

Package: smcryptoR (via r-universe)

October 14, 2024

Type Package

Title ShangMi(SM) Cryptographic Algorithms(SM2/SM3/SM4)

Version 0.1.2

Description Bindings to 'smcrypto'
<<https://github.com/zhuobie/smcrypto>>: a 'Rust' implementation of China's Standards of Encryption Algorithms, which is usually called 'ShangMi(SM)' algorithms. It contains 'SM3' message digest algorithm, 'SM2' asymmetric encryption algorithm and 'SM4' symmetric encryption algorithm. Users can do message hash, encrypt/decrypt, sign/verify, key exchange and more.

License MIT + file LICENSE

URL <https://github.com/zhuobie/smcryptoR>

BugReports <https://github.com/zhuobie/smcryptoR/issues>

Depends R (>= 4.2)

SystemRequirements Cargo (Rust's package manager), rustc

Biarch true

RoxygenNote 7.2.3

NeedsCompilation yes

Maintainer Meng Yu <610074@gmail.com>

Date/Publication 2024-02-14 00:00:00 UTC

Suggests knitr, rmarkdown

VignetteBuilder knitr

Repository <https://zhuobie.r-universe.dev>

RemoteUrl <https://github.com/zhuobie/smcryptor>

RemoteRef HEAD

RemoteSha df2d83fe640c791281aec200f969c8c59e224d20

Contents

sm2_encrypt	2
sm2_encrypt_asna1	3
sm2_encrypt_c1c2c3	4
sm2_encrypt_hex	5
sm2_encrypt_to_file	6
sm2_gen_keypair	7
sm2_keyexchange_1ab	8
sm2_sign	10
sm3_hash	11
sm4_encrypt_ecb	12
sm4_encrypt_ecb_base64	13
sm4_encrypt_ecb_to_file	15
Index	17

sm2_encrypt	<i>SM2 Encrypt/Decrypt</i>
-------------	----------------------------

Description

SM2 is an asymmetric encryption algorithm that can also be used to directly encrypt data. Typically, A encrypts a file or data using the public key, passes the ciphertext to B, and B decrypts it using the corresponding private key. SM2 encryption and decryption are suitable for shorter texts only. For larger files, the process can be very slow. According to the SM2 algorithm usage specifications, the encrypted ciphertext needs to be ASN.1 encoded. We provide the functions `sm2_encrypt_asna1` and `sm2_decrypt_asna1` for this purpose. Additionally, some scenarios use different arrangements of `c1`, `c2`, `c3`, so we also offer the functions `sm2_encrypt_c1c2c3` and `sm2_decrypt_c1c2c3`. To facilitate the transmission of binary data, we also provide functions to encrypt data into hexadecimal or base64 strings and decrypt from them.

Usage

```
sm2_encrypt(data, public_key)
```

```
sm2_decrypt(data, private_key)
```

Arguments

<code>data</code>	data to be encrypted or decrypted, must be a raw vector
<code>public_key</code>	a public key represented as a hexadecimal string
<code>private_key</code>	a private key represented as a hexadecimal string

Value

sm2_encrypt returns a raw vector of the cipher text

sm2_decrypt returns a raw vector of the plain text

Examples

```
## encrypt and decrypt - raw
keypair <- sm2_gen_keypair()
private_key <- keypair$private_key
public_key <- keypair$public_key
data <- 'abc' |> charToRaw()
enc <- sm2_encrypt(data, public_key)
enc
dec <- sm2_decrypt(enc, private_key)
dec
```

sm2_encrypt_asna1	<i>SM2 Encrypt/Decrypt - asn.1</i>
-------------------	------------------------------------

Description

According to the usage specifications of the SM2 algorithm, the encrypted data should be encoded using ASN.1, specifically including XCoordinate, YCoordinate, HASH, and CipherText. Among them, XCoordinate and YCoordinate each occupy 32 bytes, HASH occupies 32 bytes, and CipherText is the same length as the plaintext, plus a one-byte "04" identifier. After SM2 encryption and ASN.1 encoding, the ciphertext data will be 97 bytes longer than the original plaintext data.

Usage

```
sm2_encrypt_asna1(data, public_key)

sm2_decrypt_asna1(data, private_key)
```

Arguments

data	data to be encrypted or decrypted, must be a raw vector
public_key	a public key represented as a hexadecimal string
private_key	a private key represented as a hexadecimal string

Value

sm2_encrypt_asna1 returns a raw vector of the cipher text in the asn.1 encoding
sm2_decrypt_asna1 returns a raw vector of the plain text

Examples

```
## encrypt and decrypt as asn.1
keypair <- sm2_gen_keypair()
private_key <- keypair$private_key
public_key <- keypair$public_key
data <- 'abc' |> charToRaw()
enc <- sm2_encrypt_asna1(data, public_key)
```

```
enc
dec <- sm2_decrypt_asna1(enc, private_key)
dec
```

sm2_encrypt_c1c2c3 *SM2 Encrypt/Decrypt - c1c2c3*

Description

The result of SM2 asymmetric encryption consists of three parts: C1, C2, and C3. Among them, C1 is the elliptic curve point calculated based on a generated random number, C2 is the ciphertext data, and C3 is the digest value of SM3. Regarding the two modes of C1C2C3 and C1C3C2, the original Chinese national standard specified the order of C1C2C3, while the new standard follows the order of C1C3C2. These two different order modes are mainly designed to facilitate the parsing and processing of SM2 encryption results across different systems and environments.

Usage

```
sm2_encrypt_c1c2c3(data, public_key)

sm2_decrypt_c1c2c3(data, private_key)
```

Arguments

data	data to be encrypted or decrypted, must be a raw vector
public_key	a public key represented as a hexadecimal string
private_key	a private key represented as a hexadecimal string

Value

sm2_encrypt_c1c2c3 returns a raw vector of the cipher text in the order of c1c2c3

sm2_decrypt_c1c2c3 returns a raw vector of the plain text

Examples

```
## encrypt and decrypt as c1c2c3
keypair <- sm2_gen_keypair()
private_key <- keypair$private_key
public_key <- keypair$public_key
data <- 'abc' |> charToRaw()
enc <- sm2_encrypt_c1c2c3(data, public_key)
enc
dec <- sm2_decrypt_c1c2c3(enc, private_key)
dec
```

sm2_encrypt_hex	<i>SM2 Encrypt/Decrypt - hex and base64</i>
-----------------	---

Description

For ease of use, we have provided functions to encrypt data into hex or base64 format and decrypt them from these formats.

Usage

```
sm2_encrypt_hex(data, public_key)
sm2_decrypt_hex(data, private_key)
sm2_encrypt_base64(data, public_key)
sm2_decrypt_base64(data, private_key)
```

Arguments

data	for encrypt, data is a raw vector, for decrypt, data is a hex or base64 string
public_key	a public key represented as a hexadecimal string
private_key	a private key represented as a hexadecimal string

Value

sm2_encrypt_hex returns a hex string of the cipher text
sm2_decrypt_hex returns a raw vector of the plain text
sm2_encrypt_base64 returns a base64 string of the cipher text
sm2_decrypt_base64 returns a raw vector of the plain text

Examples

```
## encrypt and decrypt from hex string
keypair <- sm2_gen_keypair()
private_key <- keypair$private_key
public_key <- keypair$public_key
data <- 'abc' |> charToRaw()
enc <- sm2_encrypt_hex(data, public_key)
enc
dec <- sm2_decrypt_hex(enc, private_key)
dec
enc <- sm2_encrypt_base64(data, public_key)
enc
dec <- sm2_decrypt_base64(enc, private_key)
dec
```

sm2_encrypt_to_file *SM2 Encrypt/Decrypt - file*

Description

For ease of use, we have provided functions to encrypt or decrypt data directly from files.

Usage

```
sm2_encrypt_to_file(data, enc_file, public_key)
```

```
sm2_decrypt_from_file(dec_file, private_key)
```

Arguments

data	data to be encrypted, must be a raw vector
enc_file	the encrypted file to be saved
public_key	a public key represented as a hexadecimal string
dec_file	the encrypted file to be loaded
private_key	a private key represented as a hexadecimal string

Value

sm2_encrypt_to_file returns nothing, an encrypted file will be saved in the specified path

sm2_decrypt_from_file returns nothing, a decrypted file will be saved in the specified path

Examples

```
## encrypt and decrypt from file
## Not run:
data <- 'abc' |> charToRaw()
keypair <- sm2_gen_keypair()
private_key <- keypair$private_key
public_key <- keypair$public_key
sm2_encrypt_to_file(data, 'data.enc', public_key)
sm2_decrypt_from_file('data.enc', private_key)

## End(Not run)
```

sm2_gen_keypair	<i>SM2 Key Pair</i>
-----------------	---------------------

Description

In the SM2 encryption algorithm, the private key and public key appear in pairs. The private key is a 64-bit hexadecimal string, and the public key is a 128-bit hexadecimal string, excluding the "04" prefix at the beginning. The public key is included in the private key and can be derived from the private key. We use the public key for encryption, the private key for decryption, the private key for signing, and the public key for verification.

Usage

```
sm2_gen_keypair()  
  
sm2_pk_from_sk(private_key)  
  
sm2_privkey_valid(private_key)  
  
sm2_pubkey_valid(public_key)  
  
sm2_keypair_from_pem_file(pem_file)  
  
sm2_keypair_to_pem_file(private_key, pem_file)  
  
sm2_pubkey_from_pem_file(pem_file)  
  
sm2_pubkey_to_pem_file(public_key, pem_file)
```

Arguments

private_key	a private key represented as a hexadecimal string
public_key	a public key represented as a hexadecimal string
pem_file	local pem file path

Details

sm2_gen_keypair generate a random key pair
sm2_pk_from_sk export public key from a private key
sm2_privkey_valid check whether a private key is legal
sm2_pubkey_valid check whether a public key is legal
sm2_keypair_from_pem_file import private key from a local pem file
sm2_keypair_to_pem_file save a private key to a local pem file
sm2_pubkey_from_pem_file import public key from a local pem file
sm2_pubkey_to_pem_file save a public key to a local pem file

Value

- sm2_gen_keypair** returns a list contains a random private key and the corresponding public key
- sm2_pk_from_sk** returns a character string, the public key exported from a private key
- sm2_privkey_valid** returns 1 if valid, 0 if invalid
- sm2_pubkey_valid** returns 1 if valid, 0 if invalid
- sm2_keypair_from_pem_file** returns a list contains a random private key and the corresponding public key
- sm2_keypair_to_pem_file** returns nothing, and a local file contains the keypair will be saved in the specified path
- sm2_pubkey_from_pem_file** returns a character string, the public key saved in the local file
- sm2_pubkey_to_pem_file** returns nothing, and a local file contains the public key will be saved in the specified path

Examples

```
## generate a random keypair
keypair <- sm2_gen_keypair()
keypair$private_key
keypair$public_key
## export public key from private key
sm2_pk_from_sk(keypair$private_key)
## check whether the private key is legal
sm2_privkey_valid(keypair$private_key)
## check whether the public key is legal
sm2_pubkey_valid(keypair$public_key)
## Not run:
  sm2_keypair_to_pem_file(keypair, 'keypair.pem')
  sm2_keypair_from_pem_file('keypair.pem')
  sm2_pubkey_to_pem_file(keypair$public_key, 'pubkey.pem')
  sm2_pubkey_from_pem_file('pubkey.pem')

## End(Not run)
```

sm2_keyexchange_1ab *SM2 Key Exchange*

Description

SM2 is an asymmetric encryption algorithm, therefore, it can also be used for key agreement or key exchange. If A and B want to generate a recognized key for encryption or authentication, this algorithm can ensure that the key itself will not be transmitted through untrusted channels, and the private keys of A and B will not be disclosed. Even if an attacker intercepts the data exchanged by A and B, they cannot calculate the key agreed upon by A and B.

Usage

```
sm2_keyexchange_1ab(klen, id, private_key)
sm2_keyexchange_2a(id, private_key, private_key_r, recive_bytes)
sm2_keyexchange_2b(id, private_key, private_key_r, recive_bytes)
```

Arguments

klen	the key length, must be an integer
id	id of A or B, must be a raw vector
private_key	private key of A or B represented as a hexadecimal string
private_key_r	temp private_key of A or B
recive_bytes	for A or B, the recived data from B or A

Value

sm2_keyexchange_1ab returns a list, 'data' for the raw data sent to B(for A) or A(for B), 'private_key_r' for the temporary private key

sm2_keyexchange_2a returns a list, 'k' for the key of length 'klen', 's12' for the sm3 hash in asn.1 encoding

sm2_keyexchange_2b returns a list, 'k' for the key of length 'klen', 's12' for the sm3 hash in asn.1 encoding

Examples

```
## Step 1
klen <- 16
id_a <- "a@company.com" |> charToRaw()
id_b <- "b@company.com" |> charToRaw()
private_key_a <- sm2_gen_keypair()$private_key
private_key_b <- sm2_gen_keypair()$private_key
step_1_a <- sm2_keyexchange_1ab(klen, id_a, private_key_a)
step_1_b <- sm2_keyexchange_1ab(klen, id_b, private_key_b)

## Step 2
step_2_a <- sm2_keyexchange_2a(id_a, private_key_a, step_1_a$private_key_r, step_1_b$data)
step_2_b <- sm2_keyexchange_2b(id_b, private_key_b, step_1_b$private_key_r, step_1_a$data)
step_2_a$k
step_2_b$k
```

 sm2_sign

SM2 Sign/Verify

Description

SM2 is an asymmetric encryption algorithm, so it can be used to sign and verify signatures of data. The purpose of doing this is to ensure the integrity of the data and guarantee its authenticity. Typically, the data owner uses the SM3 message digest algorithm to calculate the hash value and signs it with the private key, generating signed data. Then the owner distributes the original data and the signed data of the original data to the receiver. The receiver uses the public key and the received signed data to perform the verification operation. If the verification is successful, it is considered that the received original data has not been tampered with.

Usage

```
sm2_sign(id, data, private_key)
sm2_verify(id, data, sign, public_key)
sm2_sign_to_file(id, data, sign_file, private_key)
sm2_verify_from_file(id, data, sign_file, public_key)
```

Arguments

id	the signer's id, must be a raw vector
data	original data, must be a raw vector
private_key	a private key represented as a hexadecimal string
sign	sign data of the original data or file
public_key	a public key represented as a hexadecimal string
sign_file	file path of the sign data to load

Value

sm2_sign returns a raw vector contains the signature
sm2_verify returns 1 if verified, 0 if not verified
sm2_sign_to_file returns nothing, and a signature file will be saved in the specified path
sm2_verify_from_file returns 1 if verified, 0 if not verified

Examples

```
## sign and verify
id <- charToRaw('yumeng@company.com')
data <- charToRaw('abc')
keypair <- sm2_gen_keypair()
```

```

private_key <- keypair$private_key
public_key <- keypair$public_key
sign_data <- sm2_sign(id, data, private_key)
verify_result <- sm2_verify(id, data, sign_data, public_key)
## Not run:
  sm2_sign_to_file(id, data, 'sign_data.sig', private_key)
  sm2_verify_from_file(id, data, 'sign_data.sig', public_key)

## End(Not run)

```

sm3_hash

SM3 Hash

Description

SM3 is a cryptographic hash function designed for digital signatures and other cryptographic applications. The output of SM3 is a 256-bit hash value, which is commonly represented as a 64-hexadecimal digit string.

Usage

```
sm3_hash(msg)
```

```
sm3_hash_string(msg_string)
```

```
sm3_hash_file(file_path)
```

Arguments

msg	data to be hashed
msg_string	a character string to be hashed
file_path	a local file to be hashed

Details

All the functions mentioned - [sm3_hash](#), [sm3_hash_string](#), and [sm3_hash_file](#) - return a 64-character hexadecimal string representing the 256-bit hash value generated by the SM3 cryptographic hash function. This hexadecimal string is commonly used to represent the hash output in a human-readable format. The [sm3_hash](#) function calculates the SM3 hash of a raw vector input and returns a 64-character hexadecimal string. Similarly, [sm3_hash_string](#) takes a string as input and also returns a 64-character hexadecimal string representing the SM3 hash of the input string. The [sm3_hash_file](#) function, on the other hand, takes a file path as input, reads the contents of the file, calculates its SM3 hash, and returns the corresponding 64-character hexadecimal string.

Value

a 64-characters hex string, which will be the sm3 hash result of the data, string or file

Examples

```
## Raw vector hashing
msg <- charToRaw('abc')
sm3_hash(msg)

## character string hashing
sm3_hash_string('abc')

## local file hashing
## Not run:
  sm3_hash_file('test.docx')

## End(Not run)
```

sm4_encrypt_ecb

SM4 Encrypt/Decrypt

Description

The SM4 algorithm is a block symmetric encryption algorithm with a block size and key length of 128 bits. Compared to the SM2 algorithm, it has higher encryption and decryption efficiency and can be used to encrypt larger amounts of data. SM4 supports both the ECB (Electronic Codebook) mode and the CBC (Cipher Block Chaining) mode. The ECB mode is a simple block cipher encryption mode that encrypts each data block independently without depending on other blocks. The CBC mode, on the other hand, is a chained block cipher encryption mode where the encryption of each block depends on the previous ciphertext block. Therefore, it requires an initialization vector (IV) of the same 128-bit length. The CBC mode provides higher security than the ECB mode.

Usage

```
sm4_encrypt_ecb(input_data, key)

sm4_decrypt_ecb(input_data, key)

sm4_encrypt_cbc(input_data, key, iv)

sm4_decrypt_cbc(input_data, key, iv)
```

Arguments

input_data	data bytes to be encrypted, must be a raw vector
key	the key, must be a raw vector of length 16
iv	the initialization vector, must be a raw vector of 16

Value

sm4_encrypt_ecb returns a raw vector of the cipher text using ecb mode

sm4_decrypt_ecb returns a raw vector of the plain text

sm4_encrypt_cbc returns a raw vector of the cipher text using cbc mode

sm4_decrypt_cbc returns a raw vector of the plain text

Examples

```
## ecb mode
data <- 'abc' |> charToRaw()
key <- '1234567812345678' |> charToRaw()
iv <- '0000000000000000' |> charToRaw()
enc <- sm4_encrypt_ecb(data, key)
enc
dec <- sm4_decrypt_ecb(enc, key)
dec
## cbc mode
enc <- sm4_encrypt_cbc(data, key, iv)
enc
dec <- sm4_decrypt_cbc(enc, key, iv)
dec
```

sm4_encrypt_ecb_base64

SM4 Encrypt/Decrypt - hex and base64

Description

For ease of use, we have provided functions to encrypt data into hex or base64 format and decrypt them from these formats.

Usage

sm4_encrypt_ecb_base64(input_data, key)

sm4_encrypt_ecb_hex(input_data, key)

sm4_decrypt_ecb_base64(input_data, key)

sm4_decrypt_ecb_hex(input_data, key)

sm4_encrypt_cbc_base64(input_data, key, iv)

sm4_encrypt_cbc_hex(input_data, key, iv)

sm4_decrypt_cbc_base64(input_data, key, iv)

sm4_decrypt_cbc_hex(input_data, key, iv)

Arguments

input_data	for encrypt, data is a raw vector, for decrypt, data is a hex or base64 string
key	the key, must be a raw vector of length 16
iv	the initialization vector, must be a raw vector of 16

Value

sm4_encrypt_ecb_base64 returns a base64 string of the cipher text using ecb mode

sm4_encrypt_ecb_hex returns a hex string of the cipher text using ecb mode

sm4_decrypt_ecb_base64 returns a raw vector of the plain text

sm4_decrypt_ecb_hex returns a raw vector of the plain text

sm4_encrypt_cbc_base64 returns a base64 string of the cipher text using cbc mode

sm4_encrypt_cbc_hex returns a hex string of the cipher text using cbc mode

sm4_decrypt_cbc_base64 returns a raw vector of the plain text

sm4_decrypt_cbc_hex returns a raw vector of the plain text

Examples

```
## SM4 Encrypt/Decrypt - hex and base64
data <- 'abc' |> charToRaw()
key <- '1234567812345678' |> charToRaw()
iv <- '0000000000000000' |> charToRaw()
## ecb mode
enc <- sm4_encrypt_ecb_base64(data, key)
enc
dec <- sm4_decrypt_ecb_base64(enc, key)
dec
enc <- sm4_encrypt_ecb_hex(data, key)
enc
dec <- sm4_decrypt_ecb_hex(enc, key)
dec
## cbc mode
enc <- sm4_encrypt_cbc_base64(data, key, iv)
enc
dec <- sm4_decrypt_cbc_base64(enc, key, iv)
dec
enc <- sm4_encrypt_cbc_hex(data, key, iv)
enc
dec <- sm4_decrypt_cbc_hex(enc, key, iv)
dec
```

sm4_encrypt_ecb_to_file
SM4 Encrypt/Decrypt - file

Description

For ease of use, we have provided functions to encrypt or decrypt data directly from files.

Usage

```
sm4_encrypt_ecb_to_file(input_file, output_file, key)
sm4_decrypt_ecb_from_file(input_file, output_file, key)
sm4_encrypt_cbc_to_file(input_file, output_file, key, iv)
sm4_decrypt_cbc_from_file(input_file, output_file, key, iv)
```

Arguments

input_file	the original file for encrypt, or the encrypted file for decrypt
output_file	the encrypted file for encrypt, or the decrypted file for decrypt
key	the key, must be a raw vector of length 16
iv	the initialization vector, must be a raw vector of 16

Value

sm4_encrypt_ecb_to_file returns nothing, and an encrypted file will be saved in the specified path using ecb mode

sm4_decrypt_ecb_from_file returns nothing, and a decrypted file will be saved in the specified path using ecb mode

sm4_encrypt_cbc_to_file returns nothing, and an encrypted file will be saved in the specified path using cbc mode

sm4_decrypt_cbc_from_file returns nothing, and a decrypted file will be saved in the specified path using cbc mode

Examples

```
## Not run:
key <- '1234567812345678' |> charToRaw()
iv <- '0000000000000000' |> charToRaw()
## ecb mode
sm4_encrypt_ecb_to_file('a.txt', 'a.enc', key)
sm4_decrypt_ecb_from_file('a.enc', 'a.dec', key)
## cbc mode
sm4_encrypt_cbc_to_file('a.txt', 'a.enc', key, iv)
```

```
sm4_decrypt_cbc_from_file('a.enc', 'a.dec', key, iv)
## End(Not run)
```


Index

sm2_decrypt, [2](#)
sm2_decrypt (sm2_encrypt), [2](#)
sm2_decrypt_asna1, [3](#)
sm2_decrypt_asna1 (sm2_encrypt_asna1), [3](#)
sm2_decrypt_base64, [5](#)
sm2_decrypt_base64 (sm2_encrypt_hex), [5](#)
sm2_decrypt_c1c2c3, [4](#)
sm2_decrypt_c1c2c3
 (sm2_encrypt_c1c2c3), [4](#)
sm2_decrypt_from_file, [6](#)
sm2_decrypt_from_file
 (sm2_encrypt_to_file), [6](#)
sm2_decrypt_hex, [5](#)
sm2_decrypt_hex (sm2_encrypt_hex), [5](#)
sm2_encrypt, [2, 2](#)
sm2_encrypt_asna1, [3, 3](#)
sm2_encrypt_base64, [5](#)
sm2_encrypt_base64 (sm2_encrypt_hex), [5](#)
sm2_encrypt_c1c2c3, [4, 4](#)
sm2_encrypt_hex, [5, 5](#)
sm2_encrypt_to_file, [6, 6](#)
sm2_gen_keypair, [7, 7, 8](#)
sm2_keyexchange_1ab, [8, 9](#)
sm2_keyexchange_2a, [9](#)
sm2_keyexchange_2a
 (sm2_keyexchange_1ab), [8](#)
sm2_keyexchange_2b, [9](#)
sm2_keyexchange_2b
 (sm2_keyexchange_1ab), [8](#)
sm2_keypair_from_pem_file, [7, 8](#)
sm2_keypair_from_pem_file
 (sm2_gen_keypair), [7](#)
sm2_keypair_to_pem_file, [7, 8](#)
sm2_keypair_to_pem_file
 (sm2_gen_keypair), [7](#)
sm2_pk_from_sk, [7, 8](#)
sm2_pk_from_sk (sm2_gen_keypair), [7](#)
sm2_privkey_valid, [7, 8](#)
sm2_privkey_valid (sm2_gen_keypair), [7](#)
sm2_pubkey_from_pem_file, [7, 8](#)
sm2_pubkey_from_pem_file
 (sm2_gen_keypair), [7](#)
sm2_pubkey_to_pem_file, [7, 8](#)
sm2_pubkey_to_pem_file
 (sm2_gen_keypair), [7](#)
sm2_pubkey_valid, [7, 8](#)
sm2_pubkey_valid (sm2_gen_keypair), [7](#)
sm2_sign, [10, 10](#)
sm2_sign_to_file, [10](#)
sm2_sign_to_file (sm2_sign), [10](#)
sm2_verify, [10](#)
sm2_verify (sm2_sign), [10](#)
sm2_verify_from_file, [10](#)
sm2_verify_from_file (sm2_sign), [10](#)
sm3_hash, [11, 11](#)
sm3_hash_file, [11](#)
sm3_hash_file (sm3_hash), [11](#)
sm3_hash_string, [11](#)
sm3_hash_string (sm3_hash), [11](#)
sm4_decrypt_cbc, [13](#)
sm4_decrypt_cbc (sm4_encrypt_ecb), [12](#)
sm4_decrypt_cbc_base64, [14](#)
sm4_decrypt_cbc_base64
 (sm4_encrypt_ecb_base64), [13](#)
sm4_decrypt_cbc_from_file, [15](#)
sm4_decrypt_cbc_from_file
 (sm4_encrypt_ecb_to_file), [15](#)
sm4_decrypt_cbc_hex, [14](#)
sm4_decrypt_cbc_hex
 (sm4_encrypt_ecb_base64), [13](#)
sm4_decrypt_ecb, [13](#)
sm4_decrypt_ecb (sm4_encrypt_ecb), [12](#)
sm4_decrypt_ecb_base64, [14](#)
sm4_decrypt_ecb_base64
 (sm4_encrypt_ecb_base64), [13](#)
sm4_decrypt_ecb_from_file, [15](#)
sm4_decrypt_ecb_from_file
 (sm4_encrypt_ecb_to_file), [15](#)

sm4_decrypt_ecb_hex, [14](#)
sm4_decrypt_ecb_hex
 (sm4_encrypt_ecb_base64), [13](#)
sm4_encrypt_cbc, [13](#)
sm4_encrypt_cbc (sm4_encrypt_ecb), [12](#)
sm4_encrypt_cbc_base64, [14](#)
sm4_encrypt_cbc_base64
 (sm4_encrypt_ecb_base64), [13](#)
sm4_encrypt_cbc_hex, [14](#)
sm4_encrypt_cbc_hex
 (sm4_encrypt_ecb_base64), [13](#)
sm4_encrypt_cbc_to_file, [15](#)
sm4_encrypt_cbc_to_file
 (sm4_encrypt_ecb_to_file), [15](#)
sm4_encrypt_ecb, [12](#), [13](#)
sm4_encrypt_ecb_base64, [13](#), [14](#)
sm4_encrypt_ecb_hex, [14](#)
sm4_encrypt_ecb_hex
 (sm4_encrypt_ecb_base64), [13](#)
sm4_encrypt_ecb_to_file, [15](#), [15](#)