GOST cryptographic function

Release 1.1.2

Drobotun Evgeny

Jul 09, 2021

Contents

1 Introduction 3
- 1.1 Overview . . . 3
  - 1.1.1 Features . . . 3
  - 1.1.2 Installation . . . 4
  - 1.1.3 Usage gosthash module . . . 4
  - 1.1.4 Usage gostcipher module . . . 5
  - 1.1.5 Usage gostsignature module . . . 6
  - 1.1.6 Usage gostrandom module . . . 8
  - 1.1.7 Usage gosthmac module . . . 8
  - 1.1.8 Usage gostpbkdf module . . . 8
- 1.2 License . . . 9
- 1.3 Source code . . . 9
- 1.4 Release History . . . 9

2 API documentation 11
- 2.1 API of the 'gostcrypto.gosthash' module . . . 11
  - 2.1.1 Introduction . . . 11
  - 2.1.2 API principles . . . 11
  - 2.1.3 Functions . . . 13
  - 2.1.4 Classes . . . 13
  - 2.1.5 Example of use . . . 16
- 2.2 API of the 'gostcrypto.gostcipher' module . . . 17
  - 2.2.1 Introduction . . . 17
  - 2.2.2 API principles . . . 18
  - 2.2.3 Constants . . . 20
  - 2.2.4 Functions . . . 20
  - 2.2.5 Classes . . . 22
  - 2.2.6 Example of use . . . 43
- 2.3 API of the 'gostcrypto.gostsignature' module . . . 45
  - 2.3.1 Introduction . . . 45
  - 2.3.2 Constants . . . 45
  - 2.3.3 Functions . . . 49
  - 2.3.4 Classes . . . 49
  - 2.3.5 Example of use . . . 53
- 2.4 API of the 'gostcrypto.gostrandom' module . . . 54
  - 2.4.1 Introduction . . . 54

2.4.2 Constants . . . 54
2.4.3 Functions . . . 54
2.4.4 Classes . . . 55
2.4.5 Example of use . . . 57
2.5 API of the 'gostcrypto.gosthmac' module . . . 57
2.5.1 Introductoon . . . 57
2.5.2 API principles . . . 58
2.5.3 Functions . . . 59
2.5.4 Classes . . . 60
2.5.5 Example of use . . . 63
2.6 API of the 'gostcrypto.gostpbkdf' module . . . 64
2.6.1 Introduction . . . 64
2.6.2 Functions . . . 64
2.6.3 Classes . . . 65
2.6.4 Example of use . . . 67

This guide describes how to use the gostcrypto Python package.

CHAPTER 1

Introduction

1.1 Overview

The package goscrypto implements various cryptographic functions defined in the State standards of the Russian Federation. All cryptographic functionalities are organized in modules; each modules is dedicated to solving a specific class of problems.

Package	Description
gostcrypto.gosthash	The module implements functions for calculating hash amounts in accordance with GOST R 34.11-2012.
gostcrypto.gostcipher	The module implements block encryption functions in accordance with GOST R 34.12-2015 and their use modes in accordance with GOST R 34.13-2015.
gostcrypto.gostsignature	The module implements the functions of forming and verifying an electronic digital signature in accordance with GOST R 34.10-2012.
gostcrypto.gostrandom	The module implements functions for generating pseudo-random sequences in accordance with R 1323565.1.006-2017.
gostcrypto.gosthmac	The module implements the functions of calculating the HMAC message authentication code in accordance with R 50.1.113-2016.
gostcrypto.gostpbkdf	The module implements the password-based key derivation function in accordance with R 50.1.111-2016.

1.1.1 Features

- Symmetric ciphers:
  - kuznechik
  - magma
- Traditional modes of operations for symmetric ciphers:
  - ECB

  - CBC
  - CFB
  - OFB
  - CTR
- Cryptographic hashes:
  - streebog 512
  - streebog 256
- Message Authentication Codes (MAC):
  - MAC
  - HMAC
- Asymmetric digital signatures:
  - (EC)DSA
- Key derivation:
  - PBKDF2

1.1.2 Installation

```
$ pip install gostcrypto
```

1.1.3 Usage gosthash module

Getting a hash for a string

```
import gostcrypto

hash_string = u'Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы'.encode(
↪'cp1251')
hash_obj = gostcrypto.gosthash.new('streebog256', data=hash_string))
hash_result = hash_obj.hexdigest()
```

Getting a hash for a file

Note: In this case the buffer_size value must be a multiple of the block_size value.

```
import gostcrypto

file_path = 'hash_file.txt'
buffer_size = 128
hash_obj = gostcrypto.gosthash.new('streebog512')
with open(file_path, 'rb') as file:
    buffer = file.read(buffer_size)
    while len(buffer) > 0:
```

(continues on next page)

(continued from previous page)

```
        hash_obj.update(buffer)
        buffer = file.read(buffer_size)
hash_result = hash_obj.hexdigest()
```

1.1.4 Usage gostcipher module

String encryption in ECB mode

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

plain_text = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_ECB,
                                       pad_mode=gostcrypto.gostcipher.PAD_MODE_1)

cipher_text = cipher_obj.encrypt(plain_text)
```

File encryption in CTR mode

Note: In this case the buffer_size value must be a multiple of the block_size value.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

init_vect = bytearray([
    0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce, 0xf0,
])

plain_file_path = 'plain_file.txt'
cipher_file_path = 'cipher_file.txt'
cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_CTR,
                                       init_vect=init_vect)
```

(continues on next page)

(continued from previous page)

```
buffer_size = 128

plain_file = open(plain_file_path, 'rb')
cipher_file = open(cipher_file_path, 'wb')
buffer = plain_file.read(buffer_size)
while len(buffer) > 0:
    cipher_data = cipher_obj.encrypt(buffer)
    cipher_file.write(cipher_data)
    buffer = plain_file.read(buffer_size))
```

Calculating MAC of the file

Note: In this case the buffer_size value must be a multiple of the block_size value.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

plain_file_path = 'plain_file.txt'
cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_MAC)

buffer_size = 128

plain_file = open(plain_file_path, 'rb')
buffer = plain_file.read(buffer_size)
while len(buffer) > 0:
    cipher_obj.update(buffer)
    buffer = plain_file.read(buffer_size)
mac_result = cipher_obj.digest(8)
```

1.1.5 Usage gostsignature module

Signing

```
import gostcrypto

private_key = bytearray([
    0x7a, 0x92, 0x9a, 0xde, 0x78, 0x9b, 0xb9, 0xbe, 0x10, 0xed, 0x35, 0x9d, 0xd3, 0x9a, 0x72, 0xc1,
    0x1b, 0x60, 0x96, 0x1f, 0x49, 0x39, 0x7e, 0xee, 0x1d, 0x19, 0xce, 0x98, 0x91, 0xec, 0x3b, 0x28,
])

digest = bytearray([
    0x2d, 0xfb, 0xc1, 0xb3, 0x72, 0xd8, 0x9a, 0x11, 0x88, 0xc0, 0x9c, 0x52, 0xe0, 0xee, 0xc6, 0x1f,
    0xce, 0x52, 0x03, 0x2a, 0xb1, 0x02, 0x2e, 0x8e, 0x67, 0xec, 0xe6, 0x67, 0x2b, 0x04, 0x3e, 0xe5,
```

(continues on next page)

(continued from previous page)

```
])

sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
    gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])

signature = sign_obj.sign(private_key, digest)
```

Verify

```
public_key = bytearray([
    0xfd, 0x21, 0xc2, 0x1a, 0xb0, 0xdc, 0x84, 0xc1, 0x54, 0xf3, 0xd2, 0x18, 0xe9, 0x04, 0x0b, 0xee,
    0x64, 0xff, 0xf4, 0x8b, 0xdf, 0xf8, 0x14, 0xb2, 0x32, 0x29, 0x5b, 0x09, 0xd0, 0xdf, 0x72, 0xe4,
    0x50, 0x26, 0xde, 0xc9, 0xac, 0x4f, 0x07, 0x06, 0x1a, 0x2a, 0x01, 0xd7, 0xa2, 0x30, 0x7e, 0x06,
    0x59, 0x23, 0x9a, 0x82, 0xa9, 0x58, 0x62, 0xdf, 0x86, 0x04, 0x1d, 0x14, 0x58, 0xe4, 0x50, 0x49,
])

digest = bytearray([
    0x2d, 0xfb, 0xc1, 0xb3, 0x72, 0xd8, 0x9a, 0x11, 0x88, 0xc0, 0x9c, 0x52, 0xe0, 0xee, 0xc6, 0x1f,
    0xce, 0x52, 0x03, 0x2a, 0xb1, 0x02, 0x2e, 0x8e, 0x67, 0xec, 0xe6, 0x67, 0x2b, 0x04, 0x3e, 0xe5,
])

signature = bytearray([
    0x4b, 0x6d, 0xd6, 0x4f, 0xa3, 0x38, 0x20, 0xe9, 0x0b, 0x14, 0xf8, 0xf4, 0xe4, 0x9e, 0xe9, 0x2e,
    0xb2, 0x66, 0x0f, 0x9e, 0xeb, 0x4e, 0x1b, 0x31, 0x35, 0x17, 0xb6, 0xba, 0x17, 0x39, 0x79, 0x65,
    0x6d, 0xf1, 0x3c, 0xd4, 0xbc, 0xea, 0xf6, 0x06, 0xed, 0x32, 0xd4, 0x10, 0xf4, 0x8f, 0x2a, 0x5c,
    0x25, 0x96, 0xc1, 0x46, 0xe8, 0xc2, 0xfa, 0x44, 0x55, 0xd0, 0x8c, 0xf6, 0x8f, 0xc2, 0xb2, 0xa7,
])

sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
    gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])

if sign_obj.verify(public_key, digest, signature):
    print('Signature is correct')
else:
    print('Signature is not correct')
```

Generating a public key

```
private_key = bytearray([
    0x7a, 0x92, 0x9a, 0xde, 0x78, 0x9b, 0xb9, 0xbe, 0x10, 0xed, 0x35, 0x9d, 0xd3, 0x9a, 0x72, 0xc1,
    0x1b, 0x60, 0x96, 0x1f, 0x49, 0x39, 0x7e, 0xee, 0x1d, 0x19, 0xce, 0x98, 0x91, 0xec, 0x3b, 0x28,
])

sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
    gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])

public_key = sign_obj.public_key_generate(private_key)
```

1.1.6 Usage gostrandom module

```
import gostcrypto

rand_k = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

random_obj = gostcrypto.gostrandom.new(32,
                                       rand_k=rand_k,
                                       size_s=gostcrypto.gostrandom.SIZE_S_256)
random_result = random_obj.random()
random_obj.clear()
```

1.1.7 Usage gosthmac module

Getting a HMAC for a string

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131516171819 1a1b1c1d1e1f')
data = bytearray.fromhex('0126bdb87800af214341456563780100')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key, data=data)
hmac_result = hmac_obj.digest()
```

Getting a HMAC for a file

Note: In this case the buffer_size value must be a multiple of the block_size value.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131516171819 1a1b1c1d1e1f')
file_path = 'hmac_file.txt'
buffer_size = 128
hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
with open(file_path, 'rb') as file:
    buffer = file.read(buffer_size)
    while len(buffer) > 0:
        hmac_obj.update(buffer)
        buffer = file.read(buffer_size)
hmac_result = hmac_obj.hexdigest()
```

1.1.8 Usage gostpbkdf module

```
import gostcrypto

password = b'password'
```

(continues on next page)

(continued from previous page)

```
salt = b'salt'

pbkdf_obj = gostcrypto.gostpbkdf.new(password, salt=salt, counter=4096)
pbkdf_result = pbkdf_obj.derive(32)
```

1.2 License

1.3 Source code

Package source code: https://github.com/drobotun/gostcrypto

1.4 Release History

1.1.2 (02.05.2020)

- Refactoring gostcipher module (changed the class hierarchy to remove code duplication)
- Refactoring gosthash module (remove code duplication)
- Fixed some minor bugs
- Updated docstring in accordance with the Google Python Style Guide

1.1.1 (20.04.2020)

- Use **kvargs in the new function with default parameters (gostrandom, gosthash, gosthmac, gostpbkdf)
- Add the ability to pass data to the new function from gosthmac
- Fixed some minor bugs in the gostrandom module

1.1.0 (15.04.2020)

- Refactoring code gostcipher module (changed the class structure)
- Each module has its own exception class added
- In the new function of the gostcipher module for MAC mode, it is now possible to pass data for MAC calculation, followed by calling the digest method without first calling the update method
- In the new function of the gosthash module, it is now possible to pass data for hash calculation, followed by calling the digest method without first calling the update method
- Added new exceptions for various conflict situations
- Fixed some minor bugs

1.0.0 (08.04.2020)

- First release of 'gostcrypto'

CHAPTER 2

API documentation

2.1 API of the 'gostcrypto.gosthash' module

2.1.1 Introduction

The module that implements the 'Streebog' hash calculation algorithm in accordance with GOST 34.11-2012 with a hash size of 512 bits and 256 bits. The module includes the GOST34112012 class, the GOSTHashError class and new function.

Note: You can hash only byte strings or byte arrays (for example b'hash_text' or bytearray([0x68, 0x61, 0x73, 0x68, 0x5f, 0x74, 0x65, 0x78, 0x74])).

2.1.2 API principles

The first message fragment for a hash can be passed to the new() function with the data parameter after specifying the name of the hashing algorithm ('streebog256' or 'streebog512'):

```
import gostcrypto

    hash_string = u'Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы'.encode(
↪'cp1251')
    hash_obj = gostrypto.gosthash.new('streebog256', data=hash_string)
```

The data argument is optional and may be not passed to the new function. In this case, the data parameter must be passed in the update() method, which is called after new():

```
import gostcrypto

    hash_string = u'Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы'.encode(
↪'cp1251')
```

(continues on next page)

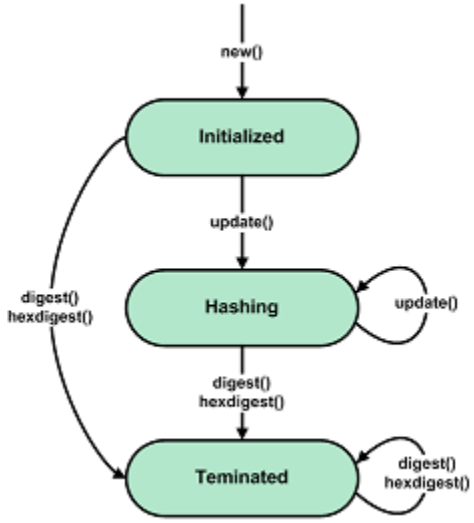

Fig. 1: Generic state diagram for a hash object

(continued from previous page)

```
    hash_obj = gostrypto.gosthash.new('streebog256')
    hash_obj.update(hash_string)
```

After that, the update method can be called any number of times as needed, with other parts of the message

Passing the first part of the message to the new () function, and the subsequent parts to the update() method:

```
import gostcrypto

    hash_obj = gostcrypto.gosthash.new('streebog512', data=b'first part message')
    hash_obj.update(b'second part message')
    hash_obj.update(b'third part message')
```

Passing the first part of the message and subsequent parts to the update() method:

```
import gostcrypto

    hash_obj = gostcrypto.gosthash.new('streebog512')
    hash_obj.update(b'first part message')
    hash_obj.update(b'second part message')
    hash_obj.update(b'third part message')
```

Hash calculation is completed using the digest() or hexdigest() method:

```
import gostcrypto

    hash_obj = gostcrypto.gosthash.new('streebog512')
    hash_obj.update(b'first part message')
    hash_obj.update(b'second part message')
    hash_obj.update(b'third part message')
    hash_result = hash_obj.digest()
```

2.1.3 Functions

new(name, **kwargs)

Creates a new hashing object and returns it.

```
import gostcrypto

    hash_string = u'Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы'.encode(
↪'cp1251')
    hash_obj = gostrypto.gosthash.new('streebog256', data=hash_string)
```

Arguments:

- name - the string with the name of the hashing algorithm `'streebog256'` for the GOST R 34.11-2012 algorithm with the resulting hash length of 32 bytes or `'streebog512'` with the resulting hash length of 64 bytes.

Keyword arguments:

- data - the data from which to get the hash (as a byte object). If this argument is passed to a function, you can immediately use the digest() (or hexdigest()) method to calculate the hash value after calling new(). If the argument is not passed to the function, then you must use the update() method before the digest() (or hexdigest()) method.

Return:

- New hashing object (as an instance of the GOST34112012 class).

Exceptions:

- GOSTHashError('unsupported hash type') - in case of invalid value name.

2.1.4 Classes

GOST34112012

Class that implements the hash calculation algorithm GOST 34.11-2012 ('Streebog').

Methods:

update(data)

Update the hash object with the bytes-like object.

```
import gostcrypto

hash_obj = gostcrypto.gosthash.new('streebog256')
hash_string = u'Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы'.encode(
↪'cp1251')
hash_obj.update(hash_string)
```

Arguments:

- data - the string from which to get the hash. Repeated calls are equivalent to a single call with the concatenation of all the arguments: m.update(a); m.update(b) is equivalent to m.update(a+b).

---

digest()

Returns the digest of the data passed to the update() method so far. This is a bytes object of size digest_size.

```
import gostcrypto

hash_obj = gostcrypto.gosthash.new('streebog256')
hash_string = u'Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы'.encode(
↪'cp1251')
hash_obj.update(hash_string)
result = hash_obj.digest()
```

Return:

- The digest value (as a byte object).

---

hexdigest()

Returns the hexadecimal digest of the data passed to the update() method so far. This is a double-sized string object (digest_size * 2).

```
import gostcrypto

hash_obj = gostcrypto.gosthash.new('streebog256')
hash_string = u'Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы'.encode(
↪'cp1251')
hash_obj.update(hash_string)
result = hash_obj.hexdigest()
```

Return:

- The digest value (as a hexadecimal string).

---

reset()

Resets the values of all class attributes.

```
import gostcrypto

hash_obj = gostcrypto.gosthash.new('streebog256')
hash_string_1 = u'Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы'.encode(
↪'cp1251')
hash_string_2 = bytearray([
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35,
    0x36, 0x37, 0x38, 0x39, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x31,
    0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
    0x38, 0x39, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x31, 0x32,
])

hash_obj.update(hash_string_1)
result_1 = hash_obj.digest()
hash_obj.reset()
hash_obj.update(hash_string_2)
result_2 = hash_obj.digest()
```

---

copy()

Returns a copy (“clone”) of the hash object. This can be used to efficiently compute the digests of data sharing a common initial substring.

```
import gostcrypto

hash_obj_1 = gostcrypto.gosthash.new('streebog256')
hash_obj_2 = hash_obj_1.copy()
```

Attributes:

digest_size

An integer value the size of the resulting hash in bytes. For the `'streebog256'` algorithm, this value is 32, for the `'streebog512'` algorithm, this value is 64.

```
import gostcrypto

hash_obj = gostcrypto.gosthash.new('streebog256')
hash_obj_digest_size = hash_obj.digest_size
```

---

block_size

An integer value the internal block size of the hash algorithm in bytes. For the 'streebog256' algorithm and the 'streebog512' algorithm, this value is 64.

```
import gostcrypto

hash_obj = gostcrypto.gosthash.new('streebog256')
hash_obj_block_size = hash_obj.block_size
```

---

name

A text string value the name of the hashing algorithm. Respectively 'streebog256' or 'streebog512'.

```
import gostcrypto

hash_obj = gostcrypto.gosthash.new('streebog256')
hash_obj_name = hash_obj.name
```

GOSTHashError

The class that implements exceptions.

```
import gostcrypto

hash_string = u'Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы'.encode(
↪'cp1251')
try:
    hash_obj = gostcrypto.gosthash.new('streebog256')
    hash_obj.update(hash_string)
except gostcrypto.gosthash.GOSTHashError as err:
    print(err)
else:
    result = hash_obj.digest()
```

Exception types:

- unsupported hash type - in case of invalid value name.
- invalid data value - in case where the data is not byte object.

2.1.5 Example of use

Getting a hash for a string

```
import gostcrypto

hash_string = u'Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы'.encode(
↪'cp1251')
hash_obj = gostcrypto.gosthash.new('streebog256', data=hash_string))
hash_result = hash_obj.hexdigest()
```

Getting a hash for a file

In this case the 'buffer_size' value must be a multiple of the 'block_size' value.

```
import gostcrypto

file_path = 'hash_file.txt'
buffer_size = 128
hash_obj = gostcrypto.gosthash.new('streebog512')
with open(file_path, 'rb') as file:
    buffer = file.read(buffer_size)
    while len(buffer) > 0:
        hash_obj.update(buffer)
        buffer = file.read(buffer_size)
hash_result = hash_obj.hexdigest()
```

2.2 API of the 'gostcrypto.gostcipher' module

2.2.1 Introduction

The module implements the modes of operation of block encryption algorithms "magma" and "kuznechik", described in GOST 34.13-2015. This document defines several encryption modes using block ciphers (ECB, CBC, CFB, OFB and CTR) and a message authentication code generation mode (MAC).

The module includes:

- GOST34122015Kuznechik: Class that implements the 'kuznechik' block encryption algorithm.
- GOST34122015Magma: Class that implements the 'magma' block encryption algorithm.
- GOST3413205: Base class of the cipher object.
- GOST3413205Cipher: Base class of the cipher object for implementing encryption modes.
- GOST3413205CipherPadding: Base class of the cipher object for implementing encryption modes with padding.
- GOST3413205CipherFeedBack: Base class of the cipher object for implementing encryption modes with feedback.
- GOST3413205ecb: Class that implements ECB mode of block encryption.
- GOST3413205cbc: Class that implements CBC mode of block encryption.
- GOST3413205cfb: Class that implements CFB mode of block encryption.
- GOST3413205ofb: Class that implements OFB mode of block encryption.
- GOST3413205ctr: Class that implements CTR mode of block encryption.
- GOST34132015mac: Class that implements MAC mode.

- GOSTCipherError: The exception class.
- new: Function that creates a new encryption object and returns it.

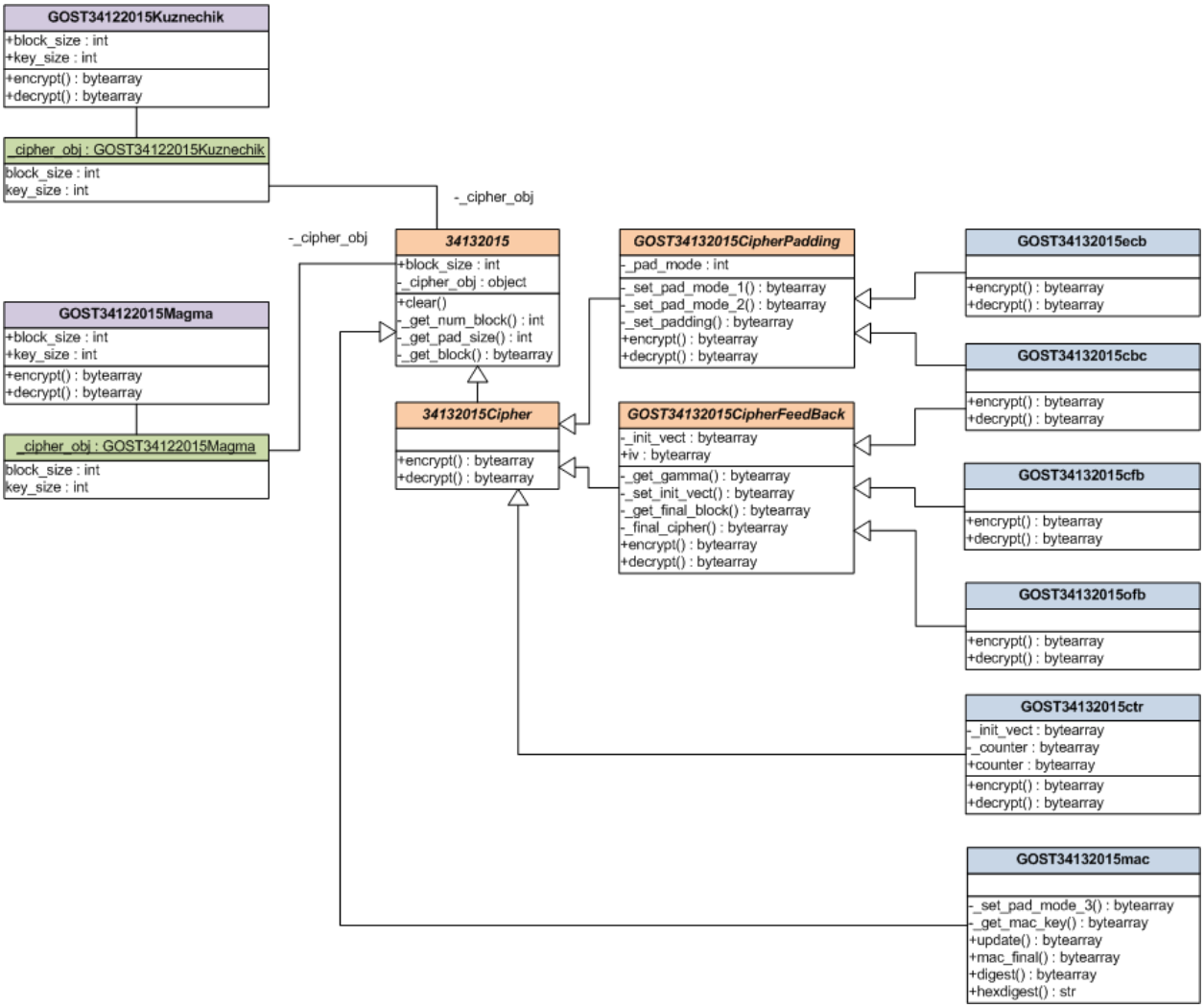

Fig. 2: Class hierarchy of the gostcipher module

Note: You can encrypting only byte strings or byte arrays (for example b'hash_text' or bytearray([0x68, 0x61, 0x73, 0x68, 0x5f, 0x74, 0x65, 0x78, 0x74])).

2.2.2 API principles

The cipher mode (ECB, CBC, CFB, OFB and CTR)

You create an instance of the cipher object by calling the new() function. The first parameter is the name of the algorithm ('kuznechik' or 'magma'), the second parameter is always the cryptographic key, and the third is the encryption mode. You can (and sometimes should) pass additional cipher or mode parameters to new() (for example, the initialization vector value or padding mode).

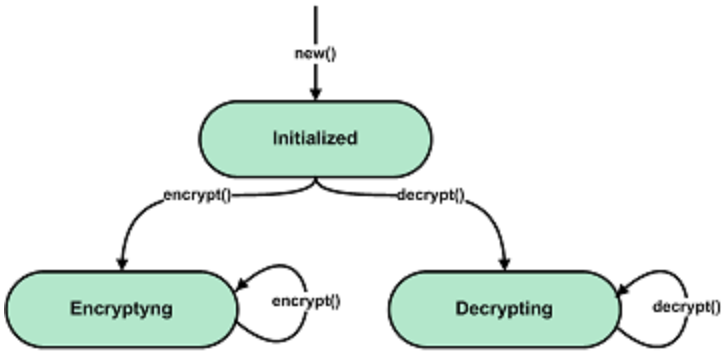

Fig. 3: General state diagram for the cipher object for ECB, CBC, CFB, OFB, and CTR modes

To encrypt data, you call the cipher object's encrypt() method with plaintext as an input parameter. The method returns a fragment of ciphertext.

To decrypt data, you call the cipher object's decrypt() method with cipher textas an input parameter. The method returns a fragment of plaintext.

The message authentication code algorithm (MAC)

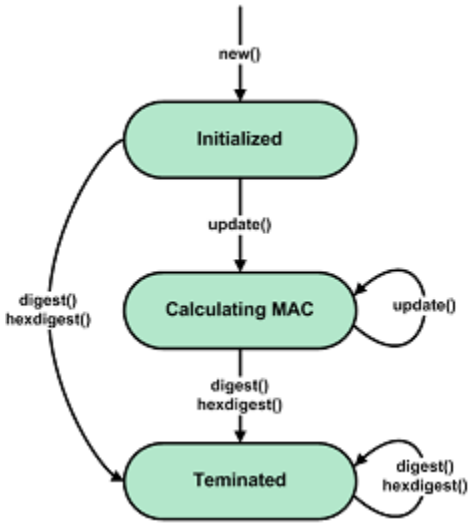

Fig. 4: General state diagram for the cipher object for MAC mode

The first message fragment for the MAC calculation can be passed to the new() function as the keyword argument data. This argument is optional for the new function. If necessary, the first fragment of the message can be passed to the update() method (in this case, the message is not passed to the new function).

Passing the first message fragment to the new() function and calling the update() method with the second message fragment:

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

mac_obj = gostcrypto.gostcipher.new('kuznechik',
                                    key,
                                    gostcrypto.gostcipher.MODE_MAC,
                                    data=b'first part message)
mac_obj.update(b'second part message')
```

Passing the first and second message fragments to the update() method:

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

mac_obj = gostcrypto.gostcipher.new('kuznechik',
                                    key,
                                    gostcrypto.gostcipher.MODE_MAC)
mac_obj.update(b'first part message')
mac_obj.update(b'second part message')
```

The MAC calculation is completed by calling the digest() (or hexdigest()) method.

2.2.3 Constants

- MODE_ECB - Electronic Codebook mode.
- MODE_CBC - Cipher Block Chaining mode
- MODE_CFB - Cipher Feedback mode
- MODE_OFB - OutputFeedback mode
- MODE_CTR - Counter mode
- MODE_MAC - Message Authentication Code algorithm
- PAD_MODE_1 - Padding a message according to procedure 1 (it can be used in ECB and CBC modes).
- PAD_MODE_2 - Padding a message according to procedure 2 (it can be used in ECB and CBC modes).

2.2.4 Functions

new(algorithm, key, mode, **kwargs)

The function creates a new cipher object and returns it.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])
cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_ECB,
                                       pad_mode=PAD_MODE_2)
```

Arguments:

- algorithm - the string with the name of the ciphering algorithm of the GOST R 34.12-201 (`'kuznechik'` with block size 128 bit or `'magma'` with block size 64 bit).
- key - byte object with 256-bit encryption key.
- mode - mode of operation of the block encryption algorithm (valid value: MODE_CBC, MODE_CFB, MODE_CTR, MODE_ECB, MODE_OFB or MODE_MAC).

Keywords arguments:

- init_vect - byte object with initialization vector. Used in CTR, OFB, CBC and CFB modes. For CTR mode, the initialization vector length is equal to half the block size. For CBC, OFB and CFB modes, it is a multiple of the block size. The default value is None.
- data - the data from which to get the MAC (as a byte object). For MODE_MAC mode only. If this argument is passed to a function, you can immediately use the digest() (or hexdigest()) method to calculate the MAC value after calling new(). If the argument is not passed to the function, then you must use the update() method before the digest() (or hexdigest()) method.
- pad_mode - padding mode for ECB and CBC modes. The default value is PAD_MODE_1.

Return:

- New cipher object (as an instance of one of the classes: GOST34132015ecb, GOST34132015cbc, GOST34132015cfb, GOST34132015ofb, GOST34132015ctr or GOST34132015mac).

Exceptions:

- GOSTCipherError('unsupported cipher mode') - in case of unsupported cipher mode (is not MODE_ECB, MODE_CBC, MODE_CFB, MODE_OFB, MODE_CTR or MODE_MAC).
- GOSTCipherError('unsupported cipher algorithm') - in case of invalid value algorithm.
- GOSTCipherError('invalid key value') - in case of invalid key value (the key value is not a byte object (bytearray or bytes) or its length is not 256 bits).
- GOSTCipherError('invalid padding mode') - in case padding mode is incorrect (for MODE_ECB and MODE_CBC modes).
- GOSTCipherError('invalid initialization vector value') - in case initialization vector value is incorrect (for all modes except MODE_ECB mode).

- GOSTCipherError('invalid text data'): in case where the text data is not byte object (for MODE_MAC mode).

2.2.5 Classes

GOST34122015Kuznechik

Class that implements block encryption in accordance with GOST 34.12-2015 with a block size of 128 bits ("Kuznechik"). An instance of this class is passed as the _cipher_obj attribute to the base class GOST34132015 when the "Kuznechik" encryption algorithm is selected.

Initialization parameter:

- key - byte object with 256-bit encryption key.

Methods:

encrypt(block)

Encrypting a block of plaintext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

plain_block = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
])

cipher_obj = gostcrypto.gostcipher.GOST34122015Kuznechik(key)
cipher_block = cipher_obj.encrypt(plain_block)
```

Arguments:

- block - the block of plaintext to be encrypted (the block size is 16 bytes).

Return:

- The block of ciphertext (as a byte object).

---

decrypt(block)

Decrypting a block of ciphertext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

cipher_block = bytearray([
    0x7f, 0x67, 0x9d, 0x90, 0xbe, 0xbc, 0x24, 0x30, 0x5a, 0x46, 0x8d, 0x42, 0xb9, 0xd4, 0xed, 0xcd,
])

cipher_obj = gostcrypto.gostcipher.GOST34122015Kuznechik(key)
plain_block = cipher_obj.encrypt(cipher_block)
```

Arguments:

- block - the block of ciphertext to be decrypted (the block size is 16 bytes).

Return:

- The block of plaintext (as a byte object).

clear()

Clearing the values of iterative encryption keys.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

cipher_block = bytearray([
    0x7f, 0x67, 0x9d, 0x90, 0xbe, 0xbc, 0x24, 0x30, 0x5a, 0x46, 0x8d, 0x42, 0xb9, 0xd4, 0xed, 0xcd,
])

cipher_obj = gostcrypto.gostcipher.GOST34122015Kuznechik(key)
plain_block = cipher_obj.encrypt(cipher_block)
cipher_obj.clear()
```

Attributes:

block_size

An integer value the internal block size of the cipher algorithm in bytes. For the 'Kuznechik' algorithm this value is 16.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

cipher_obj = gostcrypto.gostcipher.GOST34122015Kuznechik(key)
block_size = cipher_obj.block_size
```

key_size

An integer value the cipher key size.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

cipher_obj = gostcrypto.gostcipher.GOST34122015Kuznechik(key)
key_size = cipher_obj.key_size
```

GOST34122015Magma

Class that implements block encryption in accordance with GOST 34.12-2015 with a block size of 64 bits (“Magma”). An instance of this class is passed as the _cipher_obj attribute to the base class GOST34132015 when the “Magma” encryption algorithm is selected.

Initialization parameter:

- key - byte object with 256-bit encryption key.

Methods:

encrypt(block)

Encrypting a block of plaiintext.

```
import gostcrypto

key = bytearray([
    0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff
])

plain_block = bytearray([
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10,
])
```

(continues on next page)

(continued from previous page)

```
cipher_obj = gostcrypto.gostcipher.GOST34122015Magma(key)
cipher_block = cipher_obj.encrypt(plain_block)
```

Arguments:

- block - the block of plaintext to be encrypted (the block size is 8 bytes).

Return:

- The block of ciphertext (as a byte object).

---

decrypt(block)

Decrypting a block of ciphertext.

```
import gostcrypto

key = bytearray([
    0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff
])

cipher_block = bytearray([
    0x4e, 0xe9, 0x01, 0xe5, 0xc2, 0xd8, 0xca, 0x3d,
])

cipher_obj = gostcrypto.gostcipher.GOST34122015Magma(key)
plain_block = cipher_obj.encrypt(cipher_block)
```

Arguments:

- block - the block of ciphertext to be decrypted (the block size is 8 bytes).

Return:

- The block of plaintext (as a byte object).

clear()

Clearing the values of iterative encryption keys.

```
import gostcrypto

key = bytearray([
    0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff
```

(continues on next page)

(continued from previous page)

```
])

cipher_block = bytearray([
    0x4e, 0xe9, 0x01, 0xe5, 0xc2, 0xd8, 0xca, 0x3d,
])

cipher_obj = gostcrypto.gostcipher.GOST34122015Magma(key)
plain_block = cipher_obj.encrypt(cipher_block)
cipher_obj.clear()
```

Attributes:

block_size

An integer value the internal block size of the cipher algorithm in bytes. For the 'Magma' algorithm this value is 8.

```
import gostcrypto

key = bytearray([
    0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff
])

cipher_obj = gostcrypto.gostcipher.GOST34122015Magma(key)
block_size = cipher_obj.block_size
```

key_size

An integer value the cipher key size.

```
import gostcrypto

key = bytearray([
    0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff
])

cipher_obj = gostcrypto.gostcipher.GOST34122015Magma(key)
key_size = cipher_obj.key_size
```

GOST34132015

Base class of the cipher object. This class is a superclass for the GOST34132015Cipher and GOST34132015mac classes.

Methods:

clear()

Clearing the values of iterative encryption keys.

Attributes:

block_size

An integer value the internal block size of the cipher algorithm in bytes. For the 'Kuznechik' algorithm this value is 16 and the 'Magma' algorithm, this value is 8.

GOST34132015Cipher

Base class of the cipher object for implementing encryption modes. This class is the subclass of the GOST3413205 class and inherits the clear() method and the block_size attribute. Class GOST34132015Cipher is a superclass for the GOST34132015CipherPadding, GOST34132015CipherFeedBack and GOST34132015ctr classes.

Methods:

encrypt(data)

Abstract method. Implements input data validation.

This method must be redefined in subclasses of this class. For example:

```
# defining the 'encrypt' method in a subclass
def encrypt(self, data):
    data = super().encrypt(data)
    # ...further actions with data...
```

Arguments:

- data - plaintext data to be encrypted (as a byte object).

Return:

- If the data value is checked successfully returns this value unchanged.

Exceptions:

- GOSTCipherError('invalid plaintext data') - in case where the plaintext data is not byte object.

decrypt(data)

Abstract method. Implements input data validation.

This method must be redefined in subclasses of this class. For example:

```
# defining the 'decrypt' method in a subclass
def decrypt(self, data):
    data = super().decrypt(data)
    # ...further actions with data...
```

Arguments:

- data - ciphertext data to be decrypted (as a byte object).

Return:

- If the data value is checked successfully returns this value unchanged.

Exceptions:

- GOSTCipherError('invalid ciphertext data') - in case where the plaintext data is not byte object.

GOST34132015CipherPadding

Base class of the cipher object for implementing encryption modes with padding. This class is the subclass of the GOST3413205Cipher class and inherits the clear() method and the block_size attribute. The encrypt() and decrypt() methods are redefined. Class GOST34132015CipherPadding is a superclass for the GOST34132015ecb and GOST34132015cbc classes.

Methods:

encrypt(data)

Abstract method. Implementing input validation and the procedure of paddingю

This method must be redefined in subclasses of this class. For example:

```
# defining the 'encrypt' method in a subclass
def encrypt(self, data):
    data = super().encrypt(data)
    # ...further actions with data...
```

Arguments:

- data - plaintext data to be encrypted (as a byte object).

Return:

- If the data value is checked successfully, the padding procedure is performed and the resulting value is returned.

Exceptions:

- GOSTCipherError('invalid plaintext data') - in case where the plaintext data is not byte object.

decrypt(data)

Abstract method. Implements input data validation.

This method must be redefined in subclasses of this class. For example:

```
# defining the 'decrypt' method in a subclass
def decrypt(self, data):
    data = super().decrypt(data)
    # ...further actions with data...
```

Arguments:

- data - ciphertext data to be decrypted (as a byte object).

Return:

- If the data value is checked successfully returns this value unchanged.

Exceptions:

- GOSTCipherError('invalid ciphertext data') - in case where the plaintext data is not byte object.

GOST34132015CipherFeedBack

Base class of the cipher object for implementing encryption modes with feedback. This class is the subclass of the GOST3413205Cipher class and inherits the clear() method and the block_size attribute. The encrypt() and decrypt() methods are redefined. Class GOST34132015CipherFeedBack is a superclass for the GOST34132015cbc, GOST34132015cfb and GOST34132015ofb classes.

Methods:

encrypt(data)

Abstract method. Implements input data validation.

This method must be redefined in subclasses of this class. For example:

```
# defining the 'encrypt' method in a subclass
def encrypt(self, data):
    data = super().encrypt(data)
    # ...further actions with data...
```

Arguments:

- data - plaintext data to be encrypted (as a byte object).

Return:

- If the data value is checked successfully returns this value unchanged.

Exceptions:

- GOSTCipherError('invalid plaintext data') - in case where the plaintext data is not byte object.

decrypt(data)

Abstract method. Implements input data validation.

This method must be redefined in subclasses of this class. For example:

```
# defining the 'decrypt' method in a subclass
def decrypt(self, data):
    data = super().decrypt(data)
    # ...further actions with data...
```

Arguments:

- data - ciphertext data to be decrypted (as a byte object).

Return:

- If the data value is checked successfully returns this value unchanged.

Exceptions:

- GOSTCipherError('invalid ciphertext data') - in case where the plaintext data is not byte object.

Attributes:

iv

The byte object value of the initializing vector.

GOST34132015ecb

Class that implements ECB block encryption mode in accordance with GOST 34.13-2015. This class is the subclass of the GOST3413205CipherPadding class and inherits the clear() method and the block_size attribute. The encrypt() and decrypt() methods are redefined.

Methods:

encrypt(data)

Encrypting a plaintext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

plain_text = = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_ECB,
                                       pad_mode=gostcrypto.gostcipher.PAD_MODE_2)
cipher_text = cipher_obj.encrypt(plain_text)
```

Arguments:

- data - plaintext data to be encrypted (as a byte object).

Return:

- Ciphertext data (as a byte object).

Exceptions:

- GOSTCipherError('invalid plaintext data') - in case where the plaintext data is not byte object.

decrypt(data)

Decrypting a ciphertext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

cipher_text = = bytearray([
    0x7f, 0x67, 0x9d, 0x90, 0xbe, 0xbc, 0x24, 0x30, 0x5a, 0x46, 0x8d, 0x42, 0xb9, 0xd4, 0xed, 0xcd,
    0xb4, 0x29, 0x91, 0x2c, 0x6e, 0x00, 0x32, 0xf9, 0x28, 0x54, 0x52, 0xd7, 0x67, 0x18, 0xd0, 0x8b,
    0xf0, 0xca, 0x33, 0x54, 0x9d, 0x24, 0x7c, 0xee, 0xf3, 0xf5, 0xa5, 0x31, 0x3b, 0xd4, 0xb1, 0x57,
    0xd0, 0xb0, 0x9c, 0xcd, 0xe8, 0x30, 0xb9, 0xeb, 0x3a, 0x02, 0xc4, 0xc5, 0xaa, 0x8a, 0xda, 0x98,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_ECB,
                                       pad_mode=gostcrypto.gostcipher.PAD_MODE_2)
plain_text = cipher_obj.decrypt(cipher_text)
```

Arguments:

- data - ciphertext data to be decrypted (as a byte object).

Return:

- Plaintext data (as a byte object).

Exceptions:

- GOSTCipherError('invalid ciphertext data') - in case where the ciphertext data is not byte object.

GOST34132015cbc

Class that implements CBC block encryption mode in accordance with GOST 34.13-2015. This class is the subclass of the GOST3413205CipherPadding and GOST34132015CipherFeedBack classes and inherits the clear() method and the block_size and iv attributes. The encrypt() and decrypt() methods are redefined.

Methods:

encrypt(data)

Encrypting a plaintext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

init_vect = bytearray([
    0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce, 0xf0, 0xa1, 0xb2, 0xc3, 0xd4, 0xe5, 0xf0, 0x01, 0x12,
    0x23, 0x34, 0x45, 0x56, 0x67, 0x78, 0x89, 0x90, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19,
])

plain_text = = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_CBC,
                                       init_vect=init_vect,
                                       pad_mode=gostcrypto.gostcipher.PAD_MODE_2)
cipher_text = cipher_obj.encrypt(plain_text)
```

Arguments:

- data - plaintext data to be encrypted (as a byte object).

Return:

- Ciphertext data (as a byte object).

Exceptions:

- GOSTCipherError('invalid plaintext data') - in case where the plaintext data is not byte object.

decrypt(data)

Decrypting a ciphertext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

init_vect = bytearray([
    0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce, 0xf0, 0xa1, 0xb2, 0xc3, 0xd4, 0xe5, 0xf0, 0x01, 0x12,
    0x23, 0x34, 0x45, 0x56, 0x67, 0x78, 0x89, 0x90, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19,
```

(continues on next page)

(continued from previous page)

```
])

cipher_text = = bytearray([
    0x68, 0x99, 0x72, 0xd4, 0xa0, 0x85, 0xfa, 0x4d, 0x90, 0xe5, 0x2e, 0x3d, 0x6d, 0x7d, 0xcc, 0x27,
    0x28, 0x26, 0xe6, 0x61, 0xb4, 0x78, 0xec, 0xa6, 0xaf, 0x1e, 0x8e, 0x44, 0x8d, 0x5e, 0xa5, 0xac,
    0xfe, 0x7b, 0xab, 0xf1, 0xe9, 0x19, 0x99, 0xe8, 0x56, 0x40, 0xe8, 0xb0, 0xf4, 0x9d, 0x90, 0xd0,
    0x16, 0x76, 0x88, 0x06, 0x5a, 0x89, 0x5c, 0x63, 0x1a, 0x2d, 0x9a, 0x15, 0x60, 0xb6, 0x39, 0x70,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_CBC,
                                       init_vect=init_vect,
                                       pad_mode=gostcrypto.gostcipher.PAD_MODE_2)
plain_text = cipher_obj.decrypt(cipher_text)
```

Arguments:

- data - ciphertext data to be decrypted (as a byte object).

Return:

- Plaintext data (as a byte object).

Exceptions:

- GOSTCipherError('invalid ciphertext data') - in case where the ciphertext data is not byte object.

GOST34132015cfb

Class that implements CFB block encryption mode in accordance with GOST 34.13-2015. This class is the subclass of the GOST34132015CipherFeedBack class and inherits the clear() method and the block_size and iv attributes. The encrypt() and decrypt() methods are redefined.

Methods:

encrypt(data)

Encrypting a plaintext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])
```

(continues on next page)

(continued from previous page)

```
init_vect = bytearray([
    0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce, 0xf0, 0xa1, 0xb2, 0xc3, 0xd4, 0xe5, 0xf0, 0x01, 0x12,
    0x23, 0x34, 0x45, 0x56, 0x67, 0x78, 0x89, 0x90, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19,
])

plain_text = = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_CFB,
                                       init_vect=init_vect)
cipher_text = cipher_obj.encrypt(plain_text)
```

Arguments:

- data - plaintext data to be encrypted (as a byte object).

Return:

- Ciphertext data (as a byte object).

Exceptions:

- GOSTCipherError('invalid plaintext data') - in case where the plaintext data is not byte object.

decrypt(data)

Decrypting a ciphertext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

init_vect = bytearray([
    0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce, 0xf0, 0xa1, 0xb2, 0xc3, 0xd4, 0xe5, 0xf0, 0x01, 0x12,
    0x23, 0x34, 0x45, 0x56, 0x67, 0x78, 0x89, 0x90, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19,
])

cipher_text = = bytearray([
    0x81, 0x80, 0x0a, 0x59, 0xb1, 0x84, 0x2b, 0x24, 0xff, 0x1f, 0x79, 0x5e, 0x89, 0x7a, 0xbd, 0x95,
    0xed, 0x5b, 0x47, 0xa7, 0x04, 0x8c, 0xfa, 0xb4, 0x8f, 0xb5, 0x21, 0x36, 0x9d, 0x93, 0x26, 0xbf,
    0x79, 0xf2, 0xa8, 0xeb, 0x5c, 0xc6, 0x8d, 0x38, 0x84, 0x2d, 0x26, 0x4e, 0x97, 0xa2, 0x38, 0xb5,
    0x4f, 0xfe, 0xbe, 0xcd, 0x4e, 0x92, 0x2d, 0xe6, 0xc7, 0x5b, 0xd9, 0xdd, 0x44, 0xfb, 0xf4, 0xd1,
```

(continues on next page)

(continued from previous page)

```
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_CFB,
                                       init_vect=init_vect)
plain_text = cipher_obj.decrypt(cipher_text)
```

Arguments:

- data - ciphertext data to be decrypted (as a byte object).

Return:

- Plaintext data (as a byte object).

Exceptions:

- GOSTCipherError('invalid ciphertext data') - in case where the ciphertext data is not byte object.

GOST34132015ofb

Class that implements OFB block encryption mode in accordance with GOST 34.13-2015. This class is the subclass of the GOST34132015CipherFeedBack class and inherits the clear() method and the block_size and iv attributes. The encrypt() and decrypt() methods are redefined.

Methods:

encrypt(data)

Encrypting a plaintext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

init_vect = bytearray([
    0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce, 0xf0, 0xa1, 0xb2, 0xc3, 0xd4, 0xe5, 0xf0, 0x01, 0x12,
    0x23, 0x34, 0x45, 0x56, 0x67, 0x78, 0x89, 0x90, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19,
])

plain_text = = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
```

(continues on next page)

(continued from previous page)

```
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_OFB,
                                       init_vect=init_vect)
cipher_text = cipher_obj.encrypt(plain_text)
```

Arguments:

- data - plaintext data to be encrypted (as a byte object).

Return:

- Ciphertext data (as a byte object).

Exceptions:

- GOSTCipherError('invalid plaintext data') - in case where the plaintext data is not byte object.

decrypt(data)

Decrypting a ciphertext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

init_vect = bytearray([
    0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce, 0xf0, 0xa1, 0xb2, 0xc3, 0xd4, 0xe5, 0xf0, 0x01, 0x12,
    0x23, 0x34, 0x45, 0x56, 0x67, 0x78, 0x89, 0x90, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19,
])

cipher_text = = bytearray([
    0x81, 0x80, 0x0a, 0x59, 0xb1, 0x84, 0x2b, 0x24, 0xff, 0x1f, 0x79, 0x5e, 0x89, 0x7a, 0xbd, 0x95,
    0xed, 0x5b, 0x47, 0xa7, 0x04, 0x8c, 0xfa, 0xb4, 0x8f, 0xb5, 0x21, 0x36, 0x9d, 0x93, 0x26, 0xbf,
    0x66, 0xa2, 0x57, 0xac, 0x3c, 0xa0, 0xb8, 0xb1, 0xc8, 0x0f, 0xe7, 0xfc, 0x10, 0x28, 0x8a, 0x13,
    0x20, 0x3e, 0xbb, 0xc0, 0x66, 0x13, 0x86, 0x60, 0xa0, 0x29, 0x22, 0x43, 0xf6, 0x90, 0x31, 0x50,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_OFB,
                                       init_vect=init_vect)
plain_text = cipher_obj.decrypt(cipher_text)
```

Arguments:

- data - ciphertext data to be decrypted (as a byte object).

Return:

- Plaintext data (as a byte object).

Exceptions:

- GOSTCipherError('invalid ciphertext data') - in case where the ciphertext data is not byte object.

GOST34132015ctr

Class that implements CTR block encryption mode in accordance with GOST 34.13-2015. This class is the subclass of the GOST3413205Cipher class and inherits the clear() method and the block_size attribute. The encrypt() and decrypt() methods are redefined.

Methods:

encrypt(data)

Encrypting a plaintext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

init_vect = bytearray([
    0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce, 0xf0,
])

plain_text = = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_CTR,
                                       init_vect=init_vect)
cipher_text = cipher_obj.encrypt(plain_text)
```

Arguments:

- data - plaintext data to be encrypted (as a byte object).

Return:

- Ciphertext data (as a byte object).

Exceptions:

- GOSTCipherError('invalid plaintext data') - in case where the plaintext data is not byte object.

decrypt(data)

Decrypting a ciphertext.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

init_vect = bytearray([
    0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce, 0xf0,
])

cipher_text = = bytearray([
    0xf1, 0x95, 0xd8, 0xbe, 0xc1, 0x0e, 0xd1, 0xdb, 0xd5, 0x7b, 0x5f, 0xa2, 0x40, 0xbd, 0xa1, 0xb8,
    0x85, 0xee, 0xe7, 0x33, 0xf6, 0xa1, 0x3e, 0x5d, 0xf3, 0x3c, 0xe4, 0xb3, 0x3c, 0x45, 0xde, 0xe4,
    0xa5, 0xea, 0xe8, 0x8b, 0xe6, 0x35, 0x6e, 0xd3, 0xd5, 0xe8, 0x77, 0xf1, 0x35, 0x64, 0xa3, 0xa5,
    0xcb, 0x91, 0xfa, 0xb1, 0xf2, 0x0c, 0xba, 0xb6, 0xd1, 0xc6, 0xd1, 0x58, 0x20, 0xbd, 0xba, 0x73,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_CTR,
                                       init_vect=init_vect)
plain_text = cipher_obj.decrypt(cipher_text)
```

Arguments:

- data - ciphertext data to be decrypted (as a byte object).

Return:

- Plaintext data (as a byte object).

Exceptions:

- GOSTCipherError('invalid ciphertext data') - in case where the ciphertext data is not byte object.

Attributes:

counter

The byte object value of the counter block.

GOST34132015mac

Class that implements MAC mode in accordance with GOST 34.13-2015. This class is the subclass of the GOST3413205 class and inherits the clear() method and the block_size attribute.

Methods:

update(data)

Update the MAC object with the bytes-like object.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

mac_text = = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_MAC)
cipher_obj.update(mac_text)
```

Arguments:

- data - the string from which to get the MAC. Repeated calls are equivalent to a single call with the concatenation of all the arguments: m.update(a); m.update(b) is equivalent to m.update(a+b).

Exceptions:

- GOSTCipherError('invalid text data') - in case where the text data is not byte object.

digest(mac_size)

Calculating the data message authentication code (MAC) after applying the update(data) method.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

plain_text = = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_MAC)
cipher_obj.update(plain_text)
mac_result = cipher_obj.digest(8)
```

Arguments:

- mac_size - message authentication code size (in bytes).

Return:

- Message authentication code value (as a byte object).

Exceptions:

- GOSTCipherError('invalid message authentication code size') - in case of the invalid message authentication code size.

hexdigest(mac_size)

Calculating the data message authentication code (MAC) after applying the update(data) method.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

plain_text = = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
```

(continues on next page)

(continued from previous page)

```
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_MAC)
cipher_obj.update(plain_text)
mac_result = cipher_obj.hexdigest(8)
```

Arguments:

- mac_size - message authentication code size (in bytes).

Return:

- Message authentication code value (as a hexadecimal string).

Exceptions:

- GOSTCipherError('invalid message authentication code size') - in case of the invalid message authentication code size.

GOSTCipherError

The class that implements exceptions.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

plain_text = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

try:
    cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                           key,
                                           gostcrypto.gostcipher.MODE_ECB,
                                           pad_mode=gostcrypto.gostcipher.PAD_MODE_2)
    cipher_text = cipher_obj.encrypt(plain_text)
except gostcrypto.gostcipher.GOSTCipherError as err:
    print(err)
else:
    print(cipher_text.hex())
```

Exception types:

- unsupported cipher mode - in case of unsupported cipher mode (is not MODE_ECB, MODE_CBC, MODE_CFB, MODE_OFB, MODE_CTR or MODE_MAC).
- unsupported cipher algorithm - in case of invalid value algorithm.
- invalid key value - in case of invalid key value (the key value is not a byte object ('bytearray' or 'bytes') or its length is not 256 bits).
- invalid padding mode - in case padding mode is incorrect (for MODE_ECB and MODE_CBC modes).
- invalid initialization vector value - in case initialization vector value is incorrect (for all modes except MODE_ECB mode).
- invalid text data - in case where the text data is not byte object (for MODE_MAC mode).
- invalid plaintext data - in case where the plaintext data is not byte object.
- invalid ciphertext data - in case where the ciphertext data is not byte object.
- invalid message authentication code size - in case of the invalid message authentication code size.

2.2.6 Example of use

String encryption in ECB mode

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

plain_text = bytearray([
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00, 0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00,
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xee, 0xff, 0x0a, 0x00, 0x11,
])

cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_ECB,
                                       pad_mode=gostcrypto.gostcipher.PAD_MODE_1)

cipher_text = cipher_obj.encrypt(plain_text)
```

File encryption in CTR mode

Note: In this case the 'buffer_size' value must be a multiple of the 'block_size' value.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

init_vect = bytearray([
    0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce, 0xf0,
])

plain_file_path = 'plain_file.txt'
cipher_file_path = 'cipher_file.txt'
cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_CTR,
                                       init_vect=init_vect)

buffer_size = 128

plain_file = open(plain_file_path, 'rb')
cipher_file = open(cipher_file_path, 'wb')
buffer = plain_file.read(buffer_size)
while len(buffer) > 0:
    cipher_data = cipher_obj.encrypt(buffer)
    cipher_file.write(cipher_data)
    buffer = plain_file.read(buffer_size))
```

Calculating MAC of the file

Note: In this case the 'buffer_size' value must be a multiple of the 'block_size' value.

```
import gostcrypto

key = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

plain_file_path = 'plain_file.txt'
cipher_obj = gostcrypto.gostcipher.new('kuznechik',
                                       key,
                                       gostcrypto.gostcipher.MODE_MAC)

buffer_size = 128

plain_file = open(plain_file_path, 'rb')
buffer = plain_file.read(buffer_size)
while len(buffer) > 0:
    cipher_obj.update(buffer)
    buffer = plain_file.read(buffer_size)
mac_result = cipher_obj.digest(8)
```

2.3 API of the 'gostcrypto.gostsignature' module

2.3.1 Introduction

The module implements the functions of forming and verifying an electronic digital signature in accordance with GOST R 34.10-2012. The module includes the GOST34102012 and GOSTSignatureError classes, the new function and constants.

2.3.2 Constants

- MODE_256 - 256-bit key signing mode.
- MODE_512 - 512-bit key signing mode.
- CURVES_R_1323565_1_024_2019 - parameters of elliptic curves defined in accordance with recommendations R 1323565.1.024-2019. It is a dictionary with the following elements:
  - 'id-tc26-gost-3410-2012-256-paramSetB' - parameters of the elliptic curve (set "B") for the mode with the 256-bit signature key in the canonical representation form (in the form of a dictionary with elements: p-module of the elliptic curve; a, b - coefficients of the elliptic curve equation; m - order of the elliptic curve point group; q - order of the cyclic subgroup of the elliptic curve point group; x, y-coordinates of the point on the elliptic curve).
  - 'id-tc26-gost-3410-2012-256-paramSetC' - parameters of the elliptic curve (set "C") for the mode with the 256-bit signature key in the canonical representation form (in the form of a dictionary with elements: p-module of the elliptic curve; a, b - coefficients of the elliptic curve equation; m - order of the elliptic curve point group; q - order of the cyclic subgroup of the elliptic curve point group; x, y-coordinates of the point on the elliptic curve).
  - 'id-tc26-gost-3410-2012-256-paramSetD' - parameters of the elliptic curve (set "D") for the mode with the 256-bit signature key in the canonical representation form (in the form of a dictionary with elements: p-module of the elliptic curve; a, b - coefficients of the elliptic curve equation; m - order of the elliptic curve point group; q - order of the cyclic subgroup of the elliptic curve point group; x, y-coordinates of the point on the elliptic curve)
  - 'id-tc26-gost-3410-12-512-paramSetA' - parameters of the elliptic curve (set "A") for the mode with the 512-bit signature key in the canonical representation form (in the form of a dictionary with elements: p-module of the elliptic curve; a, b - coefficients of the elliptic curve equation; m - order of the elliptic curve point group; q - order of the cyclic subgroup of the elliptic curve point group; x, y-coordinates of the point on the elliptic curve)
  - 'id-tc26-gost-3410-12-512-paramSetB' - parameters of the elliptic curve (set "B") for the mode with the 512-bit signature key in the canonical representation form (in the form of a dictionary with elements: p-module of the elliptic curve; a, b - coefficients of the elliptic curve equation; m - order of the elliptic curve point group; q - order of the cyclic subgroup of the elliptic curve point group; x, y-coordinates of the point on the elliptic curve)
  - 'id-tc26-gost-3410-2012-256-paramSetA' - parameters of the elliptic curve (set "A") for a mode with a key signature of 256 bits in the canonical form representation in the form of twisted Edwards curves (in the form of a dictionary with elements: p - module of an elliptic curve; a, b - coefficients of the equation of an elliptic curve to a canonical form; e, d - coefficients of the equation of an elliptic curve in twisted Edwards curves; m is the order of the group of points of an elliptic curve; q - order of cyclic subgroup of elliptic curve points; x, y - coordinates of a point on an elliptic curve to a canonical form; u, v - coordinates of a point on an elliptic curve in the form of twisted Edwards curves).

– 'id-tc26-gost-3410-2012-512-paramSetC' - parameters of the elliptic curve (set "C") for a mode with a key signature of 512 bits in the canonical form representation in the form of twisted Edwards curves (in the form of a dictionary with elements: p - module of an elliptic curve; a, b - coefficients of the equation of an elliptic curve to a canonical form; e, d - coefficients of the equation of an elliptic curve in twisted Edwards curves; m is the order of the group of points of an elliptic curve; q - order of cyclic subgroup of elliptic curve points; x, y - coordinates of a point on an elliptic curve to a canonical form; u, v - coordinates of a point on an elliptic curve in the form of twisted Edwards curves).

Example of setting an elliptic curve in canonical form

All parameters of the elliptic curve must be set as integers. In this case the bytearray_to_int function converts a byte array to a long integer. This function is defined in the utils module of the gostcrypto package.

```
'id-tc26-gost-3410-2012-256-paramSetB': dict(
    p=bytearray_to_int(bytearray([
        0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfd,
        0x97
    ])),
    a=bytearray_to_int(bytearray([
        0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfd,
        0x94
    ])),
    b=0xa6,
    m=bytearray_to_int(bytearray([
        0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0x6c, 0x61, 0x10, 0x70, 0x99, 0x5a, 0xd1,
        0x00, 0x45, 0x84, 0x1b, 0x09, 0xb7, 0x61, 0xb8,
        0x93
    ])),
    q=bytearray_to_int(bytearray([
        0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0x6c, 0x61, 0x10, 0x70, 0x99, 0x5a, 0xd1,
        0x00, 0x45, 0x84, 0x1b, 0x09, 0xb7, 0x61, 0xb8,
        0x93
    ])),
    x=0x01,
    y=bytearray_to_int(bytearray([
        0x00, 0x8d, 0x91, 0xe4, 0x71, 0xe0, 0x98, 0x9c,
        0xda, 0x27, 0xdf, 0x50, 0x5a, 0x45, 0x3f, 0x2b,
        0x76, 0x35, 0x29, 0x4f, 0x2d, 0xdf, 0x23, 0xe3,
        0xb1, 0x22, 0xac, 0xc9, 0x9c, 0x9e, 0x9f, 0x1e,
        0x14
    ]))
)
```

Example of simultaneously setting an elliptic curve in canonical form and as twisted Edwards curves

```
'id-tc26-gost-3410-2012-256-paramSetA': dict(
    p=bytearray_to_int(bytearray([
        0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfd,
        0x97
    ])),
    a=bytearray_to_int(bytearray([
        0x00, 0xc2, 0x17, 0x3f, 0x15, 0x13, 0x98, 0x16,
        0x73, 0xaf, 0x48, 0x92, 0xc2, 0x30, 0x35, 0xa2,
        0x7c, 0xe2, 0x5e, 0x20, 0x13, 0xbf, 0x95, 0xaa,
        0x33, 0xb2, 0x2c, 0x65, 0x6f, 0x27, 0x7e, 0x73,
        0x35
    ])),
    b=bytearray_to_int(bytearray([
        0x29, 0x5f, 0x9b, 0xae, 0x74, 0x28, 0xed, 0x9c,
        0xcc, 0x20, 0xe7, 0xc3, 0x59, 0xa9, 0xd4, 0x1a,
        0x22, 0xfc, 0xcd, 0x91, 0x08, 0xe1, 0x7b, 0xf7,
        0xba, 0x93, 0x37, 0xa6, 0xf8, 0xae, 0x95, 0x13
    ])),
    e=0x01,
    d=bytearray_to_int(bytearray([
        0x06, 0x05, 0xf6, 0xb7, 0xc1, 0x83, 0xfa, 0x81,
        0x57, 0x8b, 0xc3, 0x9c, 0xfa, 0xd5, 0x18, 0x13,
        0x2b, 0x9d, 0xf6, 0x28, 0x97, 0x00, 0x9a, 0xf7,
        0xe5, 0x22, 0xc3, 0x2d, 0x6d, 0xc7, 0xbf, 0xfb
    ])),
    m=bytearray_to_int(bytearray([
        0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x3f, 0x63, 0x37, 0x7f, 0x21, 0xed, 0x98,
        0xd7, 0x04, 0x56, 0xbd, 0x55, 0xb0, 0xd8, 0x31,
        0x9c
    ])),
    q=bytearray_to_int(bytearray([
        0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x0f, 0xd8, 0xcd, 0xdf, 0xc8, 0x7b, 0x66, 0x35,
        0xc1, 0x15, 0xaf, 0x55, 0x6c, 0x36, 0x0c, 0x67
    ])),
    x=bytearray_to_int(bytearray([
        0x00, 0x91, 0xe3, 0x84, 0x43, 0xa5, 0xe8, 0x2c,
        0x0d, 0x88, 0x09, 0x23, 0x42, 0x57, 0x12, 0xb2,
        0xbb, 0x65, 0x8b, 0x91, 0x96, 0x93, 0x2e, 0x02,
        0xc7, 0x8b, 0x25, 0x82, 0xfe, 0x74, 0x2d, 0xaa,
        0x28
    ])),
    y=bytearray_to_int(bytearray([
        0x32, 0x87, 0x94, 0x23, 0xab, 0x1a, 0x03, 0x75,
        0x89, 0x57, 0x86, 0xc4, 0xbb, 0x46, 0xe9, 0x56,
        0x5f, 0xde, 0x0b, 0x53, 0x44, 0x76, 0x67, 0x40,
        0xaf, 0x26, 0x8a, 0xdb, 0x32, 0x32, 0x2e, 0x5c
    ])),
    u=0x0d,
```

(continues on next page)

(continued from previous page)

```
    v=bytearray_to_int(bytearray([
        0x60, 0xca, 0x1e, 0x32, 0xaa, 0x47, 0x5b, 0x34,
        0x84, 0x88, 0xc3, 0x8f, 0xab, 0x07, 0x64, 0x9c,
        0xe7, 0xef, 0x8d, 0xbe, 0x87, 0xf2, 0x2e, 0x81,
        0xf9, 0x2b, 0x25, 0x92, 0xdb, 0xa3, 0x00, 0xe7
    ])),
)
```

Example of setting an elliptic curve as a twisted Edwards curves

```
'id-gost-3410-2012-256-twisted-Edwards-param': dict(
    p=bytearray_to_int(bytearray([
        0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfd,
        0x97
    ])),
    e=0x01,
    d=bytearray_to_int(bytearray([
        0x06, 0x05, 0xf6, 0xb7, 0xc1, 0x83, 0xfa, 0x81,
        0x57, 0x8b, 0xc3, 0x9c, 0xfa, 0xd5, 0x18, 0x13,
        0x2b, 0x9d, 0xf6, 0x28, 0x97, 0x00, 0x9a, 0xf7,
        0xe5, 0x22, 0xc3, 0x2d, 0x6d, 0xc7, 0xbf, 0xfb
    ])),
    m=bytearray_to_int(bytearray([
        0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x3f, 0x63, 0x37, 0x7f, 0x21, 0xed, 0x98,
        0xd7, 0x04, 0x56, 0xbd, 0x55, 0xb0, 0xd8, 0x31,
        0x9c
    ])),
    q=bytearray_to_int(bytearray([
        0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x0f, 0xd8, 0xcd, 0xdf, 0xc8, 0x7b, 0x66, 0x35,
        0xc1, 0x15, 0xaf, 0x55, 0x6c, 0x36, 0x0c, 0x67
    ])),
    u=0x0d,
    v=bytearray_to_int(bytearray([
        0x60, 0xca, 0x1e, 0x32, 0xaa, 0x47, 0x5b, 0x34,
        0x84, 0x88, 0xc3, 0x8f, 0xab, 0x07, 0x64, 0x9c,
        0xe7, 0xef, 0x8d, 0xbe, 0x87, 0xf2, 0x2e, 0x81,
        0xf9, 0x2b, 0x25, 0x92, 0xdb, 0xa3, 0x00, 0xe7
    ])),
)
```

Note: It is possible to use other parameters of elliptic curves besides those defined in this module. Then these parameters must meet the requirements presented in paragraph 5.2 of GOST 34.10-2012.

2.3.3 Functions

new(mode, curve)

Creates a new signature object and returns it .

```
import gostcrypto

sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
    gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])
```

Arguments:

- mode - signature generation or verification mode (acceptable values are MODE_256 or MODE_512).
- curve - parameters of the elliptic curve.

Return:

- New signature object (as an instance of the GOST34102012 class).

Exceptions:

- GOSTSignatureError('unsupported signature mode') - in case of unsupported signature mode.
- GOSTSignatureError('invalid parameters of the elliptic curve') - if the elliptic curve parameters are incorrect.

2.3.4 Classes

GOST34102012

Class that implements processes for creating and verifying an electronic digital signature with GOST 34.10-2012.

Methods:

sign(private_key, digest, rand_k)

Creating a signature.

```
sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
    gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])

private_key = bytearray([
    0x7a, 0x92, 0x9a, 0xde, 0x78, 0x9b, 0xb9, 0xbe, 0x10, 0xed, 0x35, 0x9d, 0xd3, 0x9a, 0x72, 0xc1,
    0x1b, 0x60, 0x96, 0x1f, 0x49, 0x39, 0x7e, 0xee, 0x1d, 0x19, 0xce, 0x98, 0x91, 0xec, 0x3b, 0x28,
])
```

(continues on next page)

(continued from previous page)

```
digest = bytearray([
    0x2d, 0xfb, 0xc1, 0xb3, 0x72, 0xd8, 0x9a, 0x11, 0x88, 0xc0, 0x9c, 0x52, 0xe0, 0xee, 0xc6, 0x1f,
    0xce, 0x52, 0x03, 0x2a, 0xb1, 0x02, 0x2e, 0x8e, 0x67, 0xec, 0xe6, 0x67, 0x2b, 0x04, 0x3e, 0xe5,
])

rand_k = bytearray([
    0x77, 0x10, 0x5c, 0x9b, 0x20, 0xbc, 0xd3, 0x12, 0x28, 0x23, 0xc8, 0xcf, 0x6f, 0xcc, 0x7b, 0x95,
    0x6d, 0xe3, 0x38, 0x14, 0xe9, 0x5b, 0x7f, 0xe6, 0x4f, 0xed, 0x92, 0x45, 0x94, 0xdc, 0xea, 0xb3,
)]

signature = sign_obj.sign(private_key, digest, rand_k)
```

Arguments:

- private_key - private signature key (as a 32-byte object for MODE_256 or 64-byte object for MODE_512).
- digest - digest for which the signature is calculated (the digest should be calculated using the "streebog" algorithm for GOST 34.11-2012).
- rand_k - random (pseudo-random) number (as a byte object). If this argument is not passed to the function, the random_k value is generated by the function itself using os.urandom.

Return:

- Signature for provided digest (as a byte object).

Exception:

- GOSTSignatureError('invalid private key value') - if the private key value is incorrect.
- GOSTSignatureError('invalid digest value') - if the digest value is incorrect.
- GOSTSignatureError('invalid random value') - if the random value is incorrect.

---

verify(public_key, digest, signature)

Verify a signature.

```
sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
    gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])

private_key = bytearray([
    0x7a, 0x92, 0x9a, 0xde, 0x78, 0x9b, 0xb9, 0xbe, 0x10, 0xed, 0x35, 0x9d, 0xd3, 0x9a, 0x72, 0xc1,
    0x1b, 0x60, 0x96, 0x1f, 0x49, 0x39, 0x7e, 0xee, 0x1d, 0x19, 0xce, 0x98, 0x91, 0xec, 0x3b, 0x28,
])

digest = bytearray([
    0x2d, 0xfb, 0xc1, 0xb3, 0x72, 0xd8, 0x9a, 0x11, 0x88, 0xc0, 0x9c, 0x52, 0xe0, 0xee, 0xc6, 0x1f,
    0xce, 0x52, 0x03, 0x2a, 0xb1, 0x02, 0x2e, 0x8e, 0x67, 0xec, 0xe6, 0x67, 0x2b, 0x04, 0x3e, 0xe5,
```

(continues on next page)

(continued from previous page)

```
])

signature = bytearray([
    0x41, 0xaa, 0x28, 0xd2, 0xf1, 0xab, 0x14, 0x82, 0x80, 0xcd, 0x9e, 0xd5, 0x6f, 0xed, 0xa4, 0x19,
    0x74, 0x05, 0x35, 0x54, 0xa4, 0x27, 0x67, 0xb8, 0x3a, 0xd0, 0x43, 0xfd, 0x39, 0xdc, 0x04, 0x93,
    0x01, 0x45, 0x6c, 0x64, 0xba, 0x46, 0x42, 0xa1, 0x65, 0x3c, 0x23, 0x5a, 0x98, 0xa6, 0x02, 0x49,
    0xbc, 0xd6, 0xd3, 0xf7, 0x46, 0xb6, 0x31, 0xdf, 0x92, 0x80, 0x14, 0xf6, 0xc5, 0xbf, 0x9c, 0x40,
])

if sign_obj.verify(public_key, digest, signature):
    print('Signature is correct')
else:
    print('Signature is not correct')
```

Arguments:

- public_key - public signature key (as a byte object).
- digest - digest for which to be checked signature (as a byte object).
- signature - signature of the digest being checked (as a byte object).

Return:

- The result of the signature verification (True or False).

Exception:

- GOSTSignatureError('invalid public key value') - if the public key value is incorrect.
- GOSTSignatureError('invalid digest value') - if the digest value is incorrect.
- GOSTSignatureError('invalid random value') - if the random value is incorrect.

---

public_key_generate(private_key)

```
sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
    gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])

private_key = bytearray([
    0x7a, 0x92, 0x9a, 0xde, 0x78, 0x9b, 0xb9, 0xbe, 0x10, 0xed, 0x35, 0x9d, 0xd3, 0x9a, 0x72, 0xc1,
    0x1b, 0x60, 0x96, 0x1f, 0x49, 0x39, 0x7e, 0xee, 0x1d, 0x19, 0xce, 0x98, 0x91, 0xec, 0x3b, 0x28,
])

public_key = sign_obj.public_key_generate(private_key)
```

Arguments:

- private_key - private signature key (as a 32-byte object for MODE_256 or 64-byte object for MODE_512).

Return:

- Public key (as a byte object).

Exception:

- GOSTSignatureError('invalid private key value') - if the private key value is incorrect.

GOSTSignatureError

The class that implements exceptions.

```
private_key = bytearray([
    0x7a, 0x92, 0x9a, 0xde, 0x78, 0x9b, 0xb9, 0xbe, 0x10, 0xed, 0x35, 0x9d, 0xd3, 0x9a, 0x72, 0xc1,
    0x1b, 0x60, 0x96, 0x1f, 0x49, 0x39, 0x7e, 0xee, 0x1d, 0x19, 0xce, 0x98, 0x91, 0xec, 0x3b, 0x28,
])

digest = bytearray([
    0x2d, 0xfb, 0xc1, 0xb3, 0x72, 0xd8, 0x9a, 0x11, 0x88, 0xc0, 0x9c, 0x52, 0xe0, 0xee, 0xc6, 0x1f,
    0xce, 0x52, 0x03, 0x2a, 0xb1, 0x02, 0x2e, 0x8e, 0x67, 0xec, 0xe6, 0x67, 0x2b, 0x04, 0x3e, 0xe5,
])

rand_k = bytearray([
    0x77, 0x10, 0x5c, 0x9b, 0x20, 0xbc, 0xd3, 0x12, 0x28, 0x23, 0xc8, 0xcf, 0x6f, 0xcc, 0x7b, 0x95,
    0x6d, 0xe3, 0x38, 0x14, 0xe9, 0x5b, 0x7f, 0xe6, 0x4f, 0xed, 0x92, 0x45, 0x94, 0xdc, 0xea, 0xb3,
)]

try:
    sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
        gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])
    signature = sign_obj.sign(private_key, digest, rand_k)
except gostcrypto.gostsignature.GOSTSignatureError as err:
    print(err)
else:
    print(signature)
```

Exception types:

- unsupported signature mode - in case of unsupported signature mode.
- invalid parameters of the elliptic curve - if the elliptic curve parameters are incorrect.
- invalid private key value - if the private key value is incorrect.
- invalid digest value - if the digest value is incorrect.
- invalid random value - if the random value is incorrect.
- invalid public key value - if the public key value is incorrect.
- invalid signature value - if the signature value is incorrect.

2.3.5 Example of use

Signing

```
import gostcrypto

private_key = bytearray([
    0x7a, 0x92, 0x9a, 0xde, 0x78, 0x9b, 0xb9, 0xbe, 0x10, 0xed, 0x35, 0x9d, 0xd3, 0x9a, 0x72, 0xc1,
    0x1b, 0x60, 0x96, 0x1f, 0x49, 0x39, 0x7e, 0xee, 0x1d, 0x19, 0xce, 0x98, 0x91, 0xec, 0x3b, 0x28,
])

digest = bytearray([
    0x2d, 0xfb, 0xc1, 0xb3, 0x72, 0xd8, 0x9a, 0x11, 0x88, 0xc0, 0x9c, 0x52, 0xe0, 0xee, 0xc6, 0x1f,
    0xce, 0x52, 0x03, 0x2a, 0xb1, 0x02, 0x2e, 0x8e, 0x67, 0xec, 0xe6, 0x67, 0x2b, 0x04, 0x3e, 0xe5,
])

sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
    gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])

signature = sign_obj.sign(private_key, digest)
```

Verify

```
public_key = bytearray([
    0xfd, 0x21, 0xc2, 0x1a, 0xb0, 0xdc, 0x84, 0xc1, 0x54, 0xf3, 0xd2, 0x18, 0xe9, 0x04, 0x0b, 0xee,
    0x64, 0xff, 0xf4, 0x8b, 0xdf, 0xf8, 0x14, 0xb2, 0x32, 0x29, 0x5b, 0x09, 0xd0, 0xdf, 0x72, 0xe4,
    0x50, 0x26, 0xde, 0xc9, 0xac, 0x4f, 0x07, 0x06, 0x1a, 0x2a, 0x01, 0xd7, 0xa2, 0x30, 0x7e, 0x06,
    0x59, 0x23, 0x9a, 0x82, 0xa9, 0x58, 0x62, 0xdf, 0x86, 0x04, 0x1d, 0x14, 0x58, 0xe4, 0x50, 0x49,
])

digest = bytearray([
    0x2d, 0xfb, 0xc1, 0xb3, 0x72, 0xd8, 0x9a, 0x11, 0x88, 0xc0, 0x9c, 0x52, 0xe0, 0xee, 0xc6, 0x1f,
    0xce, 0x52, 0x03, 0x2a, 0xb1, 0x02, 0x2e, 0x8e, 0x67, 0xec, 0xe6, 0x67, 0x2b, 0x04, 0x3e, 0xe5,
])

signature = bytearray([
    0x4b, 0x6d, 0xd6, 0x4f, 0xa3, 0x38, 0x20, 0xe9, 0x0b, 0x14, 0xf8, 0xf4, 0xe4, 0x9e, 0xe9, 0x2e,
    0xb2, 0x66, 0x0f, 0x9e, 0xeb, 0x4e, 0x1b, 0x31, 0x35, 0x17, 0xb6, 0xba, 0x17, 0x39, 0x79, 0x65,
    0x6d, 0xf1, 0x3c, 0xd4, 0xbc, 0xea, 0xf6, 0x06, 0xed, 0x32, 0xd4, 0x10, 0xf4, 0x8f, 0x2a, 0x5c,
    0x25, 0x96, 0xc1, 0x46, 0xe8, 0xc2, 0xfa, 0x44, 0x55, 0xd0, 0x8c, 0xf6, 0x8f, 0xc2, 0xb2, 0xa7,
])

sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
    gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])

if sign_obj.verify(public_key, digest, signature):
    print('Signature is correct')
else:
    print('Signature is not correct')
```

Generating a public key

```
private_key = bytearray([
    0x7a, 0x92, 0x9a, 0xde, 0x78, 0x9b, 0xb9, 0xbe, 0x10, 0xed, 0x35, 0x9d, 0xd3, 0x9a, 0x72, 0xc1,
    0x1b, 0x60, 0x96, 0x1f, 0x49, 0x39, 0x7e, 0xee, 0x1d, 0x19, 0xce, 0x98, 0x91, 0xec, 0x3b, 0x28,
])

sign_obj = gostcrypto.gostsignature.new(gostcrypto.gostsignature.MODE_256,
    gostcrypto.gostsignature.CURVES_R_1323565_1_024_2019['id-tc26-gost-3410-2012-256-paramSetB'])

public_key = sign_obj.public_key_generate(private_key)
```

2.4 API of the 'gostcrypto.gostrandom' module

2.4.1 Introduction

The module that implements pseudo-random sequence generation in accordance with R 1323565.1.006-2017. The module includes the R132356510062017 and GOSTRandomError classes, the new function and constants.

2.4.2 Constants

- SIZE_S_384 - the size of the initial filling (seed) is 384 bits.
- SIZE_S_320 - the size of the initial filling (seed) is 320 bits.
- SIZE_S_256 - the size of the initial filling (seed) is 256 bits.

Note: The specified values for the initial fill size are recommended in R 1323565.1.006-2017. It is possible to use other values that meet the requirements presented out in R 1323565.1.006-2017.

2.4.3 Functions

new(rand_size, **kwargs)

Creates a new pseudo-random sequence generation object and returns it.

```
import gostcrypto

random_k = bytearray([
    0xa8, 0xe2, 0xf9, 0x00, 0xdd, 0x4d, 0x7e, 0x24,
    0x5f, 0x09, 0x75, 0x3d, 0x01, 0xe8, 0x75, 0xfc,
    0x38, 0xf1, 0x4f, 0xf5, 0x25, 0x4c, 0x94, 0xea,
    0xdb, 0x45, 0x1e, 0x4a, 0xb6, 0x03, 0xb1, 0x47,
])
random_obj = gostcrypto.gostrandom.new(64,
                                       random_k=random_k,
                                       size_s=gostcrypto.gostrandom.SIZE_S_256)
```

Arguments:

- rand_size - size of the generated random variable (in bytes).

Keyword arguments:

- rand_k - initial filling (seed). If this argument is not passed to the function, the os.urandom function is used to generate the initial filling.
- size_s - size of the initial filling (in bytes). The default value is SIZE_S_384.

Return:

- New pseudo-random sequence generation object (as an instance of the R132356510062017 class).

Exceptions:

- GOSTRandomError('invalid seed value size') - in case of invalid size of initial filling.

2.4.4 Classes

R132356510062017

Class that implements pseudo-random sequence generation in accordance with R 1323565.1.006-2017.

Methods:

random()

Generating the next value from a pseudo-random sequence.

```
import gostcrypto

random_obj = gostcrypto.gostrandom.new(32)
random_result = random_obj.random()
```

Return:

- New random value (as a byte object).

Exception:

- GOSTRandomError('exceeded the limit value of the counter') - when the counter limit is exceeded.
- GOSTRandomError('the seed value is zero') - if the seed value is zero.

---

reset(rand_k)

Resetting the counter and setting a new initial filling.

```
import gostcrypto

rand_k_1 = bytearray([
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
])

rand_k_2 = bytearray([
    0xa8, 0xe2, 0xf9, 0x00, 0xdd, 0x4d, 0x7e, 0x24,
    0x5f, 0x09, 0x75, 0x3d, 0x01, 0xe8, 0x75, 0xfc,
    0x38, 0xf1, 0x4f, 0xf5, 0x25, 0x4c, 0x94, 0xea,
    0xdb, 0x45, 0x1e, 0x4a, 0xb6, 0x03, 0xb1, 0x47,
])

random_obj = gostcrypto.gostrandom.new(32,
                                       rand_k=rand_k_1,
                                       size_s=gostcrypto.gostrandom.SIZE_S_256)
random_result_1 = random_obj.random()
random_obj.reset(rand_k_2)
random_result_2 = random_obj.random()
```

Arguments:

- rand_k - new initial filling (seed). If this argument is not passed to the function, the os.urandom function is used to generate the initial filling.

Exception:

- GOSTRandomError('invalid seed value size') - in case of invalid size of initial filling.

---

clear()

Clearing the counter value.

```
import gostcrypto

random_obj = gostcrypto.gostrandom.new(32)
random_obj.clear()
```

GOSTRandomError

The class that implements exceptions.

```
import gostcrypto

random_k = bytearray([
    0xa8, 0xe2, 0xf9, 0x00, 0xdd, 0x4d, 0x7e, 0x24,
    0x5f, 0x09, 0x75, 0x3d, 0x01, 0xe8, 0x75, 0xfc,
    0x38, 0xf1, 0x4f, 0xf5, 0x25, 0x4c, 0x94, 0xea,
    0xdb, 0x45, 0x1e, 0x4a, 0xb6, 0x03, 0xb1, 0x47,
])
try:
    random_obj = gostcrypto.gostrandom.new(64,
                                           random_k=random_k,
                                           size_s=gostcrypto.gostrandom.SIZE_S_256)
    random_result = random_obj.random()
except gostcrypto.gostrandom.GOSTRandomError as err:
    print(err)
else:
    print(random_result.hex())
```

Exception types:

- invalid seed value - in case of invalid value of initial filling.
- exceeded the limit value of the counter - when the counter limit is exceeded.
- the seed value is zero - if the seed value is zero.

2.4.5 Example of use

```
import gostcrypto

rand_k = bytearray([
    0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff,
    0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10,
    0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
])

random_obj = gostcrypto.gostrandom.new(32,
                                       rand_k=rand_k,
                                       size_s=gostcrypto.gostrandom.SIZE_S_256)
random_result = random_obj.random()
random_obj.clear()
```

2.5 API of the 'gostcrypto.gosthmac' module

2.5.1 Introductoon

The module implementing the calculating the hash-based message authentication code (HMAC) in accordance with R 50.1.113-2016. The module includes the R5011132016 and GOSTHMACError classes and the new function.

2.5.2 API principles

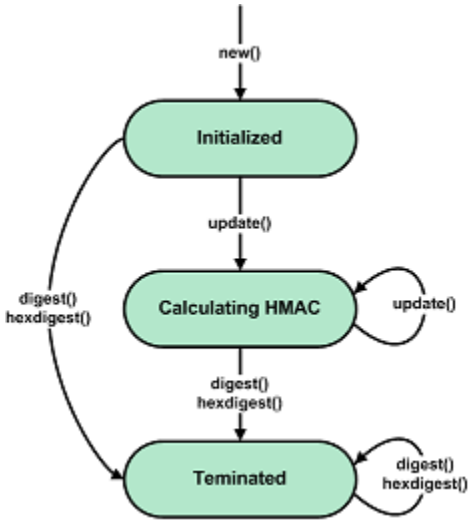

Fig. 5: Generic state diagram for a HMAC object

The first message fragment for a HMAC can be passed to the new() function with the data parameter after specifying the name of the HMAC algorithm (`'HMAC_GOSTR3411_2012_256'` or `'HMAC_GOSTR3411_2012_512'`) and after specifying key value:

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')
hmac_string = bytearray.fromhex('0126bdb87800af214341456563780100')

hmac_obj = gostrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key, data=hmac_string)
```

The data argument is optional and may be not passed to the new function. In this case, the data parameter must be passed in the update() method, which is called after new():

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')
hmac_string = bytearray.fromhex('0126bdb87800af214341456563780100')

hmac_obj = gostrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
hmac_obj.update(hmac_string)
```

After that, the update method can be called any number of times as needed, with other parts of the message

Passing the first part of the message to the new () function, and the subsequent parts to the update() method:

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key, data=b'first part message')
hmac_obj.update(b'second part message')
hmac_obj.update(b'third part message')
```

Passing the first part of the message and subsequent parts to the update() method:

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
hmac_obj.update(b'first part message')
hmac_obj.update(b'second part message')
hmac_obj.update(b'third part message')
```

HMAC calculation is completed using the digest() or hexdigest() method:

```
import gostcrypto

    key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
hmac_obj.update(b'first part message')
hmac_obj.update(b'second part message')
hmac_obj.update(b'third part message')
hmac_result = hmac_obj.digest()
```

2.5.3 Functions

new(name, key, **kwargs)

Creates a new authentication code calculation object and returns it.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')
hmac_string = bytearray.fromhex('0126bdb87800af214341456563780100')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key, data=hmac_string)
```

Arguments:

- name - name of the authentication code calculation mode ('HMAC_GOSTR3411_2012_256' or 'HMAC_GOSTR3411_2012_512').
- key - authentication key (as a byte object between 32 and 64 bytes in size).

Keyword arguments:

- data - the data from which to get the HMAC (as a byte object). If this argument is passed to a function, you can imm
  (or hexdigest()) method to calculate the HMAC value after calling new(). If the argument is not passed to the function, then you must use the"update()" method before the digest() (or hexdigest()) method.

Return:

- New authentication code calculation object object (as an instance of the R5011132016 class).

Exceptions:

- GOSTHMACError('unsupported mode') - in case of unsupported mode.
- GOSTHMACError('invalid key value') - in case of invalid key value.
- GOSTHMACError('invalid data value'): in case where the data is not byte object.

2.5.4 Classes

R5011132016

Methods:

update(data)

Update the HMAC object with the bytes-like object.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')
data = bytearray.fromhex('0126bdb87800af214341456563780100')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
hmac_obj.update(data)
```

Arguments:

- data - the message for which want to calculate the authentication code. Repeated calls are equivalent to a single call with the concatenation of all the arguments: m.update(a); m.update(b) is equivalent to m.update(a+b).

Exceptions:

- GOSTHMACError('invalid data value'): in case where the data is not byte object.

---

digest()

Returns the HMAC message authentication code.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')
data = bytearray.fromhex('0126bdb87800af214341456563780100')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
hmac_obj.update(data)
hmac_result = hmac_obj.digest()
```

Return:

- The HMAC message authentication code (as a byte object).

---

hexdigest()

Returns the HMAC message authentication code as a hexadecimal string.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')
data = bytearray.fromhex('0126bdb87800af214341456563780100')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
hmac_obj.update(data)
hmac_result = hmac_obj.hexdigest()
```

Return:

- The HMAC message authentication code (as a hexadecimal string).

---

copy()

Returns a copy ("clone") of the HMAC object. This can be used to efficiently compute the digests of data sharing a common initial substring.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')

hmac_obj_1 = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
hmac_obj_2 = hmac_obj.copy()
```

Return:

- The copy ("clone") of the HMAC object.

---

reset()

Resets the values of all class attributes.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')
data_1 = bytearray.fromhex('0126bdb87800af214341456563780100')
data_2 = bytearray.fromhex('434145656378010000126bdb87800af21')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
hmac_obj.update(data_1)
hmac_result_1 = hmac_obj.hexdigest()
hmac_obj.reset()
hmac_obj.update(data_2)
hmac_result_2 = hmac_obj.hexdigest()
```

---

clear()

Clears the key value.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
hmac_obj.clear()
```

Attributes:

digest_size

An integer value of the size of the resulting HMAC digest in bytes.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
digest_size = hmac_obj.digest_size
```

---

block_size

An integer value the internal block size of the hash algorithm in bytes.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
block_size = hmac_obj.block_size
```

---

name

A text string is the name of the authentication code calculation algorithm ('HMAC_GOSTR3411_2012_256' or 'HMAC_GOSTR3411_2012_512').

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
hmac_name = hmac_obj.name
```

GOSTHMACError

The class that implements exceptions.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')
data = bytearray.fromhex('0126bdb87800af214341456563780100')

try:
    hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key)
    hmac_obj.update(data)
except gostcrypto.gosthmac.GOSTHMACError as err:
    print(err)
else:
    hmac_result = hmac_obj.digest()
```

Exception types:

- unsupported mode - in case of unsupported mode.
- invalid key value - in case of invalid key value.
- invalid data value - in case where the data is not byte object.

2.5.5 Example of use

Getting a HMAC for a string

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131515161718191a1b1c1d1e1f')
data = bytearray.fromhex('0126bdb87800af214341456563780100')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key, data=data)
hmac_result = hmac_obj.digest()
```

Getting a HMAC for a file

In this case the buffer_size value must be a multiple of the block_size value.

```
import gostcrypto

key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131516171819 1a1b1c1d1e1f')
data = bytearray.fromhex('0126bdb87800af214341456563780100')

hmac_obj = gostcrypto.gosthmac.new('HMAC_GOSTR3411_2012_256', key, data=data)
hmac_result = hmac_obj.digest()
```

2.6 API of the 'gostcrypto.gostpbkdf' module

2.6.1 Introduction

The module implementing the password-based key derivation function in accordance with R 50.1.111-2016. The module includes the R5011112016 and GOSTPBKDFError classes and the new function.

2.6.2 Functions

new(password, **kwargs)

Creates a new object for the password-based key derivation function and returns it.

```
import gostcrypto

password = b'password'
salt = b'salt'

pbkdf_obj = gostcrypto.gostpbkdf.new(password, salt=salt, counter=2000)
```

Arguments:

- password - password that is a character string in Unicode UTF-8 encoding.

Keyword arguments:

- salt - random value. If this argument is not passed to the function, the os.urandom function is used to generate this value with the length of the generated value of 32 bytes.
- counter - number of iterations. The default value is 1000.

Return:

- New object for the password-based key derivation function (as an instance of the R5011112016 class).

Exception:

- GOSTPBKDFError(invalid password value') - if the password value is incorrect.
- GOSTPBKDFError('invalid salt value') - if the salt value is incorrect.

2.6.3 Classes

R5011112016

Class that implementing the calculating the password-based key derivation function in accordance with R 50.1.111-2016.

Methods:

derive(dk_len)

Returns a derived key (as a byte object).

```
import gostcrypto

password = b'password'
salt = b'salt'

pbkdf_obj = gostcrypto.gostpbkdf.new(password, salt=salt, counter=2000)
pbkdf_result = pbkdf_obj.derive(32)
```

Arguments:

- dk_len - Required length of the output sequence (in bytes).

Return:

- The derived key (as a byte object).

Exception:

- GOSTPBKDFError('invalid size of the derived key') - in case of invalid size of the derived key.

---

hexderive(dk_len)

Returns a derived key (as a hexadecimal string).

```
import gostcrypto

password = b'password'
salt = b'salt'

pbkdf_obj = gostcrypto.gostpbkdf.new(password, salt=salt, counter=2000)
pbkdf_result = pbkdf_obj.hexderive(32)
```

Arguments:

- dk_len - Required length of the output sequence (in bytes).

Return:

- The derived key (as a hexadecimal string).

Exception:

- GOSTPBKDFError('invalid size of the derived key') - in case of invalid size of the derived key.

---

clear()

Clears the password value.

```
import gostcrypto

password = b'password'
salt = b'salt'

pbkdf_obj = gostcrypto.gostpbkdf.new(password, salt=salt, counter=2000)
pbkdf_obj.clear()
```

Attributes:

salt

The byte object containing a random value (salt). Required when generating the salt value using os.urandom.

```
import gostcrypto

password = b'password'

pbkdf_obj = gostcrypto.gostpbkdf.new(password)
salt = pbkdf_obj.salt
```

GOSTPBKDFError

The class that implements exceptions.

```
import gostcrypto

password = b'password'
salt = b'salt'

try:
    pbkdf_obj = gostcrypto.gostpbkdf.new(password, salt=salt, counter=2000)
```

(continues on next page)

(continued from previous page)

```
    pbkdf_result = pbkdf_obj.hexderive(32)
except gostcrypto.gostpbkdf.GOSTPBKDFError as err:
    print(err)
else:
    print(pbkdf_result)
```

Exception types:

- invalid password value - if the password value is incorrect.
- invalid salt value - if the salt value is incorrect.
- invalid size of the derived key - if the size of the derived key is incorrect.

2.6.4 Example of use

```
import gostcrypto

password = b'password'
salt = b'salt'

pbkdf_obj = gostcrypto.gostpbkdf.new(password, salt=salt, counter=4096)
pbkdf_result = pbkdf_obj.derive(32)
```