# 外部养老金领取资格接口

## 接口说明

1. 下面提到的aes加密都是：aes-256-ecb-pkcs7
2. Nonce 随机串是16位（字母和数字组成）
3. 使用utf8编码
4. 请求方式：application/json 传参数也是json格式的
5. 本文后面提供AES加密校验

1) 接口URL `http://xxx/yyy` 或 `https://xxx/yyy`

2) 接口鉴权方式

> `appId+appSecret`鉴权，参见附录“接口鉴权”说明，`appId`，`appSecret`由接口提供方提供。

3) 敏感数据加密方式

> `AES`对称加密，加密`key`由接口提供方提供。

4) 请求方式 `POST`

5) 请求参数格式 `JSON`

6) 请求参数

| 参数名称 | 类型 | 描述 |
| --- | --- | --- |
| appId | String | appId |
| nonce | String | 随机串，参与signature签名算法，增加signature破解难度，和防止重放 |
| timestamp | String | UNIX时间戳，用于有效期判断 |
| signature | String | 鉴权签名 |
| idcard | String | 加密的身份证号码，aesEncrypt("身份证号码", key) |
| name | String | 加密的姓名，aesEncrypt("姓名", key) |

示例：

```
{
    appId: "appId",
    nonce: "随机串",
    timestamp: "UNIX时间戳",
    signature: "鉴权签名",
    idcard: "AES加密的身份证号码",
    name: "AES加密的姓名"
}
```

5) 响应参数格式 `JSON`

6) 响应参数

| 参数名称 | 描述 |
| --- | --- |
| errorCode | 错误码，0表示成功（养老金领取资格），其他异常 |
| errorMsg | 错误描述，成功时为“成功” |

示例：

```
{
    errorCode: 0,
    errorMsg: "成功"
}
```

# 附录

## 接口鉴权

接口鉴权通过请求参数中的appId，nonce，timestamp，signature判断。

| 参数名称 | 描述 |
| --- | --- |
| appId | 接口提供方分配的appId |
| appSecret | 接口提供方分配的appSecret |
| nonce | 随机串，参与signature签名算法，增加signature破解难度，和防重放 |
| timestamp | 时间戳，用于有效期判断 |
| signature | 由appId，appSecret，nonce，timestamp生成，参见“signature签名生成算法” |

### signature签名生成算法：

把appId，appSecret，nonce，timestamp值进行ASCII码从小到大排序（字典序）后为value1，value2，value3，value4拼接起来计算sha1值，即signature=sha1(value1value2value3value4)。

例如：

```
appId="8888"
appSecret="tencent"
nonce="random"
timestamp="1464059730"
signature=sha1("14640597308888randomtencent")="8818702782a887eec23b2d7279eb7c5eba6cc925"
```

接口提供方按此方式校验signature合法性，并可根据timestamp限制有效期（接口提供方自行决定是否要实现），可根据nonce防止重放（接口提供方自行决定是否要实现）。

## AES加密校验

使用项目提供的AES解密秘钥解密

### NodeJS 代码参考

AES-256-ECB + PKCS7

```
function encryptAes256ECBPKCS7(data, resultKey) {

    try {

        let iv = ;

        var clearEncoding = 'utf8';

        var cipherEncoding = 'base64';

        var cipherChunks = [];

        var cipher = crypto.createCipheriv('aes-256-ecb', resultKey, iv);

        cipher.setAutoPadding(true);

        cipherChunks.push(cipher.update(data, clearEncoding, cipherEncoding));

        cipherChunks.push(cipher.final(cipherEncoding));

        return cipherChunks.join('');

    } catch (e) {

        console.error(e);

        return "";

    }

}

function decryptAes256ECBPKCS7(data, resultKey) {

    try {

        if (!data) {

            return ;

        }

        let iv = ;

        var clearEncoding = 'utf8';

        var cipherEncoding = 'base64';

        var cipherChunks = [];

        var decipher = crypto.createDecipheriv('aes-256-ecb', resultKey, iv);


        decipher.setAutoPadding(true);
```

```
        let buff = data.replace('\r', '').replace('\n', '');
        cipherChunks.push(decipher.update(buff, cipherEncoding, clearEncoding));
        cipherChunks.push(decipher.final(clearEncoding));
        return cipherChunks.join('');
    } catch (e) {
        console.error(e);
        return ;
    }
}
```

### PHP 代码参考

AES-256-ECB +PKCS7（由于 PHP 底层的 256 和 Nodejs、Java 的不一样。所以 PHP 使用的是128长度）

```
/**
 * 利用mcrypt做AES加密解密
 */
class AES{
    /**
     * 算法,另外还有192和256两种长度
     */
    const CIPHER = MCRYPT_RIJNDAEL_128;
    /**
     * 模式
     * 1. MCRYPT_MODE_ECB(electronic codebook)
     适合对小数量随机数据的加密，比如加密用户的登录密码之类的。
     * 2. MCRYPT_MODE_CBC (cipher block chaining)
     适合加密安全等级较高的重要文件类型。
     * 3. MCRYPT_MODE_CFB (cipher feedback)
     适合于需要对数据流的每一个字节进行加密的场合。
     * 4. MCRYPT_MODE_OFB (output feedback, in 8bit) 和CFB模式兼容，但比C
     FB模式更安全。CFB模式会引起加密的错误扩散，如果一个byte出错，则其后
     续的所有byte都会出错。OFB模式则不会有此问题。但该模式的安全度不是很
     高，不建议使用。
     * 5. MCRYPT_MODE_NOFB (output feedback, in nbit)
     和OFB兼容，由于采用了块操作算法，安全度更高。
     * 6. MCRYPT_MODE_STREAM
     是为了WAKE或者RC4等流加密算法提供的额外模型。
     */
    const MODE = MCRYPT_MODE_ECB;

    /**
     * pkcs7补码
     * @param string $string  明文
     * @param int $blocksize Blocksize , 以 byte 为单位
     * @return String
     */
    private function addPkcs7Padding($string, $blocksize = 16) {
        $len = strlen($string); //取得字符串长度
        $pad = $blocksize - ($len % $blocksize); //取得补码的长度
        $string .= str_repeat(chr($pad), $pad); //用ASCII码为补码长度的字符， 补足最后一段
        return $string;
    }

    /**
     * 加密然后base64转码
     * @param $str
     * @param $key
     * @return string
     */
    function aes256cbcEncrypt($str,$key ) {
        $iv = mcrypt_create_iv(mcrypt_get_iv_size(self::CIPHER,self::MODE),MCRYPT_ENCRYPT);
        return base64_encode(mcrypt_encrypt(self::CIPHER, $key, $this->addPkcs7Padding($str) ,
self::MODE, $iv));
    }
```

```
    /**
     * 除去pkcs7 padding
     *
     * @param String 解密后的结果
     *
     * @return String
     */
    private function stripPkcs7Padding($string){
        $slast = ord(substr($string, -1));
        $slastc = chr($slast);
        $pcheck = substr($string, -$slast);

        if(preg_match("/$slastc{".$slast."}/", $string)){
            $string = substr($string, 0, strlen($string)-$slast);
            return $string;
        } else {
            return false;
        }
    }
    /**
     * 解密
     * @param String $encryptedText 二进制的密文
     * @param String $key 密钥
     * @return String
     */
    function aes256cbcDecrypt($encryptedText, $key) {
        $encryptedText =base64_decode($encryptedText);
        $iv = mcrypt_create_iv(mcrypt_get_iv_size(self::CIPHER,self::MODE),MCRYPT_ENCRYPT);
        return $this->stripPkcs7Padding(mcrypt_decrypt(self::CIPHER, $key, $encryptedText,
self::MODE, $iv));
    }
}
```