

D

decryptEcc() (*in module krypton.base*), 44
decryptWithUserKey() (*krypton.auth.users.bases.user method*), 35
decryptWithUserKey() (*krypton.auth.users.userModel.standardUser method*), 38
delete() (*krypton.auth.users.bases.user method*), 35
delete() (*krypton.auth.users.userModelBaseAuth.AuthUser method*), 40
deleteData() (*krypton.auth.users.bases.user method*), 35
deleteData() (*krypton.auth.users.userModel.standardUser method*), 38
deleteUnsafe() (*krypton.auth.users.bases.user method*), 35
deleteUnsafe() (*krypton.auth.users.userModel.standardUser method*), 38
disableMFA() (*krypton.auth.users.bases.user method*), 35
disableMFA() (*krypton.auth.users.userModelMFAAuth.MFAUser method*), 41
disablePWDReset() (*krypton.auth.users.userModelMFAAuth.MFAUser method*), 41

E

enableMFA() (*krypton.auth.users.bases.user method*), 35
enableMFA() (*krypton.auth.users.userModelMFAAuth.MFAUser method*), 41
enablePWDReset() (*krypton.auth.users.bases.user method*), 35
enablePWDReset() (*krypton.auth.users.userModelMFAAuth.MFAUser method*), 41
encryptEcc() (*in module krypton.base*), 44
encryptWithUserKey() (*krypton.auth.users.bases.user method*), 35
encryptWithUserKey() (*krypton.auth.users.userModel.standardUser method*), 38

F

fido (*class in krypton.auth.factors*), 42

G

generateNewKeys() (*krypton.auth.users.bases.user method*), 36
generateNewKeys() (*krypton.auth.users.userModel.standardUser*

method), 39
genOTP() (*in module krypton.base*), 45
getAuth() (*krypton.auth.factors.password static method*), 43
getData() (*krypton.auth.users.bases.user method*), 36
getData() (*krypton.auth.users.userModel.standardUser method*), 39
getFIDOOptions() (*krypton.auth.users.userModelMFAAuth.MFAUser method*), 41
getKey() (*krypton.basic.KMS method*), 34
getLogs() (*krypton.auth.users.bases.user method*), 36
getLogs() (*krypton.auth.users.userModelBaseAuth.AuthUser method*), 40
getUnsafe() (*krypton.auth.users.bases.user method*), 36
getUnsafe() (*krypton.auth.users.userModel.standardUser method*), 39

K

KeyManagementError, 34
KMS (*class in krypton.basic*), 33
krypton.auth._utils
    module, 44
krypton.auth.factors
    module, 42
krypton.auth.users.bases
    module, 35
krypton.auth.users.userModel
    module, 38
krypton.auth.users.userModelBaseAuth
    module, 40
krypton.auth.users.userModelMFAAuth
    module, 41
krypton.base
    module, 44
krypton.basic
    module, 33

L

logFailure() (*krypton.auth.users.bases.user method*), 36
logFailure() (*krypton.auth.users.userModelBaseAuth.AuthUser method*), 40
login() (*krypton.auth.users.bases.user method*), 36
login() (*krypton.auth.users.userModelBaseAuth.AuthUser method*), 40
logout() (*krypton.auth.users.bases.user method*), 36
logout() (*krypton.auth.users.userModelBaseAuth.AuthUser method*), 40

M

MFAUser (*class in krypton.auth.users.userModelMFAAuth*), 41
module
    krypton.auth._utils, 44
    krypton.auth.factors, 42
    krypton.auth.users.bases, 35
    krypton.auth.users.userModel, 38
    krypton.auth.users.userModelBaseAuth, 40
    krypton.auth.users.userModelMFAAuth, 41
    krypton.base, 44
    krypton.basic, 33

P

password (*class in krypton.auth.factors*), 43
passwordHash() (*in module krypton.base*), 45

R

register() (*krypton.auth.factors.fido static method*), 42
register_verification() (*krypton.auth.factors.fido static method*), 42
reload() (*krypton.auth.users.bases.user method*), 36
reload() (*krypton.auth.users.userModel.standardUser method*), 39
removeFIDO() (*krypton.auth.users.userModelMFAAuth.MFAUser method*), 41
removeKey() (*krypton.basic.KMS method*), 34
resetPWD() (*krypton.auth.users.bases.user method*), 36
resetPWD() (*krypton.auth.users.userModelMFAAuth.MFAUser method*), 41
restoreSession() (*krypton.auth.users.bases.user method*), 37
restoreSession() (*krypton.auth.users.userModelBaseAuth.AuthUser method*), 40
revokeSessions() (*krypton.auth.users.bases.user method*), 37
revokeSessions() (*krypton.auth.users.userModelBaseAuth.AuthUser method*), 40

S

saveNewUser() (*krypton.auth.users.bases.user method*), 37
saveNewUser() (*krypton.auth.users.userModelBaseAuth.AuthUser method*), 41
seal() (*in module krypton.base*), 45
secureCreate() (*krypton.basic.Crypto method*), 33
secureDelete() (*krypton.basic.Crypto method*), 33
secureRead() (*krypton.basic.Crypto method*), 33
secureUpdate() (*krypton.basic.Crypto method*), 33
setData() (*krypton.auth.users.bases.user method*), 37
setData() (*krypton.auth.users.userModel.standardUser method*), 39
setUnsafe() (*krypton.auth.users.bases.user method*), 37

setUnsafe() (*krypton.auth.users.userModel.standardUser method*), 39
shareDelete() (*krypton.auth.users.bases.user method*), 37
shareDelete() (*krypton.auth.users.userModel.standardUser method*), 39
shareGet() (*krypton.auth.users.bases.user method*), 37
shareGet() (*krypton.auth.users.userModel.standardUser method*), 39
shareSet() (*krypton.auth.users.bases.user method*), 37
shareSet() (*krypton.auth.users.userModel.standardUser method*), 40
sleepOutOfGIL() (*in module krypton.base*), 45
standardUser (*class in krypton.auth.users.userModel*), 38

T

totp (*class in krypton.auth.factors*), 43

U

unSeal() (*in module krypton.base*), 45
user (*class in krypton.auth.users.bases*), 35
UserError, 35
userExistRequired() (*in module krypton.auth.users.bases*), 38

V

verifyTOTP() (*in module krypton.base*), 45
verifyTOTP() (*krypton.auth.factors.totp static method*), 43

Z

zeromem() (*in module krypton.base*), 46