云核客户端国密 SSL
（鸿蒙版）
用户手册

北京云核网络技术有限公司

2025 年 1 月 22 日

文档修改记录

版本	内容	编写人	编写日期	审核人	审核日期
1.0.1	初版	changbf	2024-10-12		
1.0.1	savefile 更换为 download，添加 upload 上传	changbf	2024-10-24		
1.0.2	使用异步交易，提升交易效率	changbf	2024-11-02		
1.0.3	添加 clear，可在登出等场景清楚 cookie	changbf	2024-11-05		
1.0.4	form 请求使用 curl_mime 代替 curl_formadd	changbf	2024-11-20		
1.0.5	cookie 清除中添加已有 easy handle 的清除	changbf	2024-12-04		
1.1.0	正式文档发布,同时增加异步和同步，适应不同场景	changbf	2024-12-13		

目　录

1. 引言

1.1 编写目的

国密 SSL 协议在 GM/T 中没有单独规范的文件，而是在 SSL VPN 技术规范中定义了国密 SSL 协议。GMT 0024-2014《SSL VPN 技术规范》中，国密 SSL 协议内容参照传输层安全协议（RFC 4346 TLS1.1），按照我国相关密码政策和法规，结合我国实际应用需求及实践经验，在 TLS 1.1 的握手协议中，增加了 ECC、IBC 的认证模式和密钥交换模式，取消了 DH 密钥交换方式，修改了密码套件的定义，另外就是增加了网关到网关协议。

国密 SSL 协议包括握手协议、密码规格变更协议、报警协议、网关到网关协议和记录层协议。握手协议用于身份鉴别和安全参数协商；密码规格变更协议用于通知安全参数的变更；报警协议用于关闭通知和对错误进行报警；网关到网关协议用于建立网关到网关的传输层隧道；记录层协议用于传输数据的分段、压缩及解压缩、加密及解密、完整性校验等。

国密 SSL 握手协议族包含密码规格变更协议、握手协议和报警协议 3 个子协议，用于通信双方协商出可供记录层使用的安全参数，进行身份验证以及向对方报告错误等。

国密 SSL 握手协议协商的会话包括：

- 会话标识：有服务端选取的随意的字节序列，用于识别活跃或可恢复的会话
- 证书：X.509 V3 格式的数字证书，符合 GM/T 0015
- 压缩方法：压缩数据的算法
- 密码规格：指定的密码算法
- 主密钥：客户端和服务端共享的 48 字节的密钥
- 重用标识：标明能否用该会话发起一个新连接的标识

利用以上数据可以产生安全参数，利用握手协议的重用特性，可以使用相同会话建立多个连接。

云核客户端国密 SSL 在原有国际 SSL 的基础上新增国密 SSL，为金融机构在 Android 手机上国密化提供解决方案。

1.2 背景知识及参考资料

假定开发人员对下列技术有一定的理解：

技术	有关内容
鸿蒙 SDK	鸿蒙 ETS 的基本知识
SSL	SSL 握手协议、SSL 密码参数修改协议、应用数据协议等
NAPI-node	实现 ArkTS/TS/JS 和 C/C++之间的交互

1.3 使用环境

鸿蒙星河版。

2. 概述

2.1 系统构成

云核客户端国密 SSL 鸿蒙版本以 HAR 格式提供，由两部分构成：

协议核心实现模块：由 C++实现的 SO 动态库。

接口模块：由 TS 实现。

2.2 功能特点

鸿蒙国密 SSL 使用静态共享保（HAR）的引用方式，SO 动态库实现算法主体，typescript 提供接口，通过 HAR 实现国密 SSL 通讯，具备以下基本特征：

- 支持 POST、GET、PUT、DELETE、HEAD、TRACE、OPTIONS 请求
- 支持文件下载和进度提示
- 支持国密 SSL 单向协议
- 支持国密 SSL 双向协议
- 支持 HTTP1.1/2/3
- 支持请求头设置
- 支持字符串/对象请求体
- 支持 SSL2/SSL3/TLS1.0/TLS1.1/TLS1.2/TLS1.3

2.3 技术特点

- 通过配置，提供国际、国密；
- 提供并发通讯；

2.4 接口描述

2.4.1 引用

对象	说明
cGmSSL	国密 SSL 对象创建接口
CertType	枚举，证书类型
HttpDataType	枚举，Http 数据类型
HttpProtocol	枚举，http 协议
RequestMethod	枚举，请求方法
ResponseCode	枚举，http 应答结果码
SSLPolicy	枚举，SSL 握手策略
SSLVersion	枚举，SSL 版本
CHttpClient	Http 请求对象
HttpGlobalOptions	全局设置
HttpRequestOptions	Http 请求选项
HttpResponse	Http 应该结果

2.4.2 cGmSSL

```
export class cGmSSL {
  /**
   * 国密 SSL 版本号
   * @returns 版本号
   */
  public static getVersion(): string {
    return ccGmSSL.getVersion()
  }

  /**
   * 创建 http client 句柄， 在不使用时自动释放
   * @returns httpclient 句柄
   */
  public static createCHttpClient(): CHttpClient {
    return ccGmSSL.createCHttpClient();
  }
}
```

2.4.3 HttpDataType

```
/**
 *  AUTO: 根据 response 返回的 MIME，确定数据类型
 *  STRING: 数据结果为字符串
 *  OBJECT: 数据结果为对象(Json 对象)
 *  ARRAY_BUFFER: 数据流
 */
export enum HttpDataType {
  AUTO = 0,
  STRING = 1,
  OBJECT = 2,
  ARRAY_BUFFER = 3
}
```

2.4.4 HttpProtocol

```
/**
 *  HTTP_DEFAULT: 不制定版本，由当前 Har 中决定，当前默认值问 1.1
 *  HTTP1_1: 采用 HTTP/1.1 进行通信。 HTTP/1.1 是一个非常成熟和广泛支持的协议版本，适用于大多数网络通信需求
 *  HTTP2: 采用 HTTP/2 进行通信
 *  HTTP3: 采用 HTTP/3 进行通信
 */
export enum HttpProtocol {
  HTTP_DEFAULT,
  HTTP1_1,
  HTTP2,
  HTTP3
}
```

2.4.5 RequestMethod

```
/**
 *  HTTP 请求方法
 *    GET:  没有请求体，因数据通过 URL 发送，使用时避免发送敏感数据。
 *    POST: 用于提交数据给服务器，例如提交表单或上传文件，数据通常作为请求体发送。
 *    PUT: 用于更新资源。它通常用于更新资源的全部内容，数据通常作为请求体发送。
 *    DELETE: 用于删除指定的资源，不需要请求体。
 *    HEAD: 与 GET 类似，但是它不返回请求的响应体，只返回头部信息。
 *    TRACE: 用于沿着到目标资源的路径执行一个消息回环测试。它回应了请求的最终接收者收到的原始请求，这样客户端可以看到中间代理添加或更改了哪些首部。
 *    OPTIONS: 用于描述目标资源的通信选项。可以通过这个方法来确定服务器支持哪些 HTTP 方法，或者针对特定资源支持哪些自定义的请求头。
```

```
 */
export enum RequestMethod {
  GET     = "GET",
  POST    = "POST",
  PUT     = "PUT",
  DELETE  = "DELETE",
  HEAD    = "HEAD",
  TRACE   = "TRACE",
  OPTIONS = "OPTIONS",
}
```

2.4.6 ResponseCode

```
/**
 *  响应码（Response Code）是服务器发送回客户端的数字状态代码，是 HTTP 请求处理结果。
 *   2xx: 成功状态码
 *    200 OK: 请求成功，服务器返回了请求的资源。
 *    201 Created: 请求成功，并且服务器创建了新的资源。
 *    202 Accepted: 请求已被接受处理，但处理尚未完成。
 *    203 Non-Authoritative Information: 服务器是一个转换代理服务器，它返回了资源
的原始响应的修改版。
 *    204 No Content: 请求成功，但没有返回任何内容。
 *    205 Reset Content: 请求成功，客户端应重置文档视图。
 *    206 Partial Content: 服务器成功处理了部分 GET 请求。
 *   3xx: 重定向状态码
 *    300 Multiple Choices: 资源有多种表示，用户可以选择一个。
 *    301 Moved Permanently: 请求的资源已永久移动到新位置。
 *    302 Found: 请求的资源临时移动到新位置。
 *    303 See Other: 服务器发送了一个新的位置，客户端应该使用 GET 请求它。
 *    304 Not Modified: 资源未修改，可以使用缓存版本。
 *    305 Use Proxy: 客户端应通过代理访问请求的资源。
 *    307 Temporary Redirect: 请求的资源临时移动到新位置，但应保持原始请求方法。
 *   4xx: 客户端错误状态码
 *    400 Bad Request: 服务器无法理解请求。
 *    401 Unauthorized: 请求需要用户身份验证。
 *    403 Forbidden: 服务器拒绝请求。
 *    404 Not Found: 请求的资源不存在。
 *    405 Method Not Allowed: 请求行中指定的方法不允许用于请求的资源。
 *    406 Not Acceptable: 服务器无法生成符合客户端要求的响应。
 *    407 Proxy Authentication Required: 客户端必须首先通过代理服务器进行身份验
证。
 *    408 Request Timeout: 服务器等待客户端发送的请求时间过长，请求超时。
 *    409 Conflict: 请求与服务器当前状态冲突。
 *    410 Gone: 请求的资源已被永久删除。
```

```
 *   5xx: 服务器错误状态码
 *    500 Internal Server Error: 服务器遇到了一个意外的情况，阻止它完成请求。
 *    501 Not Implemented: 服务器不支持请求的功能。
 *    502 Bad Gateway: 服务器作为网关或代理，从上游服务器收到了无效的响应。
 *    503 Service Unavailable: 服务器目前无法处理请求，通常是暂时性的。
 *    504 Gateway Timeout: 服务器作为网关或代理，没有及时从上游服务器收到响应。
 *    505 HTTP Version Not Supported: 服务器不支持请求的 HTTP 版本。
 */

export enum ResponseCode {
  OK = 200,
  CREATED,
  ACCEPTED,
  NOT_AUTHORITATIVE,
  NO_CONTENT,
  RESET_CONTENT,
  PARTIAL_CONTENT,
  MULT_CHOICE = 300,
  MOVED_PERM,
  MOVED_TEMP,
  SEE_OTHER,
  NOT_MODIFIED,
  USE_PROXY,
  BAD_REQUEST = 400,
  UNAUTHORIZED,
  PAYMENT_REQUIRED,
  FORBIDDEN,
  NOT_FOUND,
  BAD_METHOD,
  NOT_ACCEPTABLE,
  PROXY_AUTH,
  CLIENT_TIMEOUT,
  CONFLICT,
  GONE,
  LENGTH_REQUIRED,
  PRECON_FAILED,
  ENTITY_TOO_LARGE,
  REQ_TOO_LONG,
  UNSUPPORTED_TYPE,
  INTERNAL_ERROR = 500,
  NOT_IMPLEMENTED,
  BAD_GATEWAY,
  UNAVAILABLE,
  GATEWAY_TIMEOUT,
```

```
  VERSION
}
```

2.4.7 SSLPolicy

```
/**
 *  SSL 握手策略
 *   SSLPOLICY_DEFAULT: 默认
 *   SSLPOLICY_RSAThenSM2: 先 RSA 握手，然后 SM2 握手
 *   SSLPOLICY_RSAThenSM2: 先 SM2 握手，然后 RSA 握手
 */
export enum SSLPolicy {
  SSLPOLICY_DEFAULT,
  SSLPOLICY_RSAThenSM2,
  SSLPOLICY_SM2ThenRSA
}
```

2.4.8 SSLVersion

```
/**
 *  SSL 协议版本，默认国密协议
 */
export enum SSLVersion {
  SSLVERSION_DEFAULT,
  SSLVERSION_TLSv1, /* TLS 1.x */
  SSLVERSION_SSLv2,
  SSLVERSION_SSLv3,
  SSLVERSION_TLSv1_0,
  SSLVERSION_TLSv1_1,
  SSLVERSION_TLSv1_2,
  SSLVERSION_TLSv1_3,
  SSLVERSION_GMTLS      // GM SSL version(GM-T 0024-2014)
}
```

2.4.9 CHttpClient

```
/**
 *  Http Client 对象，用于发起交易请求
 */
export interface CHttpClient {

  /**
   * 使用回调，发起请求。 无 option
```

```
   * @param url      请求地址
   * @param callback 返回交易结果
   */
  request(url: string, callback: AsyncCallback<HttpResponse>): void;

  /**
   * 使用回调，发起请求
   * @param url      请求地址
   * @param options  请求选项
   * @param callback 返回交易结果
   */
  request(url: string, options: HttpRequestOptions, callback:
AsyncCallback<HttpResponse>): void;

  /**
   * 使用 Promise，发起请求
   * @param url       请求地址
   * @param options   请求选项
   * @returns         返回交易结果
   */
  request(url: string, options?: HttpRequestOptions):
Promise<HttpResponse>;


  /**
   * 设置请求的全局选项
   * @param option   全局选项
   */
  set(option: HttpGlobalOptions): void;
}
```

2.4.10 HttpGlobalOptions

```
/**
 *  http 请求全局选项
 */
export  interface HttpGlobalOptions {
  debug?: boolean,   default is false. if this parameter is true, Hilog
will print info
  readTimeout?: number; // Read timeout period. The default value is
60,000, in ms.
  connectTimeout?: number; // Connection timeout interval. The default
value is 60,000, in ms.
  usingProtocol?: HttpProtocol; // default is automatically specified by
```

```
the system.
  sslVersion?: SSLVersion;
  sslPolicy?: SSLPolicy;
  usingProxy?: boolean | connection.HttpProxy;

  caCerts?: string;

  sslVerify?: boolean; // If this parameter is set, it will via the
incoming url to verify host.
  maxLimit?: number;   // The maximum limit of the request .
}
```

2.4.11 HttpRequestOptions

```
/**
 *  单次请求的参数选项
 */
export  interface HttpRequestOptions {
  method?: RequestMethod;    // 默认 GET.
  extraData?: string | Object | Uint8Array;  //  body 数据，支持三种数据格式
  expectDataType?: HttpDataType;              // 期望返回的数据。当返回数据是
Object，可转换成希望的任何类型，当返回 string，可转成成希望的 string 和 array
  header?: Object | string;        // HTTP request header.
  readTimeout?: number;            // Read timeout 不设置使用全局
  connectTimeout?: number;         // Connection timeout 不设置使用全局
  realTimeReturn?: boolean;        // 实时返回数据，当文件存储时，只返回进度
  download?: string;               // 设置数据保存到文件/路径。当设置路径时，文件
名从返回 header 中的 Content-Disposition 获取，若获取不到生成随机文件名
  upload?: string;                 // 设置上传的文件路径，并在 header 中使用
filename 通知服务端上传的文件名

  usingProtocol?: HttpProtocol;  // http 协议
  sslVersion?: SSLVersion;       // TLS v1.0 or later.
  sslPolicy?: SSLPolicy;         // default use sslVersion.
  caCerts?: string;
}


/**
 * Defines the response to an HTTP request.
 */
export interface HttpResponse {
  result: string | Object | Uint8Array; // result can be  string /
Uint8Array / Object.
  resultLength: number;     // 结果长度
```

```
  receiveLength: number;    // 本次收到长度
  resultType: HttpDataType;// 结果数据类型
  responseCode: ResponseCode | number;       // Server status code.
  /**
   * All headers in the response from the server.
   */
  header: Object;       // All headers in the response from the server.
  cookies: string;      // Cookies returned by the server.
}
```

3. 开发流程概述

3.1 获取请求对象

```
import { cGmSSL } from '@security/cGmSSL';
let httpClient: CHttpClient = cGmSSL.createCHttpClient()
```

3.2 设置全局参数

```
// 全局设置
httpClient.set({
  debug: true,         // hilog 输出通讯过程
  readTimeout: 45,
  connectTimeout: 60,
  usingProtocol: HttpProtocol.HTTP_DEFAULT,
  sslVersion: SSLVersion.SSLVERSION_DEFAULT,
  caCerts:'',
  sslVerify: false,  // 不验证服务端证书和域名
  maxLimit: 10         // 同时并发最大 10 个
})
```

3.3 发起交易

```
httpClient.request(this.baseURL,
    {
      method: RequestMethod.GET,
      header: {'Content-Type': 'application/x-www-form-urlencoded',
'Connection': 'keep-alive'},
      // extraData: {"age":"16"},
      extraData: {},
      expectDataType: HttpDataType.ARRAY_BUFFER,
      readTimeout: 30,
      connectTimeout:40,
```

```
      usingProtocol: HttpProtocol.HTTP1_1,
      sslVersion: SSLVersion.SSLVERSION_GMTLS,
      sslPolicy: SSLPolicy.SSLPOLICY_DEFAULT,
    }, (err: BusinessError<void>, data: HttpResponse) => {
      if(!err){
        this.text = `${data.result}`
      } else {
        this.text = JSON.stringify(err);
      }
    })
})
```