pygp

May 31, 2018

Contents

PyGP is a Python module that easy the use of Global Platform functions.

- Runs on Python 3.5.

CHAPTER 1

Assorted Info

- Issue tracker - Report bugs, ask questions, and share ideas here.
- GitHub project - Source code and issue tracking.

CHAPTER 2

Topics

2.1 Installation

pygp currently requires Python 3.5. There are no plans to add support for earlier versions of Python 2 or 3.

PyGP needs python cryptography package

2.2 Terminal functions

terminal(*readerName=None*)

Open the terminal using its name. If no terminal name is entered, we use the first 'available' reader found in the registry

Parameters **readerName** (*str*) – the name of the terminal to open.

Returns a dict mapping error codes with error status ERROR_STATUS_SUCCESS if no error occurs, error code and error message otherwise.

```
# error_status dict
{   error_status['errorStatus']  = ERROR_STATUS_FAILURE
    error_status['errorCode']    = 0x80301000
    error_status['errorMessage'] = "A APDU command can't be recognized as a valid␣
↪T=0 protocol Case 1-4 ISO7816-4 APDU"
}
```

Raises **ValueError** – if illegal parameter combination is supplied.

close()

Close the current selected terminal.

Returns a dict mapping error codes with error status ERROR_STATUS_SUCCESS if no error occurs, error code and error message otherwise.

```
# error_status dict
{   error_status['errorStatus']  = ERROR_STATUS_FAILURE
    error_status['errorCode']    = 0x80301000
    error_status['errorMessage'] = "A APDU command can't be recognized as a valid␣
↪T=0 protocol Case 1-4 ISO7816-4 APDU"
}
```

2.3 Card functions

atr()
Reset inserted card and get ATR.

Returns str the card ATR

Note: This command should be executed between opening a terminal and sending other card-related commands.

---

card()
Reset inserted card, get ATR and select the Issuer Security Domain

Returns str the card ATR

Note: This command should be executed between opening a terminal and sending other card-related commands.

---

change_protocol(*protocol*)
Set the protocol to select during the next card reset

Parameters **protocol** (*str*) – The protocol to select.

The value could be **'T0'** (T=0), **'T1'** (T=1), **'RAW'** (Raw mode) or **'Tx'** (T=1 or T=0))

2.4 Runtime functions

echo(*message*, *log_level=8*)
Log the message argument depending on the logging level

param str message the message to log.

param int log_level the logging level of this message

The logging level could be:

- DEBUG_LEVEL (0x04): All logging messages are displayed
- INFO_LEVEL (0x08): Information and error logging messages are displayed
- ERROR_LEVEL (0x10): Error logging messages are displayed

```
# echo the message only if the DEBUG_LEVEL is set
echo("my message", DEBUG_LEVEL)
```

get_key_in_repository(*keysetversion*, *key_identifier=None*)
Returns the list of Tuple (key value/Key type) stored into the off card key repository regarding their key version number and eventually their key identifier.

Parameters

- **keysetversion** (*str*) – the key set version.
- **key_identifier** (*str*) – the key identifier.

Returns list key_list A list of Tuple (key_version_number, key_id, key_type, key_value) matching the key version number

get_version()
Returns current PyGP API version

get_payload_list()
Returns the list of payload apdu.

Returns list payload_list: the list of apdu

get_total_execution_time()
Returns the total execution time.

Returns time in second.

last_response()
Returns the last card response as a haxadecimal string.

Returns str response: the last APDU card response.

Note: The response doesn't contain status word. Use *last_status()* to get it.

---

last_status()
Returns the last card status word as a haxadecimal string.

Returns str response: the last APDU status word.

set_log_mode(*loggingMode*, *file_path=None*)
Manages the logging capabilities.

Parameters

- **loggingMode** (*int*) – a mask value to configure logging capabilities

  The logging mode could be:

  – NONE (0x00): No log

  – CONSOLE_TRACE (0x01): Logging output is sent to sys.stdout, sys.stderr console.

  – **FILE_TRACE (0x02): Logging output is sent to a file specified by the file parameter.** If you didn't set file name then [script file name].log file will be created.

  The logging level could be:

  – DEBUG_LEVEL (0x04): All logging messages are displayed

  – INFO_LEVEL (0x08): Information and error logging messages are displayed

  – ERROR_LEVEL (0x10): Error logging messages are displayed

  The logging option could be:

  – APDU (0x20): APDU exchanges are displayed

  - APDU_MNGT (0x40): APDU exchanges due to the protocol (formelly T=0 protocol) are displayed
  - APDU_TIMING (0x80): APDU exchanges timing are displayed
- **file_path** (*str*) – the path of the logging file if the logging mode is set to FILE_TRACE

```
# set the logging mode to a file with debug logging level
set_log_mode(FILE_TRACE|DEBUG_LEVEL, "C:/log/myLoggingFile.txt")

# set the logging mode to a file with debug logging level, but without filename
set_log_mode(FILE_TRACE|DEBUG_LEVEL)
# set the logging mode to the console only with  information logging level and␣
↪APDU exchanges
set_log_mode(CONSOLE_TRACE|INFO_LEVEL|APDU)
```

set_payload_mode(*activate*)
Allows to store all apdu to send into a list in place of sending them to the card. The list containing apdus could be retreive by using the :func get_payload_list

Parameters **activate** (*bool*) – Activate the payload mode

set_key(**args*)
Put key definition into the off card key repository.

Parameters **args** (*str*) – key defined using a specific format: "KEY_VERSION_NUMBER/KET_ID/KEY_TYPE/KEY_VALUE"

Note: KEY_TYPE value could be: **DES-ECB**, **DES-CBC**, **AES**, **RSA-PRIV**, **RSA-PUB**

Note: If a key defined by its key version number is already present into the off card key repository, the new value will replace the old one.

---

set_start_timing()
Start to measure APDU transaction time. It will add the time consumption of each command.

sleep(*milliseconds*)
Delay execution for a given amount of time in millisecond unit.

Parameters **milliseconds** (*int*) – a number of milliseconds to delay execution process

stop_on_error(*value*)
Allows to stop the execution if an error occured.

Parameters **value** (*bool*) – True if execution should be stopped, False otherwise

2.5 Global Platform functions

auth(*enc_key=None*, *mac_key=None*, *dek_key=None*, *scp=None*, *scpi=None*, *keysetversion='21'*, *sequence_counter='000000'*, *securitylevel=0*)
Performs a complete authentication with the card using the specifiied key set, secure channel protocol,and security level for secure messaging.

Parameters

- **enc_key** (*str*) – The Session Encryption Key. If None (default) the off card repository key with the specified keyset number is used.
- **mac_key** (*str*) – The Secure Channel Message Authentication Code Key. If None (default) the off card repository key with the specified keyset number is used.
- **dek_key** (*str*) – The Key Encryption Key. If None (default) the off card repository key with the specified keyset number is used.
- **scp** (*str*) – The Session Channel Protocol to used. If None (default) the SCP returned by the card is used.
- **scpi** (*str*) – The Secure Channel Protocol Implementation to used. If None (default) the SCP implementation returned by the card is used.
- **keysetversion** (*str*) – The Key Set version to used.
- **sequence_counter** (*str*) – The current sequence counter. Use only in case of payload mode.
- **securitylevel** (*int*) – The security level of the secure messaging. Could be:
  - SECURITY_LEVEL_NO_SECURE_MESSAGING (0x00): No secure messaging expected.
  - SECURITY_LEVEL_C_MAC (0x01): C-MAC.
  - SECURITY_LEVEL_C_DEC_C_MAC (0x03): C-DECRYPTION and C-MAC.
  - SECURITY_LEVEL_R_MAC (0x10): R-MAC.
  - SECURITY_LEVEL_C_MAC_R_MAC (0x11): C-MAC and R-MAC.
  - SECURITY_LEVEL_C_DEC_C_MAC_R_MAC (0x13): C-DECRYPTION, C-MAC and R-MAC.
  - SECURITY_LEVEL_C_DEC_R_ENC_C_MAC_R_MAC (0x33): C-Decryption, C-MAC, R-Mac and R-Encryption.

Note: Depending of SCP mode used, some security level will not be available.

channel(*logical_channel=None*)

Selects the logical channel to use.

Parameters **logical_channel** (*int*) – The Logical Channel number (0..3) to select.

Note: You must track on your own, what channels are opened.

delete(*aid*, *exsw=None*)

Performs an application deletion by its AID.

Parameters

- **aid** (*str*) – The AID of the application to delete.
- **exsw** (*str*) – Set expected Status word.

delete_key(*key_version_number*, *key_identifier*, *exsw=None*)

Performs a key deletion identifies by its version number and its key identifier.

Parameters

- **key_version_number** (*str*) – The key version number.
- **key_identifier** (*str*) – The key identifier.
- **exsw** (*str*) – Set expected Status word.

Note: The key is not deleted into the the off card key repository.

delete_package(*aid*, *exsw=None*)
Performs a package and related application deletion by its AID.

Parameters

- **aid** (*str*) – The AID of the package to delete.
- **exsw** (*str*) – Set expected Status word.

extradite(*security_domain_AID*, *application_aid*, *identification_number=None*, *image_Number=None*, *application_provider_identifier=None*, *token_identifier=None*, *extraditeToken=None*)
Performs an application extradition into a Security Domain.

Parameters

- **security_domain_AID** (*str*) – The AID of the Security domain.
- **application_aid** (*str*) – The AID of the application to extradite.
- **identification_number** (*str*) – The Identification Number of the Security Domain with the Token Verification privilege.
- **image_Number** (*str*) – The Image Number of the Security Domain with the Token Verification privilege.
- **application_provider_identifier** (*str*) – The Application Provider identifier.
- **token_identifier** (*str*) – The Token identifier/number (digital signature counter).
- **extraditeToken** (*str*) – The extradition token (None by default).

ext_auth(*hostCryptogram*, *securitylevel=0*)
Performs an external authenticate using the specifiied cryptogram and security level to use during secure messaging.

Parameters

- **hostCryptogram** (*str*) – The off card host cryptogram retreived during the *init_update()* command.
- **securitylevel** (*int*) – The security level of the secure messaging. Could be:
  - SECURITY_LEVEL_NO_SECURE_MESSAGING (0x00): No secure messaging expected.
  - SECURITY_LEVEL_C_MAC (0x01): C-MAC.
  - SECURITY_LEVEL_C_DEC_C_MAC (0x03): C-DECRYPTION and C-MAC.
  - SECURITY_LEVEL_R_MAC (0x10): R-MAC.
  - SECURITY_LEVEL_C_MAC_R_MAC (0x11): C-MAC and R-MAC.
  - SECURITY_LEVEL_C_DEC_C_MAC_R_MAC (0x13): C-DECRYPTION, C-MAC and R-MAC.
  - SECURITY_LEVEL_C_DEC_R_ENC_C_MAC_R_MAC (0x33): C-Decryption, C-MAC, R-Mac and R-Encryption.

Note: Depending of SCP mode used during the *init_update()*, some security level will not be available.

get_certificate(*key_version_number*, *key_identifier*)
Retrieves a CERT.SD.ECKA from the SD.

Parameters

- **key_version_number** (*str*) – the key set version.
- **key_identifier** (*str*) – the key identifier.

Returns str data_response The response data containing the certificate.

get_cplc()
Get Card Production life cycle data and log it through the logger.

get_crs_status(*aid=None*, *tag_list=None*)
Retrieves the CRS registered Contactless Applications display information, the Lifecycle status and other information according to the given match/search criteria.

Parameters

- **aid** (*str*) – search criterion AID, if empty it will search previously selected CRS Application recursively.
- **tag_list** (*str*) – Indicates to the CRS Application how to construct the response data for matching search criteria.

get_data(*identifier*)
Retrieves a single card data object from the card identified by identifier. Some cards do not provide some data objects. Some possible identifiers are predefined. There is a convenience method *get_key_information()* to get the key information containing key set version, key index, key type and key length of the keys.

Parameters **identifier** (*str*) – the Two byte string with high and low order tag value for identifying card data object.

get_key_information()
Get key information for the currently selected Application and log it through the logger.

get_status_isd()
Get the AID, the life cycle state and the privileges of the Issuer Security Domain and log it through the logger.

get_status_applications()
Get the AID, the life cycle state and the privileges of all applications and log it through the logger.

get_status_executable_load_files()
Get the AID and the life cycle state of all executable load files and log it through the logger.

get_status_executable_load_files_and_modules()
Get the AID, the life cycle state and the modules AID of all executable load file and modules and then log it through the logger.

init_update(*enc_key=None*, *mac_key=None*, *dek_key=None*, *scp=None*, *scpi=None*, *keysetversion='21'*, *sequence_counter='000000'*)
Performs an initialize update using specifiied key set and secure channel protocol.

Parameters

- **enc_key** (*str*) – The Session Encryption Key. If None (default) the off card repository key with the specified keyset number is used.
- **mac_key** (*str*) – The Secure Channel Message Authentication Code Key. If None (default) the off card repository key with the specified keyset number is used.

- **dek_key** (*str*) – The Key Encryption Key. If None (default) the off card repository key with the specified keyset number is used.
- **scp** (*str*) – The Session Channel Protocol to used. If None (default) the SCP returned by the card is used.
- **scpi** (*str*) – The Secure Channel Protocol Implementation to used. If None (default) the SCP implementation returned by the card is used.
- **keysetversion** (*str*) – The Key Set version to used.
- **sequence_counter** (*str*) – The current sequence counter. Use only in case of payload mode.

Returns str hostCryptogram The off card host cryptogram to use into the *ext_auth()* function.

install(*make_selectable*, *executable_LoadFile_AID*, *executable_Module_AID*, *application_AID*, *application_privileges=[]*, *application_specific_parameters=None*, *install_parameters=None*, *install_token=None*)

Performs an install for install of an application into a Security Domain.

Parameters

- **make_selectable** (*boolean*) – True if the application must be selectable.
- **executable_LoadFile_AID** (*str*) – The AID of the load file package.
- **executable_Module_AID** (*str*) – The AID of the load file module.
- **application_AID** (*str*) – The AID of the application instance.
- **application_privileges** (*str*) – A list of *Application privileges* for the Application ([] by default).
- **application_specific_parameters** (*str*) – The application parameters (under tag C9)
- **install_parameters** (*str*) – The installation parameter (under tag EF).
- **install_token** (*str*) – The install token (None by default).

Note: example of application_privileges parameter : ["SD", "TP"] means privilege Security Domain with Trusted Path

---

install_load(*load_file_path*, *security_domain_aid*, *load_file_data_block_hash=None*, *load_parameters=None*, *load_token=None*)

Performs an install for load of a load file into a Security Domain.

Parameters

- **load_file_path** (*str*) – The path of the load file to install.
- **security_domain_aid** (*str*) – The AID of the Security domain.
- **load_file_data_block_hash** (*str*) – Mandatory if a Load Token is present or if the Load File contains one or more DAP Blocks.
- **load_parameters** (*str*) – The load file parameter (under tag EF, None by default).
- **load_token** (*str*) – The load token (None by default).

internal_auth(*key_version_number*, *key_identifier*, *crt_data*, *ePK_OCE_ECKA*)

Performs an internal authenticate command using the specified parameters.

Parameters

- **key_version_number** (*str*) – the key set version.
- **key_identifier** (*str*) – the key identifier.
- **crt_data** (*str*) – The data for key establishment.
- **ePK_OCE_ECKA** (*str*) – The Ephemeral public key of the OCE used for key agreement

Returns str data_response The response data containing the Ephemeral public key of the SD used for key agreement and the receipt.

load_file(*load_file_path*, *block_size=230*)
Performs a set of load commands using the load file parameter.

Parameters

- **load_file_path** (*str*) – The path of the load file to load.
- **block_size** (*int*) – The size of the data blocks.

ls()
Get the status of all executable load file, modules and applications and then log it through the logger.

manage_channel(*open_channel*, *logical_channel=None*)
Uses to open or close supplementaty logical channels.

Parameters

- **open_channel** (*boolean*) – wether open/close the channel. True means to open and False to close the channel.
- **logical_channel** (*int*) – The Logical channel number (0..3) to open/close.

mutual_auth(*key_version_number*, *key_identifier*, *crt_data*, *ePK_OCE_ECKA*)
Performs an mutual authenticate command using the specified parameters.

Parameters

- **key_version_number** (*str*) – the key set version.
- **key_identifier** (*str*) – the key identifier.
- **crt_data** (*str*) – The data for key establishment.
- **ePK_OCE_ECKA** (*str*) – The Ephemeral public key of the OCE used for key agreement

Returns str data_response The response data containing the Ephemeral public key of the SD used for key agreement and the receipt.

perform_security_operation(*key_version_number*, *key_identifier*, *certificate*)
The PERFORM SECURITY OPERATION command is used to send the OCE certificate to the SD. This is required as a precondition to the initiation of an SCP11 secure channel.

Parameters

- **key_version_number** (*str*) – the key set version.
- **key_identifier** (*str*) – the key identifier.
- **certificate** (*str*) – The certificate data.

put_key(*key_version_number*, *key_identifier=None*, *replace=False*)
Add or replace a new key identified by its version number and key identifier.

Parameters

- **key_version_number** (*str*) – The key version number.

- **key_identifier** (*str*) – The key identifier.
- **replace** (*bool*) – True if the key must be replaced, False otherwise.

Note: The key must be presented in the the off card key repository before set it into the card. see *set_key()*

put_scp_key(*key_version_number*, *replace=False*)
Add or replace scp keys identified by its version number.

Parameters

- **key_version_number** (*str*) – The key version number.
- **replace** (*bool*) – True if the key must be replaced, False otherwise.

Note: The key must be present in the the off card key repository before set it into the card. see *set_key()*

registry_update(*security_domain_AID*, *application_aid*, *application_privileges=[]*, *registry_parameter_field=None*, *install_token=None*)
Performs an install for registry update of an application into a Security Domain.

Parameters

- **security_domain_AID** (*str*) – The AID of the security domain.
- **application_AID** (*str*) – The AID of the application instance.
- **application_privileges** (*str*) – A list of *Application privileges* for the Application ([] by default).
- **registry_parameter_field** (*str*) – The application parameters (under tag EF)
- **install_token** (*str*) – The install token (None by default).

Note: example of application_privileges parameter : ["SD", "TP"] means privilege Security Domain with Trusted Path

select(*aid*, *channel=0*)
Performs an application selection by its AID.

Parameters

- **aid** (*str*) – The AID of the application to select
- **channel** (*int*) – The logical channel number. If None (default) channel number 00 will be used.

select_isd(*channel=0*)
Select the Issuer Security Domain using select by default APDU command. :param int channel: The logical channel number. If None (default) channel number 00 will be used.

send(*apdu*, *raw_mode=False*, *exsw=None*, *exdata=None*)
Sends an APDU Command according to the security level of the selected Security Domain

Parameters

- **apdu** (*str*) – The apdu command.
- **raw_mode** (*bool*) – If True apdu is sent without security level management.

- **exsw** (*str*) – Set expected Status word. exsw="6A88, 6982". There should be ',' as delimiter
- **exdata** (*str*) – Set expected response data. It compare R-APDU with response data.

set_sd_state(*lifeCycleState*, *aid*)
Modifies the security domain Life Cycle State.

Parameters

- **lifeCycleState** (*str*) – The new life cycle state.
- **aid** (*str*) – the AID of the target Application or Security Domain for which a Life Cycle change is requested.

set_app_state(*lifeCycleState*, *aid*)
Modifies the Application Life Cycle State.

Parameters

- **lifeCycleState** (*str*) – The new life cycle state.
- **aid** (*str*) – the AID of the target Application or Security Domain for which a Life Cycle change is requested.

set_crs_status(*status_type*, *status_value*, *aid*)
Modifies the card Life Cycle State or the Application Life Cycle State.

Parameters

- **status_type** (*str*) – Type of information shall be updated. (i.e. availability, priority order, etc.)
- **status_value** (*str*) – Updating value depends on the status type in 'status_type'
- **aid** (*str*) – the AID of the target CRS Application.

set_status(*cardElement*, *lifeCycleState*, *aid*)
Modifies the card Life Cycle State or the Application Life Cycle State.

Parameters

- **cardElement** (*str*) – Identifier for Load Files, Applications or the Card Manager.See constants values in *Set_Status Reference Control Parameter P1*.
- **lifeCycleState** (*str*) – The new life cycle state.
- **aid** (*str*) – the AID of the target Application or Security Domain for which a Life Cycle change is requested.

store_data(*data*)
Allows to transfer data to an Application or Security Domain processing the command. Depending of the data length, Multiple STORE DATA commands are used to send data to the Application or Security Domain by breaking the data into smaller components for transmission.

Parameters **data** (*str*) – data in a format expected by the Security Domain or the Application.

upload(*load_file_path*, *security_domain_aid*)
Performs a load of an application under the Security Domain

Parameters

- **load_file_path** (*str*) – The path of the load file to load.
- **security_domain_aid** (*str*) – The AID of the Security Domain.

Note: The install for install command is not send by this function.

upload_install(*load_file_path*, *security_domain_aid*, *executable_module_aid*, *application_aid*)
Performs a full load of an application under the selected Security Domain

Parameters

- **load_file_path** (*str*) – The path of the load file to load.
- **executable_LoadFile_AID** (*str*) – The AID of the load file package.
- **executable_Module_AID** (*str*) – The AID of the load file module.
- **application_AID** (*str*) – The AID of the application instance.

2.6 Cryptography functions

RANDOM(*bloc_size=8*)
Returns a block_size long random hexadecimal string

Parameters **bloc_size** (*int*) – the size in bye of the random string.

Returns str rand_str the random hexadecimal string.

ISO_9797_M1_Padding_left(*data*, *bloc_size=8*)
Performs a ISO_9797_M1 Padding by left. This padding is done in the following way: before the original data null bytes is added in order for the whole block to have a length in bytes that is a multiple of bloc_size If original data length is already a multiple of bloc_size, no padding is needed

Parameters

- **data** (*str*) – Hexadecimal string to pad.
- **bloc_size** (*int*) – the block size modulus

Returns str data_pad the padded data.

ISO_9797_M1_Padding(*data*, *bloc_size=8*)
Performs a ISO_9797_M1 Padding. This padding is done in the following way: after the original data null bytes is added in order for the whole block to have a length in bytes that is a multiple of bloc_size If original data length is already a multiple of bloc_size, no padding is needed

Parameters

- **data** (*str*) – Hexadecimal string to pad.
- **bloc_size** (*int*) – the block size modulus

Returns str data_pad the padded data.

ISO_9797_M2_Padding_left(*data*, *bloc_size=8*)
Performs a ISO_9797_M2 Padding by left. This padding is done in the following way: before the original data a byte '80' is added in order for the whole block to have a length in bytes that is a multiple of bloc_size If original data length is already a multiple of bloc_size, no padding is needed

Parameters

- **data** (*str*) – Hexadecimal string to pad.
- **bloc_size** (*int*) – the block size modulus

Returns str data_pad the padded data.

ISO_9797_M2_Padding(*data*, *bloc_size=8*)
Performs a ISO_9797_M2 Padding. This padding is done in the following way: after the original data a byte '80' and then null bytes are added. Then, in order for the whole block to have a length in bytes that is a multiple of bloc_size, null bytes can be added (byte '80' and null bytes are optional and not present in case the length is already a multiple of bloc_size)

Parameters

- **data** (*str*) – Hexadecimal string to pad.
- **bloc_size** (*int*) – the block size modulus

Returns str data_pad the padded data.

Remove_ISO_9797_M2_Padding(*data*)
Remove a ISO_9797_M2 Padding from an hexadecimal string .

Parameters

- **data** (*str*) – Hexadecimal string to unpad.
- **bloc_size** (*int*) – the block size modulus

Returns str data_pad the unpadded data.

RSA_PKCS_1_Padding(*data*, *key_size=1024*)
Performs a PKCS_1 Padding use to sign data with a RSA Private Key. The generated block of data is:

Leading	Block Type	Padding	Data
00	01 | FF...FF	00	D

Parameters

- **data** (*str*) – Hexadecimal string to pad.
- **key_size** (*int*) – the RSA key size that will be used to sign data.

Returns str padded_data the data padded

DES_CBC(*data*, *key*, *iv='0000000000000000'*)
Performs a DES CBC on the hexadecimal string using the specified key and the specified initial vector

Parameters

- **data** (*str*) – Hexadecimal string to cipher.
- **key** (*str*) – the key to use
- **iv** (*str*) – the initial vector (0000000000000000 by default)

Returns str data_ret the ciphered data.

DES_INV_CBC(*data*, *key*, *iv='0000000000000000'*)
Performs a DES-1 CBC on the hexadecimal string using the specified key and the specified initial vector

Parameters

- **data** (*str*) – Hexadecimal string to decipher.
- **key** (*str*) – the key to use
- **iv** (*str*) – the initial vector (0000000000000000 by default)

Returns str data_ret the deciphered data.

DES_ECB(*data*, *key*)
Performs a DES ECB on the hexadecimal string using the specified key

Parameters

- **data** (*str*) – Hexadecimal string to cipher.
- **key** (*str*) – the key to use

Returns str data_ret the ciphered data.

DES_INV_ECB(*data*, *key*)
Performs a DES-1 ECB on the hexadecimal string using the specified key

Parameters

- **data** (*str*) – Hexadecimal string to decipher.
- **key** (*str*) – the key to use

Returns str data_ret the deciphered data.

DES3_CBC(*data*, *key*, *iv='0000000000000000'*)
Performs a 3DES CBC on the hexadecimal string using the specified key and the specified initial vector

Parameters

- **data** (*str*) – Hexadecimal string to cipher.
- **key** (*str*) – the key to use
- **iv** (*str*) – the initial vector (0000000000000000 by default)

Returns str data_ret the ciphered data.

DES3_INV_CBC(*data*, *key*, *iv='0000000000000000'*)
Performs a 3DES-1 CBC on the hexadecimal string using the specified key and the specified initial vector

Parameters

- **data** (*str*) – Hexadecimal string to decipher.
- **key** (*str*) – the key to use
- **iv** (*str*) – the initial vector (0000000000000000 by default)

Returns str data_ret the deciphered data.

DES3_ECB(*data*, *key*)
Performs a 3DES ECB on the hexadecimal string using the specified key

Parameters

- **data** (*str*) – Hexadecimal string to cipher.
- **key** (*str*) – the key to use

Returns str data_ret the ciphered data.

DES3_INV_ECB(*data*, *key*)
Performs a 3DES-1 ECB on the hexadecimal string using the specified key

Parameters

- **data** (*str*) – Hexadecimal string to cipher.
- **key** (*str*) – the key to use

Returns str data_ret the ciphered data.

AES_CMAC (*data*, *key*)
Performs a AES CMAC on the hexadecimal string using the specified key

Parameters

- **data** (*str*) – Hexadecimal string to cipher.
- **key** (*str*) – the key to use

Returns str data_ret the ciphered data.

AES_ECB (*data*, *key*)
Performs a AES ECB on the hexadecimal string using the specified key

Parameters

- **data** (*str*) – Hexadecimal string to cipher.
- **key** (*str*) – the key to use

Returns str data_ret the ciphered data.

AES_CBC (*data*, *key*, *iv='00000000000000000000000000000000'*)
Performs a AES CBC on the hexadecimal string using the specified key and the specified initial vector

Parameters

- **data** (*str*) – Hexadecimal string to cipher.
- **key** (*str*) – the key to use
- **iv** (*str*) – the initial vector (00000000000000000000000000000000 by default)

Returns str data_ret the ciphered data.

AES_INV_ECB (*data*, *key*)
Performs a AES-1 ECB on the hexadecimal string using the specified key

Parameters

- **data** (*str*) – Hexadecimal string to decipher.
- **key** (*str*) – the key to use

Returns str data_ret the deciphered data.

AES_INV_CBC (*data*, *key*, *iv='00000000000000000000000000000000'*)
Performs a AES-1 CBC on the hexadecimal string using the specified key and the specified initial vector

Parameters

- **data** (*str*) – Hexadecimal string to decipher.
- **key** (*str*) – the key to use
- **iv** (*str*) – the initial vector (00000000000000000000000000000000 by default)

Returns str data_ret the deciphered data.

MAC33 (*data*, *key*, *iv='0000000000000000'*)
Performs a MAC33 on the hexadecimal string using the specified key and the specified initial vector

Parameters

- **data** (*str*) – Hexadecimal string to mac.
- **key** (*str*) – the key to use
- **iv** (*str*) – the initial vector (0000000000000000 by default)

Returns str data_ret the MAC33 of the data.

MAC3 (*data*, *key*, *padding='ISO_9797_M2'*, *iv='0000000000000000'*)
Performs a MAC3 on the hexadecimal string using the specified key and the specified initial vector

Parameters

- **data** (*str*) – Hexadecimal string to mac.
- **key** (*str*) – the key to use
- **padding** (*str*) – the padding method to use. Could be ISO_9797_M1, ISO_9797_M2 (default), None
- **iv** (*str*) – the initial vector (0000000000000000 by default)

Returns str data_ret the MAC3 of the data.

MAC (*data*, *key*, *iv='0000000000000000'*)
Performs a MAC on the hexadecimal string using the specified key and the specified initial vector

Parameters

- **data** (*str*) – Hexadecimal string to mac.
- **key** (*str*) – the key to use
- **iv** (*str*) – the initial vector (0000000000000000 by default)

Returns str data_ret the MAC of the data.

SHA1 (*data*)
Performs the SHA-1 algorithm on hexadecimal data

Parameters **data** (*str*) – Hexadecimal string.

Returns str data_ret the hash data.

SHA224 (*data*)
Performs the SHA-224 algorithm on hexadecimal data

Parameters **data** (*str*) – Hexadecimal string.

Returns str data_ret the hash data.

SHA256 (*data*)
Performs the SHA-256 algorithm on hexadecimal data

Parameters **data** (*str*) – Hexadecimal string.

Returns str data_ret the hash data.

SHA384 (*data*)
Performs the SHA-384 algorithm on hexadecimal data

Parameters **data** (*str*) – Hexadecimal string.

Returns str data_ret the hash data.

SHA512 (*data*)
Performs the SHA-512 algorithm on hexadecimal data

Parameters **data** (*str*) – Hexadecimal string.

Returns str data_ret the hash data.

MD5 (*data*)
Performs the MD5 algorithm on hexadecimal data

Parameters **data** (*str*) – Hexadecimal string.

Returns str data_ret the hash data.

generate_RSA_keys(*exponent*, *key_size=1024*)
RSA keys generation

Parameters

- **exponent** (*str*) – the public key exponent.
- **key_size** (*int*) – the key size in bit (1024 by default).

Returns tuple data_ret the private and public key objects

build_RSA_keys(*public_modulus*, *public_exponent*, *p*, *q*, *d*, *dmp1*, *dmq1*, *iqmp*)
Build RSA keys using specific values

Parameters

- **public_modulus** (*str*) – the public key modulus.
- **public_exponent** (*str*) – the public key exponent.
- **p** (*str*) – the private key large_modulus
- **q** (*str*) – the private key small_modulus
- **d** (*str*) – The private key exponent
- **dmp1** (*str*) – the key component d mod (p-1)
- **dmq1** (*str*) – the key component d mod (q-1)
- **iqmp** (*str*) – the key component q-1 mod p

Returns tuple data_ret the private and public key objects

RSA_signature(*message*, *private_key*, *padding_algorithm='PKCS1'*, *hash_algorithm='SHA1'*)
Performs a RSA signature on data using the padding and hash algorithm.

Parameters

- **message** (*str*) – the message to sign as hexadecimal string.
- **private_key** (*str*) – the private key object see *build_RSA_keys()* or *generate_RSA_keys()*
- **padding_algorithm** (*str*) – the padding to apply on data. Could be 'PKCS1', 'PSS' or 'OEAP'
- **hash_algorithm** (*str*) – the hash algorithm if the message you want to sign has already been hashed. Could be 'SHA1', 'SHA224', 'SHA256', 'SHA384' or 'SHA512'

Returns str data_ret the signature

RSA_verify(*message*, *signature*, *public_key*, *padding_algorithm='PKCS1'*, *hash_algorithm='SHA1'*)
Performs a RSA signature verification on data using the padding and hash algorithm.

Parameters

- **message** (*str*) – the message to sign as hexadecimal string.
- **signature** (*str*) – the signature of the message.
- **public_key** (*str*) – the public key object see *build_RSA_keys()* or *generate_RSA_keys()*

- **padding_algorithm** (*str*) – the padding to apply on data. Could be 'PKCS1', 'PSS' or 'OEAP'
- **hash_algorithm** (*str*) – the hash algorithm if the message you want to sign has already been hashed. Could be 'SHA1', 'SHA224', 'SHA256', 'SHA384' or 'SHA512'

Returns bool data_ret True if the signature is verified, False otherwize

ECDSA_signature (*message*, *private_key*, *hash_algorithm='SHA1'*)
Performs a ECDSA signature on data using hash algorithm.

Parameters

- **message** (*str*) – the message to sign as hexadecimal string.
- **private_key** (*str*) – the private key object see `build_ECDSA_keys()` or `generate_ECDSA_keys()`
- **hash_algorithm** (*str*) – the hash algorithm if the message you want to sign has already been hashed. Could be 'SHA1', 'SHA224', 'SHA256', 'SHA384' or 'SHA512'

Returns str data_ret the signature

ECDSA_verify (*message*, *signature*, *public_key*, *hash_algorithm='SHA1'*)
Performs a ECDSA signature verification on data using the hash algorithm.

Parameters

- **message** (*str*) – the message to sign as hexadecimal string.
- **signature** (*str*) – the signature of the message.
- **public_key** (*str*) – the public key object see *build_EC_keys()* or *generate_EC_keys()*
- **hash_algorithm** (*str*) – the hash algorithm if the message you want to sign has already been hashed. Could be 'SHA1', 'SHA224', 'SHA256', 'SHA384' or 'SHA512'

Returns bool data_ret True if the signature is verified, False otherwize

generate_ECDH_key_agreement (*private_key*, *public_key*)
Performs a ECDH key aggrement.

Parameters

- **private_key** (*str*) – the private key object see *build_EC_keys()* or *generate_EC_keys()*
- **public_key** (*str*) – the public key object see *build_EC_keys()* or *generate_EC_keys()*

Returns str data_ret The agreed key

generate_EC_keys (*curve_name='brainpoolP256r1'*)
EC keys generation

Parameters **curve_name** (*str*) – the name of the curve. Possible curve names:

Value	Description
“nistP521r1”	NIST P-521
“nistP256r1”	NIST P-256
“brainpoolP192r1”	Brainpool P-192 R1
“brainpoolP192t1”	Brainpool P-192 T1
“brainpoolP256r1”	Brainpool P-256 R1
“brainpoolP256t1”	Brainpool P-256 T1
“brainpoolP384r1”	Brainpool P-384 R1
“brainpoolP384t1”	Brainpool P-384 T1
“brainpoolP512r1”	Brainpool P-512 R1
“brainpoolP512t1”	Brainpool P-512 T1

Returns tuple data_ret the private and public key objects

build_EC_keys(*s*, *x*, *y*, *curve_name='brainpoolP256r1'*)
Build EC keys using parameters

Parameters

- **s** (*str*) – The private value
- **x** (*str*) – The affine x component of the public point
- **y** (*str*) – The affine y component of the public point
- **curve_name** (*str*) – the name of the curve. Possible curve names:

Value	Description
“nistP521r1”	NIST P-521
“nistP256r1”	NIST P-256
“brainpoolP192r1”	Brainpool P-192 R1
“brainpoolP192t1”	Brainpool P-192 T1
“brainpoolP256r1”	Brainpool P-256 R1
“brainpoolP256t1”	Brainpool P-256 T1
“brainpoolP384r1”	Brainpool P-384 R1
“brainpoolP384t1”	Brainpool P-384 T1
“brainpoolP512r1”	Brainpool P-512 R1
“brainpoolP512t1”	Brainpool P-512 T1

Returns tuple data_ret the private and public key objects

2.7 Utility functions

check_expected_data(*data*, *expected_data*)
Returns True if the data and expected_data are the same.

Parameters

- **data** (*str*) – a hexadecimal string
- **expected_data** (*str*) – a hexadecimal string or a string with a list of hexadecimal string with , as separator (“9000, 6Cxx, 6xxx”)

Returns bool : True if data matches with the expected data, False otherwize

Note: The character X or x could be set into expected data as a wildcard value

getBytes(*data*, *byteNumber*, *length=1*)
Returns the part of data string from byte number to length bytes

Parameters

- **data** (*str*) – a hexadecimal string
- **byteNumber** (*int*) – the start offset into the string
- **length** (*int*) – The bytes length to get

Returns str bytestr: the hexadecimal string

```
# get the string from byte 2 with a length of 3
astr = "3B65000  09C11 0101 03"
getBytes(astr, 2, 3) # returns  "650000"
```

getLength(*bytestr*, *length=1*)
Returns a string representing the length of the string as parameter on numberOfBytes bytes

Parameters

- **bytestr** (*str*) – a hexadecimal string
- **length** (*int*) – the number of byte expected for the string

```
getLength("A0A40002", 2) # returns "0004"
getLength("A0A40002", 1) # returns "04"
```

increment(*bytestr*, *value*)
This function increments an hexadecimal string with the integer value. The value could be an integer or a string.

Parameters

- **bytestr** (*str*) – a hexadecimal string
- **value** (*int*) – the value to increment

```
aStr = '01'
aInt = 0x03
newStr = increment ( aStr, aInt ) # returns  "04"
aInt = '03'
newStr = increment ( aStr, aInt ) # returns  "04"
```

intToHexString(*intValue*, *len=1*)
Returns a hexadecimal string representing an integer

Parameters

- **intValue** (*int*) – an integer value
- **len** (*int*) – the number of byte expected for the string

Returns str bytestr: the hexadecimal string

```
# get the string representation of the integer
aInt = 10
intToHexString ( aInt, 2 ) #returns  "000A"
```

lv(*bytestring*)
Returns a byte String representing the length content preceding by its length.

Parameters **bytestring** (*str*) – a hexadecimal string

Returns str bytestr: the hexadecimal string preceded by its length

Note: for byte string up to FF bytes, the length is coded with 2 bytes.

```
# get the string preceded by its length
astr = "3B65000  09C11 0101 03"
lv(astr) # returns  "093B6500009C11010103"
```

ber_lv(*bytestring*)
Returns a byte String representing the BER type length content preceding by its length.

Parameters **bytestring** (*str*) – a hexadecimal string

Returns str bytestr: the hexadecimal string preceded by its length

Note: short form consist of a single byte in which bit 8 is 0. (length: range is 0 ~ 127) long form. Initial octet, bit 8 is 1, and bits 1-7 encode the number of octets that follow.

```
# get the string preceded by its length
astr = "3B65000  09C11 0101 03"
ber_lv(astr) # returns  "093B6500009C11010103"

# the length of astr is longer than 127. Assume that length is 0x80
astr = "3B65000  09C11 0101 03..... 00"
ber_lv(astr) # returns  "81803B6500009C11010103....00"
```

der_lv(*bytestring*)
Returns a byte String representing the length content preceding by its length.

Parameters **bytestring** (*str*) – a hexadecimal string

Returns str bytestr: the hexadecimal string preceded by its length

Note: short form consist of a single byte (length: range is 0 ~ 127) long from consist of three octec, start with 0xFF(length: rnage is 0x0100 ~ 0xFFFF)

```
# get the string preceded by its length
astr = "3B65000  09C11 0101 03"
der_lv(astr) # returns  "093B6500009C11010103"

# the length of astr is longer than 255. Assume that length is 0x0100
astr = "3B65000  09C11 0101 03..... 00"
der_lv(astr) # returns  "FF01003B6500009C11010103....00"
```

remove_space(*bytestring*)
Removes all whitespace characters of a string.

Parameters **bytestring** (*str*) – The string to manage

Returns str bytestr: the string without whitepace characters

toByteArray(*byteString*)
Returns a list of bytes from a byte string

Parameters **bytestring** (*str*) – a hexadecimal string

Returns list list_byte: the list of bytes

```
# get the list of bytes from the hexadecimal string
astr = "3B65000  09C11 0101 03"
toByteArray( astr ) # returns  [ 0x3B, 0x65, 0x00, 0x00, 0x9C, 0x11, 0x01, 0x01,␣
↪0x03 ]
```

toHexString(*bytes=[]*)
Returns a hexadecimal string from a list of bytes

Parameters **bytes** (*list*) – a list of bytes

Returns str bytestr: the hexadecimal string

```
# get the string from the list of bytes
a_list = [ 0x3B, 0x65, 0x00, 0x00, 0x9C, 0x11, 0x01, 0x01, 0x03 ]
toHexString( a_list ) # returns  "3B6500009C11010103"
```

2.8 Constants values

2.8.1 Set_Status Reference Control Parameter P1

The following constants valurs could be used with the `set_status()` function for the parameter cardElement.

CARD_ELEMENT_ISD ('80') to indicate Issuer Security Domain.

CARD_ELEMENT_APPLICATION_AND_SSD ('40') to indicate Application or Supplementary Security Domain.

CARD_ELEMENT_SD_AND_APPLICATIONS ('60') to indicate Application or Supplementary Security Domain.

2.8.2 Application privileges

Value	Description
“CSA”	Contactless Self Activation
“CA”	Contactless Activation
“CLFDB”	Ciphered Load File Block
“RG”	Receipt Generation
“GS”	Global Service
“FA”	Final Application
“GR”	Global Registry
“GL”	Global Lock
“GD”	Global Delete
“TM”	Token Management
“AM”	Authorized Management
“TP”	Trusted Path
“MDAPV”	Mandated DAP Verification
“CVMM”	CVM management
“CR”	Card Reset
“CT”	Card Terminated
“CL”	Card Lock
“DM”	Delegated Management
“DAPV”	DAP Verification
“SD”	Security Domain

2.9 Release Notes

2.9.1 1.0.0

- Initial GitHub release.

Python Module Index

p

Index