

CM-SDK-JS

1.0.0

SDK 使用说明

前置操作

1. 引入zjquantum-cm-sdk-1.0.0-release.js中的方法
2. 安装axios

使用流程

配置服务信息 -> 调用所需方法 -> 使用SDK返回的数据

功能

- 1.身份认证
- 2.根据所需密钥数量获取国密SM2非对称密钥
- 3.根据已知的密钥编号获取国密SM2非对称密钥
- 4.根据所需密钥数量获取国密SM4对称密钥
- 5.根据已知的密钥编号获取国密SM4对称密钥
- 6.国密SM4加密
- 7.国密SM4解密
- 8.国密SM3摘要计算
- 9.国密SM2签名计算
- 10.国密SM2签名验证
- 11.国密SM2加密算法
- 12.国密SM2解密算法

示例

0.初始化配置

```
1
2  const IP = 'IP地址' // 默认值: 'api.zjquantum.cn'
3  const port = '端口号' // 默认值: '80'
4  const clientId = 'clientId' //由SDK提供方提供
5  const clientSecret = 'clientSecret' //由SDK提供方提供
6  const pubKey = '公钥' //由SDK提供方提供
7  const priKey = '私钥' //由SDK提供方提供
8  let prefix = '' //由SDK提供方提供, 一般无需配置
9  zjquantumConfig(IP, port, prefix, clientId, clientSecret, pubKey, priKey)
```

1.身份认证

```
1  const Login = async () => {
2    try {
3      const loginData = await login()
4    } catch (error) {
5      console.log(error)
6    }
7  }
```

2.根据所需密钥数量获取国密SM2非对称密钥

```
1  const sm2GetKeyNum = async () => {
2    const resEncode = '输出结果编码规则' // 【HEX、BASE64】
3    const number = '数量' // 正整数
4    try {
5      const sm2KeyData = await getSm2KeyNum(resEncode,number)
6      const result = sm2KeyData // 返回数组形式的密钥对
7    } catch (error) {
8      console.log(error)
9    }
10 }
```

3.根据已知的密钥编号获取国密SM2非对称密钥

```
1  const sm2GetKeyById = async () => {
2    const resEncode = '输出结果编码规则' // 【HEX、BASE64】
3    let keyIds = [] // 密钥对ID（支持多个）
4    try {
5      const sm2KeyIdData = await getSm2KeyId(resEncode,keyIds)
6      const result = sm2KeyIdData // 返回数组形式的密钥对
7    } catch (error) {
8      console.log(error)
9    }
10 }
```

4.根据所需密钥数量获取国密SM4对称密钥

```
1  const sm4GetKeyByNum = async () => {
2    const resEncode = '输出结果编码规则' // 【HEX、BASE64】
3    const number = '数量' // 正整数
4    try {
5      const sm4KeyNumData = await getSm4KeyNum(number,resEncode)
6      const result = sm4KeyNumData // 返回数组形式的密钥
7    } catch (error) {
8      console.log(error)
9    }
10 }
```

5.根据已知的密钥编号获取国密SM4对称密钥

```
1  const sm4GetKeyById = async () => {
2    const resEncode = '输出结果编码规则' // 【HEX、BASE64】
3    let keyIds = [] // 密钥ID（支持多个）
4    try {
5      const sm4KeyIdData = await getSm4KeyId(keyIds, resEncode)
6      const result = sm4KeyIdData // 返回数组形式的密钥
7    } catch (error) {
8      console.log(error)
9    }
10 }
```

6.国密SM4加密（对称算法加解密）

```
1  const sm4Encrypt = async () => {
2    const plaintextEncode = '原文数据编码规则' // 【UTF8、HEX、BASE64】
3    const keyEncode = '密钥编码规则' // 【HEX、BASE64】
4    const ivEncode = 'IV编码规则' // 【HEX、BASE64】
5    const resEncode = '输出结果编码规则' // 【HEX、BASE64】
6    const plaintext = '原文'
7    const key = '密钥'
8    const mode = 'ECB/CBC'
9    const IV = 'IV向量（CBC模式需要输入IV向量）'
10   try {
11     const encryptedData = await symEncData(plaintextEncode, keyEncode,
12       ivEncode, resEncode, mode, plaintext, key, IV)
13     const result = encryptedData // 加密后的密文
14   } catch (error) {
15     console.log(error)
16   }
17 }
```

7.国密SM4解密

```
1  const sm4Decrypt = async () => {
2    const ciphertextEncode = '密文数据编码规则' // 【HEX、BASE64】
3    const keyEncode = '密钥编码规则' // 【HEX、BASE64】
4    const ivEncode = 'IV编码规则' // 【HEX、BASE64】
5    const resEncode = '输出结果编码规则' // 【UTF8、HEX、BASE64】
6    const ciphertext = '密文'
7    const key = '密钥'
8    const mode = 'ECB/CBC'
9    const IV = 'IV向量（CBC模式需要输入IV向量）'
10   try {
11     const decryptedData = await symDecData(ciphertextEncode, keyEncode,
12       ivEncode, resEncode, mode, ciphertext, key, IV)
13     const result = decryptedData // 解密后的原文
14   } catch (error) {
15     console.log(error)
16   }
17 }
```

8.国密SM3摘要计算

```
1  const sm3Hash = async () => {
2    const messageEncode = '请求数据编码规则' // 【UTF8、HEX、BASE64】
3    const resEncode = '输出结果编码规则' // 【HEX、BASE64】
4    const message = '计算原文'
5    try {
6      const hashData = await hash(messageEncode, resEncode, message)
7      const result = hashData // 计算后的结果
8    } catch (error) {
9      console.log(error)
10   }
11 }
```

9.国密SM2签名计算

```
1  const sm2Sign = async () => {
2    const plaintextEncode = '待签名数据编码规则' // 【UTF8、HEX、BASE64】
3    const keyEncode = '公私钥编码规则' // 【HEX、BASE64】
4    const userIDEncode = '用户ID编码规则' // 【UTF8、HEX、BASE64】
5    const resEncode = '输出结果编码规则' // 【HEX、BASE64】
6    const plaintext = '待签名数据'
7    const publicKey = '公钥'
8    const privateKey = '私钥'
9    const userID = 'userID'
10   try {
11     const signData = await asymSign(plaintextEncode, keyEncode, userIDEncode,
12     resEncode, publicKey, privateKey, userID, plaintext)
13     const result = signData // 签名结果
14   } catch (error) {
15     console.log(error)
16   }
17 }
```

10.国密SM2签名验证

```
1  const sm2Verify = async () => {
2    const plaintextEncode = '验证数据编码规则' // 【UTF8、HEX、BASE64】
3    const keyEncode = '公钥编码规则' // 【HEX、BASE64】
4    const userIDEncode = '签名结果编码规则' // 【HEX、BASE64】
5    const signDataEncode = '用户ID编码规则' // 【UTF8、HEX、BASE64】
6    const plaintext = '验证数据'
7    const publicKey = '公钥'
8    const signData = '签名结果'
9    const userID = 'userID'
10   try {
11     const verifyData = await asymVerify(plaintextEncode, keyEncode,
12     userIDEncode, signDataEncode, publicKey, userID, signData, plaintext)
13     const result = verifyData // 验签结果 (true/false)
14   } catch (error) {
15     console.log(error)
16   }
17 }
```

11.国密SM2加密算法

```
1  const sm2Encrypt = async () => {
2    const plaintextEncode = '原文编码规则' // 【UTF8、HEX、BASE64】
3    const keyEncode = '公钥编码规则' // 【HEX、BASE64】
4    const resEncode = '输出结果编码规则' // 【HEX、BASE64】
5    const plaintext = '原文'
6    const publicKey = '公钥'
7    try {
8      const encryptedData = await asymEncData(plaintextEncode, keyEncode,
        resEncode, publicKey, plaintext)
9      const result = encryptedData // 加密后的密文
10   } catch (error) {
11     console.log(error)
12   }
13 }
```

12.国密SM2解密算法

```
1  const sm2Decrypt = async () => {
2    const ciphertextEncode = '密文编码规则' // 【HEX、BASE64】
3    const keyEncode = '私钥编码规则' // 【HEX、BASE64】
4    const resEncode = '输出结果编码规则' // 【UTF8、HEX、BASE64】
5    const ciphertext = '密文'
6    const privateKey = '私钥'
7    try {
8      const decryptedData = await asymDecData(ciphertextEncode, keyEncode,
        resEncode, privateKey, ciphertext)
9      const result = decryptedData // 解密后的原文
10   } catch (error) {
11     console.log(error)
12   }
13 }
```