c't SESAM - python Documentation

Release 2.0-beta1

Johannes Merkert

Feb 14, 2018

Contents

c't SESAM is a password manager which calculates passwords from masterpasswords and domains using PBKDF2. There are compatible versions of this software for different platforms. This is the documentation for the console version written in Python.

Contents:

CHAPTER 1

Password generation

c't SESAM uses an encrypted secret to generate your passwords: the kgk (Key-Generation-Key). This trick enables you to change your masterpassword and makes sure that the secret used for the calculation of passwords is 64 bytes.

The kgk is stored and decrypted in the `KgkManager` class: The KGK manager stores the kgk and manages storage and encryption of kgk blocks.

class kgk_manager.**KgkManager**

New KgkManagers are uninitialized and need either a new kgk or get one by decrypting an existing one.

create_and_save_new_kgk_block(*kgk_crypter=None*)

Creates a fresh kgk block and saves it.

Parameters **kgk_crypter** (Crypter) –

Returns kgk block

Return type bytes

create_new_kgk()

Creates a new kgk. This overwrites the previous one.

Returns the new kgk

Return type bytes

decrypt_kgk(*encrypted_kgk*, *kgk_crypter=None*, *password=b''*, *salt=b''*)

Decrypts kgk blobs. If a crypter is passed it is used. If none is passed a new crypter is created with the salt and password. This takes relatively long. If the encrypted_kgk has a wrong length a new kgk is created.

Parameters

- **encrypted_kgk** (*bytes*) –
- **kgk_crypter** (Crypter) –
- **password** (*bytes*) –
- **salt** (*bytes*) –

fresh_iv2()
Creates a fresh iv for the settings encryption (iv2).

fresh_salt2()
Creates a fresh salt for the settings encryption (salt2).

get_encrypted_kgk()
Returns an encrypted kgk block.

Returns kgk block

Return type bytes

get_fresh_encrypted_kgk()
Returns a new encrypted kgk block with fresh salt2 and iv2. This does not create a new kgk.

Returns kgk block

Return type bytes

get_iv2()
Returns the iv2

Returns iv2

Return type bytes

get_kgk()
Returns the kgk.

Returns the kgk

Return type bytes

get_kgk_crypter(*password*, *salt*)
Creates a kgk crypter for the given credentials. This is a very expensive operation.

Parameters

- **password** (*bytes*) –
- **salt** (*bytes*) –

Returns a kgk crypter

Return type *Crypter*

get_kgk_crypter_salt()
Loads the public salt. If there is none it is created and stored.

Returns

get_salt2()
Returns the salt2

Returns salt2

Return type bytes

has_kgk()
Returns true if there is a kgk and a crypter.

Returns kgk state

Return type bool

reset()
Resets the kgk manager.

set_preference_manager(*preference_manager*)
Pass a preference manager to load and store settings locally

Parameters **preference_manager** (PreferenceManager) –

store_local_kgk_block()
Stores the local kgk block.

store_salt(*salt*)
Stores the salt using the preference manager.

Parameters **salt** (*bytes*) – the salt

update_from_blob(*password*, *blob*)
Updates the kgk from a remote data blob.

Parameters

- **password** (*bytes*) – the masterpassword
- **blob** (*bytes*) – the encrypted data

The encrypted kgk, and the settings are stored in the hidden file `.ctSESAM.pws` in your home directory. Reading and writing of this file is handled by the `PreferenceManager`: The preference manager handles the access to the settings file.

class preference_manager.**PreferenceManager**(*settings_file='/home/docs/.ctSESAM.pws'*)

Parameters **settings_file** (*str*) – Filename of the settings file. Defaults to PASSWORD_SETTINGS_FILE as defined in the source

get_kgk_block()
Reads the kgk_block.

Returns 112 bytes of kgk data

Return type bytes

get_salt()
Reads the salt.

Returns the salt

Return type bytes

get_settings_data()
Reads the settings data.

Returns encrypted settings

Return type bytes

read_file()
Read the settings file.

set_hidden()
Hides the settings file if possible.

store_kgk_block(*kgk_block*)
Writes the kgk_block into bytes 32 to 143.

Parameters **kgk_block** (*bytes*) – encrypted kgk data

store_salt(*salt*)
Writes the salt into the first 32 bytes of the file.

Parameters **salt** (*bytes*) – 32 bytes salt

store_settings_data(*settings_data*)
Writes the settings data after byte 144.

Parameters **settings_data** (*bytes*) – encrypted settings data

Passwords are generated with the `PasswordManager` class: Password manager. It's name is CtSesam because it produces passwords which are compatible to those created by other c't SESAM implementations.

class password_generator.**CtSesam**(*domain*, *username*, *kgk*, *salt=b'pepper'*, *iterations=4096*)
Calculates passwords from masterpasswords and domain names. You may set the salt and iteration count to something of your liking. If not set default values will be used.

Parameters

- **domain** (*str*) – the domain str
- **username** (*str*) – the username str
- **kgk** (*bytes*) – the kgk
- **salt** (*bytes*) – the salt
- **iterations** (*int*) – iteration count (should be 1 or higher, default is 4096)

generate(*setting*)
Generates a password string.

Parameters **setting** (PasswordSetting) – a setting object

Returns password

Return type str

CHAPTER 2

Managing settings

Settings are stored as `PasswordSetting` objects. Sets of password settings for a domain.

class password_setting.**PasswordSetting**(*domain*)

This saves one set of settings for a certain domain. Use a PasswordSettingsManager to save the settings to a file.

ask_for_input()

Displays some input prompts for the settings properties.

calculate_template(*use_lower_case=None*, *use_upper_case=None*, *use_digits=None*, *use_extra=None*)

Calculates a new template based on the character set configuration and the length.

Parameters

- **use_extra** (*bool*) – Gets this setting from the current template if None.
- **use_digits** (*bool*) – Gets this setting from the current template if None.
- **use_upper_case** (*bool*) – Gets this setting from the current template if None.
- **use_lower_case** (*bool*) – Gets this setting from the current template if None.

get_c_date()

Returns the creation date as a datetime object.

Returns the creation date

Return type datetime

get_character_set()

Returns the character set as a string.

Returns character set

Return type str

get_complexity()

Returns the complexity as a digit from 0 to 6. If the character selection does not match a complexity group -1 is returned.

Returns a digit from 0 to 6 or -1

Return type int

get_creation_date()
Returns the creation date as string.

Returns the creation date

Return type str

static get_default_character_set()
Returns the default character set.

Returns the default character set

Return type str

static get_digits_character_set()
Returns the digits character set.

Returns the digits character set

Return type str

get_domain()
Returns the domain name or another string used in the domain field.

Returns the domain

Return type str

get_extra_character_set()
Returns the set of special characters.

Returns set of special characters

Return type str

get_full_template()
Constructs a template string with digit and semicolon.

Returns template string

Return type str

get_iterations()
Returns the iteration count which is to be used.

Returns iteration count

Return type int

get_legacy_password()
Returns the legacy password if set or an empty string otherwise.

Returns the legacy password

Return type str

get_length()
Returns the desired password length.

Returns length

Return type int

static get_lower_case_character_set()
Returns the lower case character set.

Returns the lower case character set

Return type str

get_m_date()
Returns the modification date as a datetime object.

Returns the modification date

Return type datetime

get_modification_date()
Returns the modification date as string.

Returns the modification date

Return type str

get_notes()
Returns the notes.

Returns the notes

Return type str

get_salt()
Returns the salt.

Returns the salt

Return type bytes

get_template()
Returns the template without digit and semicolon.

Returns template

Return type str

static get_upper_case_character_set()
Returns the upper case character set.

Returns the upper case character set

Return type str

get_url()
Returns a url if there is one.

Returns the url

Return type str

get_username()
Returns the username or an empty string if there was no username.

Returns the username

Return type str

has_legacy_password()
Returns True if the legacy password is set.

Returns

Return type bool

has_username()
Returns True if the username is set.

Returns

Return type bool

is_synced()
Query if the synced flag is set. The flag switches to false if settings are changed.

Returns is synced?

Return type bool

load_from_dict(*loaded_setting*)
Loads the setting from a dictionary.

Parameters **loaded_setting** (*dict*) –

new_salt()
Creates a new salt for the setting.

set_complexity(*complexity*)
Sets the complexity by activating the appropriate character groups.

Parameters **complexity** (*int*) – 0, 1, 2, 3, 4, 5, 6 or 7

set_creation_date(*creation_date*)
Sets the creation date passed as string.

Parameters **creation_date** (*str*) –

set_domain(*domain*)
Change the domain string.

Parameters **domain** (*str*) – the domain

set_extra_character_set(*extra_set*)
Sets the set of special characters. This function does not check if these characters are in the whole character set.

Parameters **extra_set** (*str*) – string of special characters

set_iterations(*iterations*)
Sets the iteration count integer.

Parameters **iterations** (*int*) –

set_legacy_password(*legacy_password*)
Set a legacy password.

Parameters **legacy_password** (*str*) – a legacy password

set_modification_date(*modification_date=None*)
Sets the modification date passed as string.

Parameters **modification_date** (*str*) –

set_notes(*notes*)
Sets some note. This overwrites existing notes.

Parameters **notes** (*str*) –

set_salt(*salt*)
You should normally pass bytes as a salt. For convenience this method also accepts strings which get UTF-8 encoded and stored in binary format. If in doubt pass bytes.

Parameters **salt** (*bytes or str*) –

set_synced(*is_synced=True*)
Sets the synced state. Call this after syncing.

Parameters **is_synced** (*bool*) –

set_template(*full_template*)
Sets a template from a complete template string with digit and semicolon. This also preferences the template so other settings might get ignored.

Parameters **full_template** (*str*) – complete template string

set_url(*url*)
Sets a url.

Parameters **url** (*str*) – the url

set_username(*username*)
Set the username.

Parameters **username** (*str*) – the username

to_dict()
Returns a dictionary with settings to be saved.

Returns a dictionary with settings to be saved

Return type dict

The PasswordSettingsManager saves and manages the PasswordSetting objects. The PasswordSettingsManager handles the settings and manages storage and synchronization.

class password_settings_manager.**PasswordSettingsManager**(*preference_manager*)
Use this class to manage password settings. It can save the settings locally to the settings file and it can export them to be sent to a sync server.

Parameters **preference_manager** (PreferenceManager) – a PreferenceManager object

delete_setting(*setting*)
This removes the setting from the internal list. Call save_settings_to_file if you want to have the change saved to disk.

Parameters **setting** (PasswordSetting) – PasswordSetting object

get_domain_list()
This gives you a list of saved domains.

Returns a list of domain names

Return type [str]

get_export_data(*kgk_manager*)
This gives you a base64 encoded string of encrypted settings data (the blob).

Parameters **kgk_manager** (KgkManager) – kgk manager

Returns encrypted settings blob

Return type str

get_setting(*domain*)
This function always returns a setting. If no setting was stored for the given domain a new PasswordSetting object is created.

Parameters **domain** (*str*) – The "domain" is the identifier of a settings object.

Returns a setting object

Return type *PasswordSetting*

get_settings_as_dict()
Constructs a dictionary with a list of settings (no PasswordSetting objects but dicts) and a list of domain names of synced domains.

Returns a dictionary

Return type dict

static get_settings_crypter(*kgk_manager*)
Creates a settings crypter

Parameters **kgk_manager** (KgkManager) – a kgk manager

Returns Crypter for settings

Return type *Crypter*

load_local_settings(*kgk_manager*)
This loads the saved settings. It is a good idea to call this method the minute you have a kgk manager.

Parameters **kgk_manager** (KgkManager) – kgk manager

load_settings(*kgk_manager*, *password*, *no_sync=False*)
Loads settings from local file and from a sync server if possible.

Parameters

- **kgk_manager** (KgkManager) – kgk manager
- **password** (*str*) – the masterpassword
- **no_sync** (*bool*) – skip the sync update?

set_all_settings_to_synced()
Convenience function for marking all saved settings as synced. Call this after a successful update at the sync server.

set_setting(*setting*)
This saves the supplied setting only in memory. Call save_settings_to_file if you want to have it saved to disk.

Parameters **setting** (PasswordSetting) – the setting which should be saved

store_local_settings(*kgk_manager*)
This actually saves the settings to a file on the disk. The file is encrypted so you need to supply the password.

Parameters **kgk_manager** (KgkManager) – kgk manager

store_settings(*kgk_manager*)
Stores settings locally and remotely.

Parameters **kgk_manager** (KgkManager) – the kgk manager used for the encryption

update_from_export_data(*kgk_manager*, *blob*)
Call this method to pull settings from the sync server.

Parameters

- **kgk_manager** (KgkManager) – the kgk manager used for the decryption
- **blob** (*bytes*) – the export data

update_sync_server_if_necessary(*kgk_manager*)
Checks if the sync server needs to be updated. If necessary it does a push.

Parameters **kgk_manager** (KgkManager) – the kgk manager used for the encryption

It uses a Packer to compress data for storage and a Crypter to encrypt it. Compression with DEFLATE.

class packer.**Packer**
You do not need to create instances of this class because compress and decompress are both static methods.

static compress(*data*)
Compresses the given data with the DEFLATE algorithm. The first four bytes contain the length of the uncompressed data.

Parameters **data** (*bytes or str*) – uncompressed data

Returns compressed data

Return type bytes

static decompress(*compressed_data*)
Decompresses the given data. Please be aware that the first four bytes are the length of the uncompressed data.

Parameters **compressed_data** (*bytes*) – compressed data

Returns uncompressed data

Return type bytes

Encryption and decryption module.

class crypter.**Crypter**(*key_iv*)
Encrypt and decrypt with AES in CBC mode with PKCS7 padding. The constructor calculates the key from the given password and salt with PBKDF2 using HMAC with SHA512 and 32768 iterations.

static add_pkcs7_padding(*data*)
Adds PKCS7 padding so it can be divided into full blocks of 16 bytes.

Parameters **data** (*bytes*) – data without padding

Returns padded data

Return type bytes

static createIv()
Create a new ivj

Returns an iv with 16 bytes

Return type bytes

static createIvKey(*password*, *salt*, *iterations=32768*)
Creates a key for encrypting/decrypting kgk blocks.

Parameters

- **password** (*bytes*) – this is the kgk
- **salt** (*bytes*) – the salt2
- **iterations** (*int*) – an iteration count

Returns a key

Return type bytes

static createSalt()
Create a new salt.

Returns a salt with 32 bytes

Return type bytes

static create_key(*password*, *salt*, *iterations=1024*)
Creates a key for encrypting/decrypting settings.

Parameters

- **password** (*bytes*) – this is the kgk
- **salt** (*bytes*) – the salt2
- **iterations** (*int*) – an iteration count

Returns a key

Return type bytes

decrypt(*encrypted_data*)
Decrypts with AES in CBC mode with PKCS7 padding.

Parameters **encrypted_data** (*bytes*) – encrypted data

Returns decrypted data

Return type bytes

decrypt_unpadded(*encrypted_data*)
Decrypts with AES in CBC mode without padding. The data has to fit into blocks of 16 bytes.

Parameters **encrypted_data** (*bytes*) – encrypted data

Returns decrypted data

Return type bytes

encrypt(*data*)
Encrypts with AES in CBC mode with PKCS7 padding.

Parameters **data** (*bytes*) – data for encryption

Returns encrypted data

Return type bytes

encrypt_unpadded(*data*)
Encrypts with AES in CBC mode without padding. The data has to fit into blocks of 16 bytes.

Parameters **data** (*bytes*) – data for encryption

Returns encrypted data

Return type bytes

static remove_pkcs7_padding(*data*)
Removes the PKCS7 padding.

Parameters **data** (*bytes*) – padded data

Returns data without padding

Return type bytes

CHAPTER 3

Synchronisation

ctSESAM can synchronize your password settings with a ctSESAM-Server. The exact protocol is specified in the Wiki.

Basic communication part is implemented in the `Sync` class.

class sync.**Sync**(*server_url*, *username*, *password*, *cert_filename*)
Sync connection wrapper.

Parameters

- **server_url** (*str*) – https://my.server.domain/path/to/php/
- **username** (*str*) –
- **password** (*str*) –

pull()
Read the base64 encoded data from the sync server.

Returns base64 encoded data

Return type str

push(*data*)
Push data to the server. This overwrites data living there. Please pull and merge first.

Parameters **data** (*str*) – base64 encoded data

Returns was the push successful?

Return type bool

This class is wrapped by a `SyncManager` which handles the settings management for the server connection. Manages Sync connections.

class sync_manager.**SyncManager**
Synchronization manager. This initializes and stores settings and handles the Sync object.

ask_for_sync_settings()
Ask the user for sync settings: Asks for server-URL, username and password.

create_sync()
creates a sync object.

get_binary_sync_settings()
returns packed sync settings

Returns binary settings

Return type bytes

has_settings()
Returns true if pull or push are possible

Returns Are there settings?

Return type bool

load_binary_sync_settings(*data*)
loads sync settings

Parameters **data** (*bytes*) – packed json data of sync settings

pull()
pulls data from the sync server. Returns an empty string if no connection is possible.

Returns pulled base64 data

Return type str

push(*data*)
pushes data to the sync server. If the push fails an error message is displayed.

Parameters **data** (*str*) – base64 data

set_certificate(*certificate*)
Sets the certificate from a string in PEM format.

Parameters **certificate** (*str*) – certificate in PEM format

set_password(*password*)
Sets the password.

Parameters **password** (*str*) – the password

set_server_address(*url*)
Sets the url without ajax folder and php file names but with https://

Parameters **url** (*str*) – the url

set_username(*username*)
Sets the username.

Parameters **username** (*str*) – the username

CHAPTER 4

Tools

Functions for extracting domains.

domain_extractor.**extract_full_domain**(*url*)
Extracts the domain from an url

Parameters **url** (*str*) – Url with https:// and /some/path

Returns domain name without protocol or path

Return type str

domain_extractor.**extract_top_domain**(*url*)
Extracts the domain from an url. Subdomains are ignored

Parameters **url** (*str*) – Url with https:// and /some/path

Returns domain name without protocol, subdomains or path

Return type str

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

H

I

K

L

N

P

R

S

T

U