Snooper

APDU script program language

Programmer's Reference Manual

About this document

为实现 apdu 脚本简单化以及规范化，taoism 3 版对常规版的脚本工具进行了较大修改，本手册中涉及的软件为 snoopers taoism 3 版，适用于 0.0.6.6 以上版本。

目　录

简介

基本介绍

Snooper是为智能卡应用开发者提供测试数据的工具。Snooper的开发过程是随着智能卡应用开发需求的不断变化逐步完善的过程。Snooper脚本具有独特的风格，注重简单、可靠的脚本编写体验，支持动态编辑执行、自动计算、流程跳转、动态提示、自动完成、自定义过程等多种功能，积累了丰富的测试脚本。使用Snooper工具，可以高效进行智能卡开发与测试。

更新变化

0.0.5.9----删除了 snooper taoism 2 中若干历史遗留的特性，基本兼容常规版，详细信息见删除记录
0.0.5.9----修改了 rk4 专用通讯，增加了 rockey4_encode 和 rockey_decode
0.0.5.9----增加了部分 cbor 编解码函数
0.0.5.9----增加了两个关键字 inc_indent, dec_indent，用于控制脚本缩进
0.0.5.9----增加了简单截屏功能，快捷键 CTRL + SHIFT + T，CTRL + SHIFT + Y
0.0.5.9----修改了部分图标
0.0.5.9----增加了 blakehash 部分函数
0.0.5.9----增加了 memory 操作的部分函数
0.0.5.9----增加了未保存的脚本自动保存及恢复的功能，增加了 oid_encrypt，oid_decrypt 两个函数。
0.0.5.9----增加了变量、函数的部分匹配自动提示。
0.0.5.9----增加了 QrCode 相关函数 ScanQrCode。
0.0.5.9----修改了截屏功能的 bug。
0.0.5.9----加入了 crc32, rc4_crypt 函数
0.0.6.0----加入了部分 libusb 函数，部分 plc 控制函数，加入了少量全局变量功能。
0.0.6.0----加入了 getlv_scp11c 函数，此函数对 tag 判断做特殊处理。
0.0.6.0----加入了可以在不同显示器启动的功能，or 函数支持多参数。
0.0.6.0----加入了 settings(json)读取功能
0.0.6.0----给 txt2num 增加别名 parse,num2txt 增加别名 h2s。
0.0.6.0----在脚本级别和界面操作中加入 Scp11c 的部分支持。
0.0.6.0----给 expr10 增加别名 int，给 expr16 增加别名 hex
0.0.6.0----将第二套脚本中的 int 改为 dec（此为重大改动，可能涉及既往脚本不兼容）
0.0.6.0----将第二套脚本中的 to_int 改为 to_dec（此为重大改动，可能涉及既往脚本不兼容）
0.0.6.0----给脚本添加 max_response_length 内置变量，用于控制 get response 的最大长度。
0.0.6.0----给脚本添加 jcop22_ext_auth_nonblock，在 gp 认证失败时，会继续执行脚本供后续处理。
0.0.6.0----增加单线程可控断电 hub 控制函数。
0.0.6.0----修改 fido 通讯函数，加入 fido_90_apdu 前缀关键字。
0.0.6.0----修改了界面，增强了 tlv 解析时的 tree 与 list 的互动。

0.0.6.0----增加了 ccid 读卡器控制通道注册与部分 ccid 读卡器控制功能。
0.0.6.0----增加了 unix 时间戳与 utc 时间转换的功能。
0.0.6.0----增加单线程可控断电 hub 控制函数（此 hub 带外接电源）。
0.0.6.0----增加了 check_tlv，aes_ccm 等函数。
0.0.6.0----修改了拷贝成 Rtf 格式功能，拷贝长度太长时拷贝成普通文本。
0.0.6.1----增加了 Asn.1 view/editor 功能
0.0.6.1----增加了 aes_gcm 函数
0.0.6.1----增加了 asn1 拼接函数，增加 settlv 函数，可简单修改 asn1 证书。
0.0.6.1----增加了 gettlv_bypath, settlv_bypath, passport_calcdight 函数。
0.0.6.1----增加了 ext_euclid, big_div_u32, big_mod_u32 函数。
0.0.6.1----修改拷贝脚本默认成 rtf 格式的行为，改成拷贝成普通文本。
0.0.6.1----修改工具占用某些文件夹的行为
0.0.6.2----增加了 sm9 相关的函数以及计算工具，支持显示 apdu 时间。
0.0.6.2----增加了 wsq 格式数据显示以及转存为 bmp 的功能。
0.0.6.2----增加了 reverse_block_dowrd, reverse_block_bit, reverse_block_word 函数。
0.0.6.2----增加了 new_rsa_calculate_other_by_pqe，big_div，big_mod 函数，修改 big_mod_inv 函数。
0.0.6.2----rsa 相关函数支持 E 参数大于 32bit。
0.0.6.2----增加 big_mod_div, big_exp, big_gcd 等函数。
0.0.6.2----修改了 sm2_pri_docode 的描述,simple_dump 由 2 个参数改为 3 个参数
0.0.6.2----增加了 new_rsa_calculate_other_by_pqe 函数，脚本中调用更加简单
0.0.6.2----临时删除了 new_ecc_j2a 函数
0.0.6.2----增加了 new_ecc_length_random，取得当前 ecc 长度的随机数。
0.0.6.2----增加了 getnextprime，getprevprime 两个函数
0.0.6.2----修改了 new_sm2_keyexchange 函数的参数，参数减少，用起来更方便
0.0.6.2----增加了 new_sm2_get_yflag, new_sm2_computer_y, new_sm2_check_point
0.0.6.2----增加了 new_sm2_point_add, new_sm2_point_double, new_sm2_check_pubkey
0.0.6.2----增加了 new_sm2_ecdh_gm_map, new_sm2_kp, new_sm2_kp_add_lq, new_ecc_get_yflag
0.0.6.2----增加了 setwindowtext 函数
0.0.6.2----增加了 file_read_linehex, get_line_count, get_linehex_from_mem 函数
0.0.6.2----增加了 new_rsa_calc_other_by_nde, big_sqrt 函数
增加了 hmac 函数
增加了 ed25519 和 x25519 函数
增加了 blakehash 以及 hmac 函数
增加了 local 关键字
增加了 vpn_direct_onlysend, vpn_direct_sendrecv 两个函数
增加了 systemtime 的几个函数
修改了 upload 关键字的参数，可以控制 upload 函数的指令长度。
增加了 t1crc 这个函数，用于 t1 时块传输的 crc 校验，正确性未验证。
增加了 sm4_encode_gcm 和 sm4_decode_gcm 函数，采用修改 aes 的 gcm 的方法。
增加了 scrypt 函数，这是一个内存依赖函数，不同的参数会占用不同的内存。
去掉了 pbkdf2 相关的函数，去掉了 sha1_hmac 一族的函数，新增加了 pbkdf1, pbkdf2, hkdf, hmac 一组新函数。
修改了原来的 xxx_hash_init, xxx_hash_update, xxx_hash_dofinal 函数。

加入了电信部分鉴权函数。
按照 aes ccm 加入了 sm4 ccm 函数
0.0.6.5----修改了 rsa 系列的函数，支持**模长为 32（含）以上的偶数长度**。
0.0.6.5----增加了 new_rsa_calculate_d2 函数，用于计算可能 d 可能的其他值。
0.0.6.5----增加了 xor2 和 and2 两个很简单的函数。
0.0.6.5----将了 new_ecc_length_random，改名为 new_ecc_prilen_random。
0.0.6.5----增加了 new_ecc_prilen, new_ecc_publen 两个函数，取得私钥、公钥 X 或 Y 的长度。
0.0.6.5----增加了 urldownloadtofile，用于下载一些文件。
0.0.6.5----增加了 float 这个宏，用于解决一些浮点运算的问题。
0.0.6.5----增加了 rsa 相关的几个多线程函数。
0.0.6.5----将 a3a8 改为 a3a8_fun。
0.0.6.5----增加 replace_in_file 和 delete_in_file 两个函数。
0.0.6.5----增加 crc16_x25 函数。
0.0.6.5----增加 pkcs7_pad, pkcs7_unpad 函数。
0.0.6.5----增加 ansi_utf16_big, utf8_utf16_big, utf8_utf16_little 函数。
0.0.6.5----增加 pkcs12_sha1_diver_by_password
0.0.6.5----修改 new_sm2_sign 和 new_ecc_sign，签名的时候可以指定随机数
0.0.6.5----修改 sm3_otp 和 sm4_otp 函数名为 otp_sm3, otp_sm4
0.0.6.5----增加 hotp_sha1 和 hotp_sha256 两个函数。
0.0.6.5----将 hotp_sha1 和 hotp_sha256 改成 hotp_sha1_int 和 hotp_sha256_int,增加了 hotp_sha1_hex 和 htop_sha256_hex。
0.0.6.5----增加 totp_sha1_int、totp_sha256_int、totp_sha1_hex 和 totp_sha256_hex。
0.0.6.5----增加 chenqi_crc_mem 函数。
0.0.6.5----增加 apdu 窗口的全局热键管理窗口。
0.0.6.5----增加 get_reader_name, strstr, utf16_little_ansi。
0.0.6.6----升级了 scintilla 控件。
0.0.6.6----增加 loadlibrary, getprocaddress, freelibrary, dllfun 等 4 个函数，用于调用用户编写的 dll。
0.0.6.6----增加 rsa oaep 模式的一些加解密函数。
0.0.6.6----增加 rsa pss 模式的一些签名验签函数。
0.0.6.6----按sm9规范，修改了部分代码，将Ecb改成了cbc，对应4个新函数new_sm9_encrypt_block_2018, new_sm9_encrypt_stream_2018, new_sm9_decrypt_block_2018, new_sm9_decrypt_stream_2018
0.0.6.6----增加一些s2k函数。
0.0.6.6----增加crc_13239_lsb, crc_13239_msb函数

使用注意

1） 本程序是一个测试工具，有一定的编辑功能，日常情况下可以满足编辑要求，也可使用其他专业工具来编辑脚本，使用本程序执行脚本。

2） 本程序是一个类似多文档的工具，可以同时打开多个脚本，但是只存在一个工作环境，工作环

境中的多个脚本，逻辑上是统一的，允许互相调用。

3） 第一个被打开的称为主脚本，主脚本中可以使用 load_script 来打开辅助脚本。

4） load_script 语句只有在主脚本中对其他辅助脚本才有效。

5） load_script 语句只有在打开主脚本时检测，动态编辑的无效。

6） 主脚本和load_script 语句中所有涉及的脚本（无论是否存在）都会被工具监控是否被改动。

7） dummy函数，dummy函数是一个api接口，可调用snooper中的另一组函数，新的函数支持多层嵌套，函数名称与当前脚本函数有极大部分名称相同，所有使用时要仔细区分。

8） 一般情况下，打开新脚本时会检查现有脚本是否被改动，并提示保存，用户应按实际需求决定是否保存脚本。

9） 脚本编辑窗口具有定时保存临时文件功能。

多个 apdu 窗口的关系

Snooper 软件中目前有 10 个脚本窗口，编号为 0-9，按日常作用分为下面几类

1. 0，1，7，8-------用户编写脚本
2. 2----------------命令行
3. 3，9-------------平常不可见，用于C功能扩展
4. 4，5，6----------平常不可见，用于脚本功能扩展

窗口号	窗口层名称	窗口层级别	与下一层关系
0，1，7，8，2	用户层	0	0层用于与用户直接交互 在0层中执行某个脚本调用1层 在0层中扩展某个脚本的功能调用2层
4，5，6	辅助层	1	1层本质上与0层无差别，一般被0层窗口用来执行脚本文件,也可以被1层甚至2层调用。
3，9	扩展层	2	2层一般用来实现某个复杂功能，由于某种原因，这个功能没有用C代码实现，而使用脚本实现。

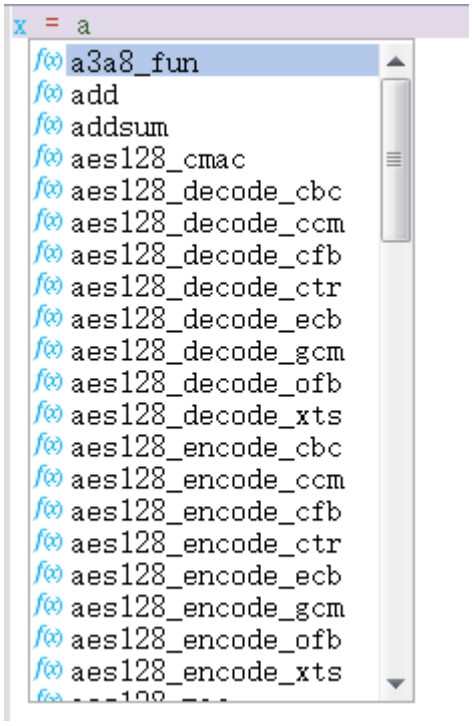
			一般在某个窗口内调用扩展层，使用3，如果是通用功能，则使用9，3可以调用9.

两套自成体系的函数

因为历史原因，Snooper 脚本中有两套函数，这两套函数中的大部分是相同的。

第一套函数

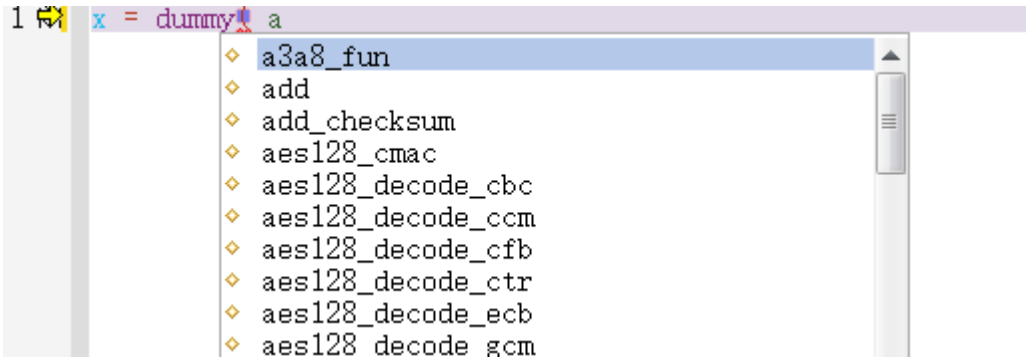
在脚本中，输入诸如 x = a



此时会有自动完成窗口弹出供使用者选择，这里面列出的函数，就是第一套函数

第二套函数

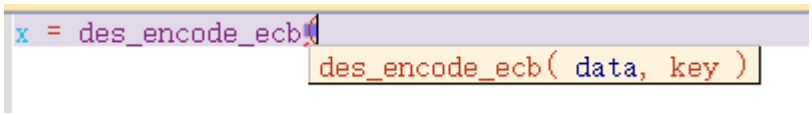
在脚本中输入诸如 x = dummy(a



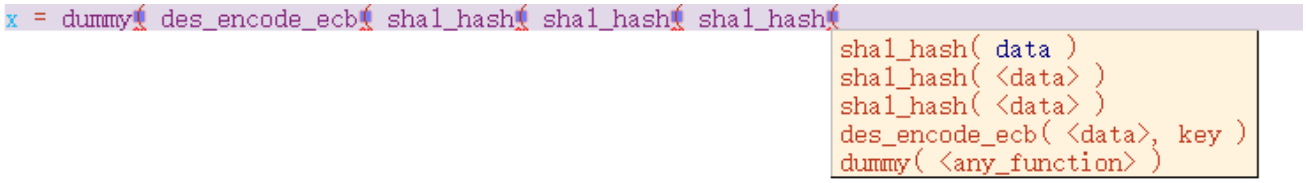
此时会有自动完成窗口弹出供使用者选择，这里面列出的函数，就是第二套函数

两套函数的差别

这两套函数，第一套是逐步迭代发展而来的，不支持嵌套调用，第二套则是试图改进并替代第一套的目的而编写的，支持嵌套调用，这套函数的整体设计较第一套函数有较大改进，在开发与维护上都相对容易。



举例，在第一套函数des_encode_ecb，第一个参数data，就不能是个函数，必须是常量或者变量。



而在第二套函数中，每个参数都可以是常量、变量、或者函数，比如可以使用一行脚本就把hmac实现。

两套函数的联系

考虑到脚本要简单、可读性高，一般不使用特别复杂的表达式，维护两套函数的成本也比较大，所以在后来的修改中，第一套函数调用的是第二套函数。

特别注意

前面说过，这套脚本框架是逐步迭代出来的，开始时只支持极其简单的静态脚本，后来慢慢的增加了

其他功能，出于尽量兼容性的原则，保留了很多历史功能，从而导致有些功能看起来很别扭，比如什么时候使用10进制，什么时候使用16进制……为解决这样的问题，不得不分别实现了10进制的函数和16进制的函数。

在第二套函数中，只存在16进制，如果要使用10进制参数，基本做法是使用 dummy(dec(2999))，比如a = dummy(dec(2999))，此时a的值

```
//a =  DEC ( 2999 ) =
//--DEC
//---   input = 2999  hex = 00 00 0B B7
    //00000BB7
```

这里的dec，表明dec内部输入值应该是10进制的，其输出值永远是16进制。同样第二套函数还有binary函数，例如：

```
a = dummy( binary( 00000101 ) )
```

而在第一套函数中，恰恰相反，expr10这个函数，不是要求输入是10进制数，而是其输出值是10进制数，至于输入值，可以是10进制数，也可以是16进制数。

同样expr16函数，表明输出值是16进制数。

而且为了用起来更加流畅，给expr10起了别名叫int，给expr16起了别名叫hex。

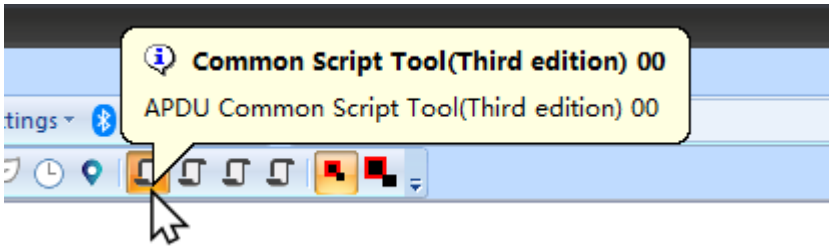
下面写一些示例

```
// expr10与int等价
// expr16与hex等价
x   = int( 40 )                 // x = 40的10进制值，也就是40
y   = hex( 40 )                 // x = 40的16进制值，也就是28
x   = int( 40, 3 )              // x = 40的10进制值，长度为3，也就是00 00 40
? int( 10000040, 305 )          // 将10000040，输出至少5*2位
                                //       数，并且每3个数字加一个逗号

x   = expr10( 10000040, 305 )   // x = 10000040的10进制值，长度为5，也就是0010000040

x = int( 0x158 )                // x = 0x158的10进制值
y = hex( $x )                   // 复杂的来了，y = 0x158的10进制值的16进制值
x = int( 0x$x )
x = int( 0x$x )
```

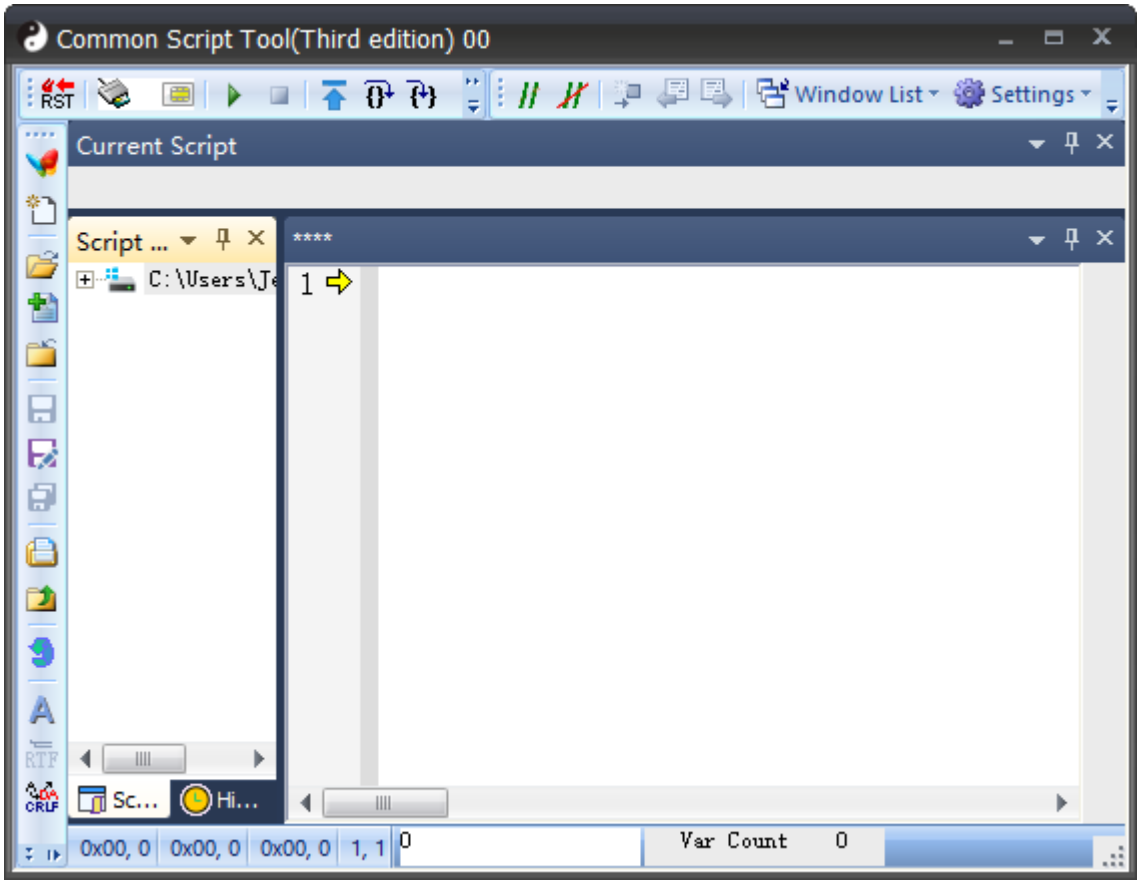
```
x = int( 0x$x )                    // 套娃计算      0x158
                                   // 的10进制值
                                   // 看成16进制数
                                   // 的10进制值
                                   // 看成16进制数的
                                   // 10进制值
```

脚本编辑窗口简介

打开脚本工具窗口



4个脚本窗口，供用户编写并执行脚本使用



这两个按钮，供用户进行脚本工作模式选择，左边是简单模式，脚本环境提供最多5个子脚本、256个变量。

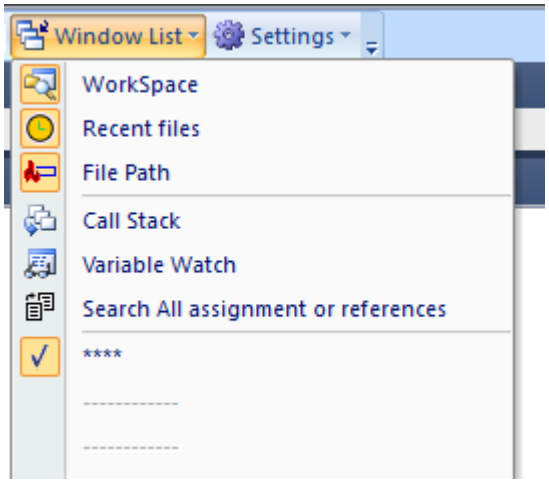
右边是普通模式，脚本环境提供最多16个子脚本，2048个变量。

顶部工具条简介

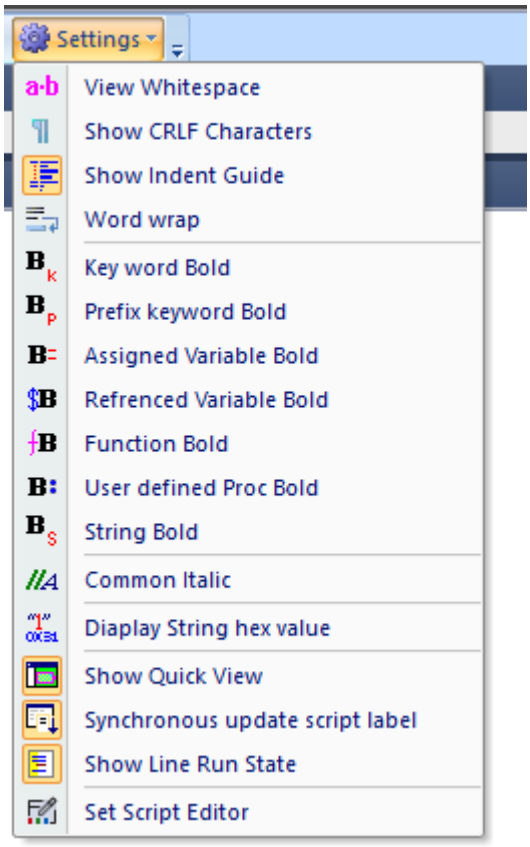
这几个按钮分别表示

- RST复位（热复位）
- 指示是否正在apdu通讯中
- 运行脚本，快捷键‘F5’或‘Ctrl + F5’，Ctrl + f5会自动保存log。
- 定位到脚本开始，在apdu窗口获得焦点的时候，快捷键‘F7’。
- 单步运行脚本，快捷键‘F10’。
- 逐行运行脚本，快捷键‘F11’。
- 运行到函数结束，此功能在脚本运行到子函数中时有效，快捷键‘Shift + F11’。
- 运行到光标所在行，快捷键‘Ctrl + F10’。
- 从最开始运行到光标所在行，快捷键‘Shift + F10’。
- 停止运行脚本，此按钮只有在脚本正在运行时生效
- 清除所有断点，快捷键‘Ctrl + F9’。
- 是否显示通讯时的apdu，可在脚本中使用 showapdu on/off 来控制
- 是否自动取响应，或重试指令，可在脚本中使用 auto_response on/off 来控制
- 是否显示变量的变化，可在脚本中使用 prompt on/off 来控制
- 打开脚本文件时，当前脚本所在的文件夹是否按GB来排序
- mac的长度是否是4，可在脚本中用 mac_length = 4 来设置
- mac的长度是否是8，可在脚本中用 mac_length = 8 来设置
- mac的长度是否是16，可在脚本中用 mac_length = 16 来设置

- 用 // 块注释所选中的文本，快捷键‘Ctrl + /’或 ‘Ctrl + \’
- 去掉选中文本的 // 注释，快捷键‘Ctrl + U’
- 跳转到函数或变量定义处，快捷键‘F12’。
- 跳转到上一个（前一个）位置，快捷键‘Alt + left’
- 跳转到下一个（后一个）位置，快捷键‘Alt + right’



- 显示或隐藏文件窗口
- 显示或隐藏文件历史记录
- 显示或隐藏文件路径窗口
- 堆栈信息
- 变量查看
- 引用信息查看
- 脚本文件列表



- 显示空白字符
- 显示不可见字符
- 显示缩进参考线

- 切换自动换行，快捷键‘Ctrl + K’
- ---
- 关键字加粗
- 前缀关键字加粗
- 被赋值变量加粗
- 被引用变量加粗
- 函数加粗
- 用户自定义函数加粗
- 字符串加粗
- ---
- 注释斜体
- ---
- 显示字符串的16进制值
- ---
- 鼠标快速查看文档图
- 在编辑脚本时同步更新用户自定义标号和变量
- 显示行运行状态
- 设置外部脚本编辑器

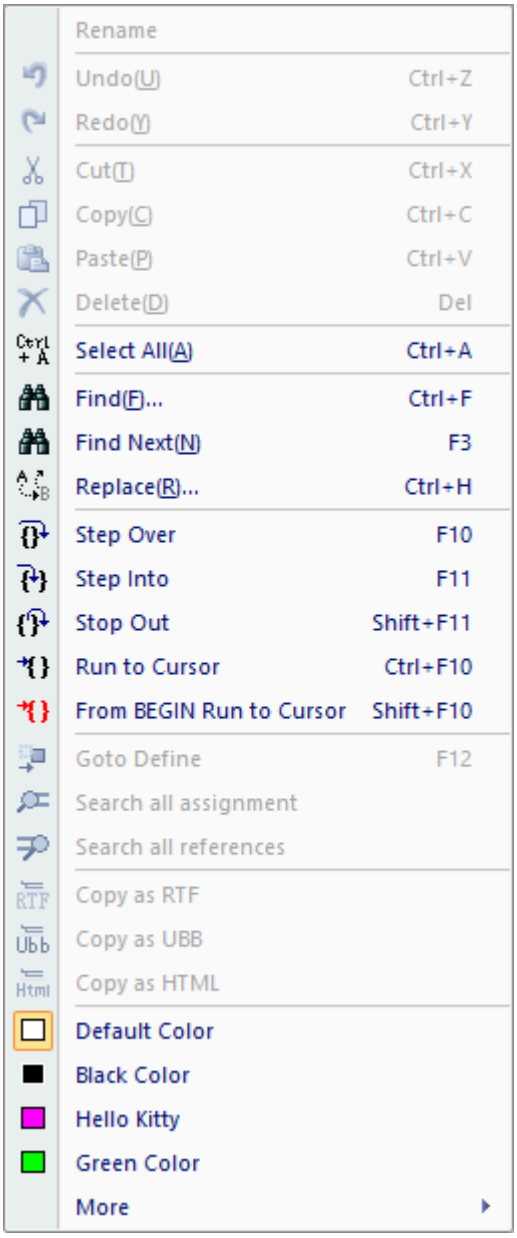
左部工具条简介

这几个按钮分别表示

- 关闭工作环境并新建脚本
- 新建脚本
- 打开以.txt为扩展名的脚本文件，并同时打开辅助脚本，形成工作环境
- 添加以.txt为扩展名的脚本文件到工作环境
- 保存脚本
- 另存脚本
- 保存全部脚本
- 从工作环境中移除脚本
- 关闭工作环境
- 打开当前脚本文件所在的文件夹
- 向上一级文件夹
- 字体设置
- 使用外部编辑工具编辑脚本，当编辑完成之后，需要重新加载脚本，如果还没有指定外部工具，会首先提示用户选择一个编辑工具，推荐使用ultraedit。
- 重新加载主脚本（并同时加载辅助脚本）
- 隐藏主窗口
- 将换行符换成crlf(\r\n)格式
- 拷贝成rtf格式

- 热键管理器

右键菜单



右键菜单用于一些常见操作，一个独立于工具条的功能就是拷贝选中的文本为RTF格式，此功能有助于文档的编写。

快捷键

脚本快捷键（脚本窗口要求是当前窗口）

F1	函数参数提示
F2	使用外部编辑工具编辑脚本（第一次使用时要求指定编辑工具）
F3	查找选中的内容
F4	文件相关函数、读卡器列表函数中，弹出选择列表。
F5	运行程序
Ctrl + F5	运行程序并保存日志
F6	运行焦点转到鼠标焦点所在的文本行
F7	运行焦点转到主脚本第一行
F9	断点
Ctrl + F7	去除所有行状态
Alt + F8	格式化当前脚本
Ctrl + F9	去除所有断点
F10	单步运行
Ctrl + F10	运行到光标所在行
Shift + F10	从开始运行到光标所在行
F11	逐行运行
Shift + F11	运行到函数结束
F12	跳转到用户函数定义或跳转到所有脚本中第一条给变量赋值的语句，按脚本加载次序，由上到下依次查找
Ctrl + S	保存文件
Ctrl + W	另存文件
Ctrl + F	查找对话框
ESC	普通状态下复位当前设备，有弹出窗口时取消窗口。
Alt + left	前一个位置
Alt + right	后一个位置
Ctrl + / 或 Ctrl + \	注释语句
Ctrl + U	取消语句注释
Ctrl + G	跳转到脚本某一行
Ctrl + K	自动换行切换
Alt + ‘<’	‘<’选中文字去掉空格
Alt + ‘>’	‘>’选中文字用空格分隔 注：选择一段文字，先 alt + ‘<’ 再 alt + ‘>’，可将此段文字格式化成 16 进制数
Alt + U	选中文字大写
Alt + L	选中文字小写

Alt + ‘”’	选中行加双引号
Alt + Shift + ‘”’	选中行去双引号
Alt + ‘?’	选中行加 “? “
Alt + Shift +’?’	选中行去 “? ”
Alt + M	删除行尾空格
Alt + T	选中行绘制表格
Alt + 8	选中文字转成 utf8_string
Alt + 2	选中文字转成 gb2312 字符串
Alt + 6	选中文字转成 utf16_little_string

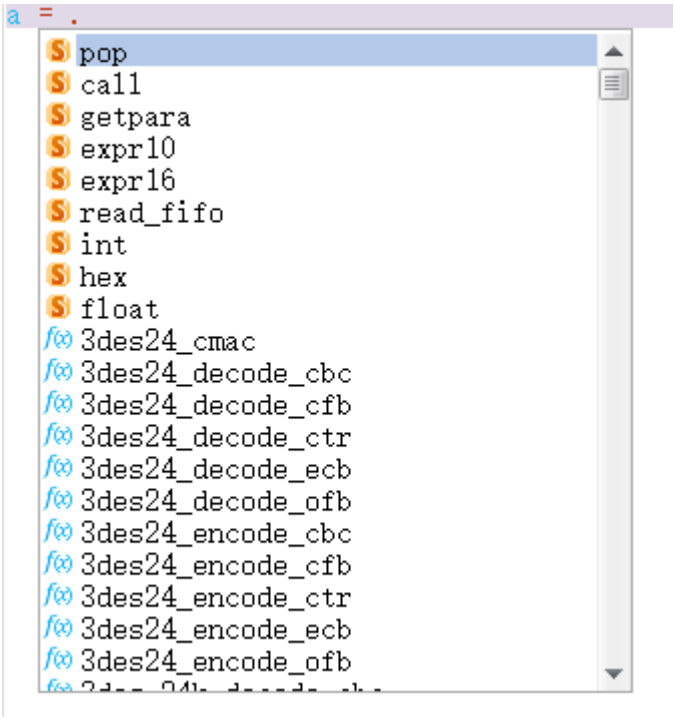
文件窗口快捷键（文件窗口要求是当前窗口）

F2	修改文件名
Del	删除文件到回收站

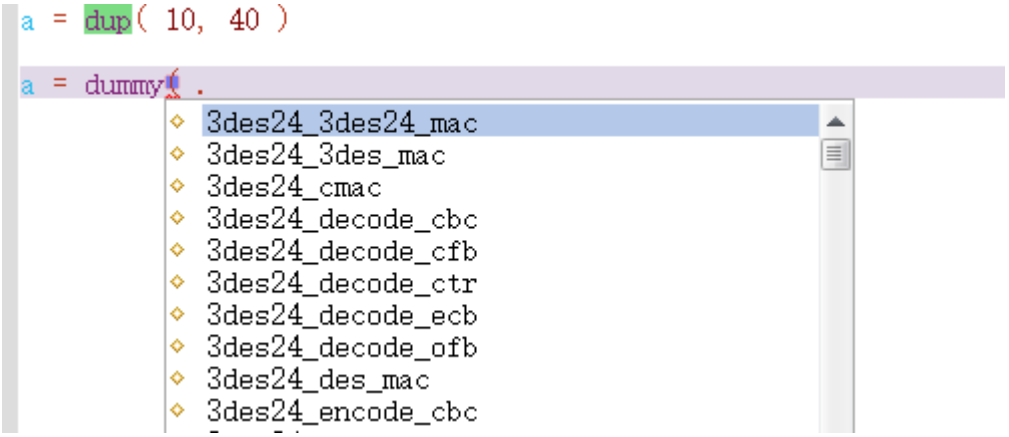
动态提示功能

内部函数提示

按键“.”提示系统内部函数，按空格或回车可以自动完成，自动之后会自动删除“.”

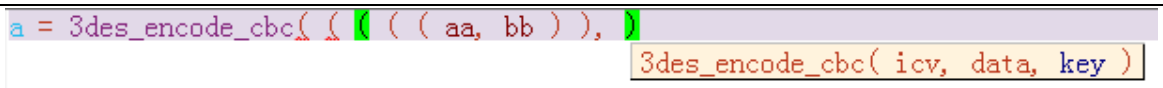


如果是在dummy函数中输入，则会提示第二套系数内部函数，按空格或回车可以自动完成，自动之后会自动删除“.”



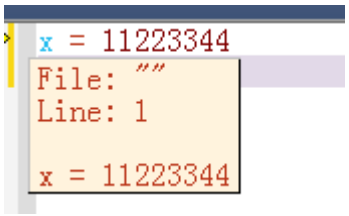
函数参数提示

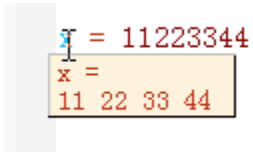
函数名称输入完整，按“(”，或“,”时，会进行参数提示，输入的括号会进行颜色匹配。光标位于系统函数语句内部时，’F1’键也会弹出参数提示。



1. 变量内容提示

变量在系统内存在时，鼠标指向变量，会进行内容提示，比如输入 x = 11223344 时，因为语句还未执行，鼠标指向 x 时，会提示源代码所在的文件与行数，运行过此语句之后，x作为变量在系统内存在，此时鼠标指向 x 时，会提示变量内容。

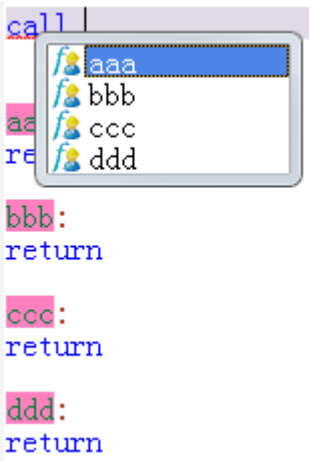




注：变量内容也可以通过变量窗口查看

脚本自定义语句提示

当在脚本中输入“call”或“goto”时，会弹出窗口。



窗口内容为加载时存在的函数，或上次运行时存在的函数，新编辑的函数不在列表中，要同步提示这些函数，可以选中工具栏的  ，此选项在编辑较大的脚本时会消耗一定的时间。

脚本自定义变量提示

当在脚本中输入“$”时，会弹出窗口。

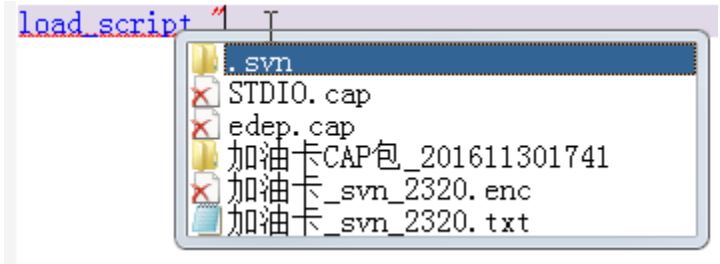
窗口内容为加载时存在的变量，或上次运行时存在的变量，新输入语句的变量不在列表中，要同步提示这些函数，可以选中工具栏的  ，此选项在编辑较大的脚本时会消耗一定的时间。

错误语法分析

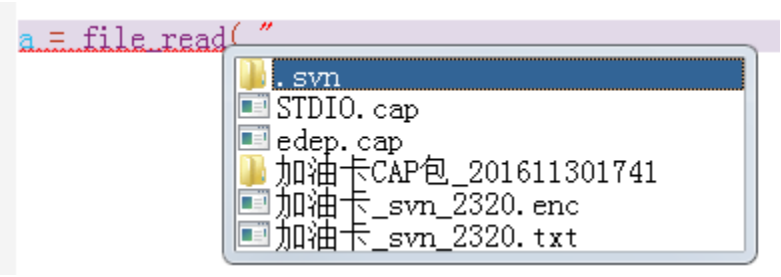
脚本加载或编辑时，具有简单的语法分析功能，错误的语句会在部分或整行下面划一条波浪线，鼠标指向错误的行，能够带有简单的提示。

load_script、读卡器增强提示

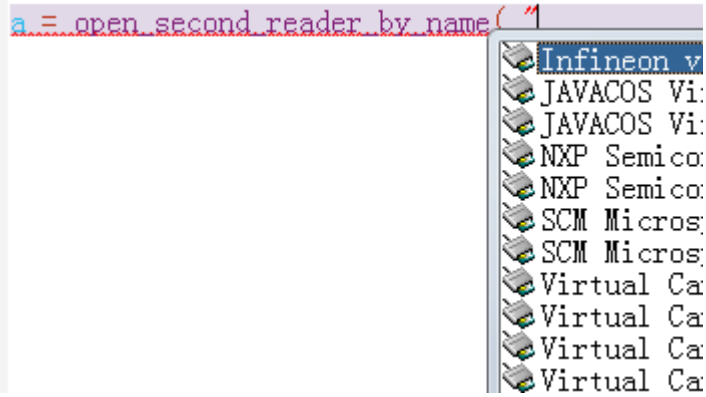
load_script 需要一个文件名参数，脚本可以弹出文件列表，列表中可以加入相对路径



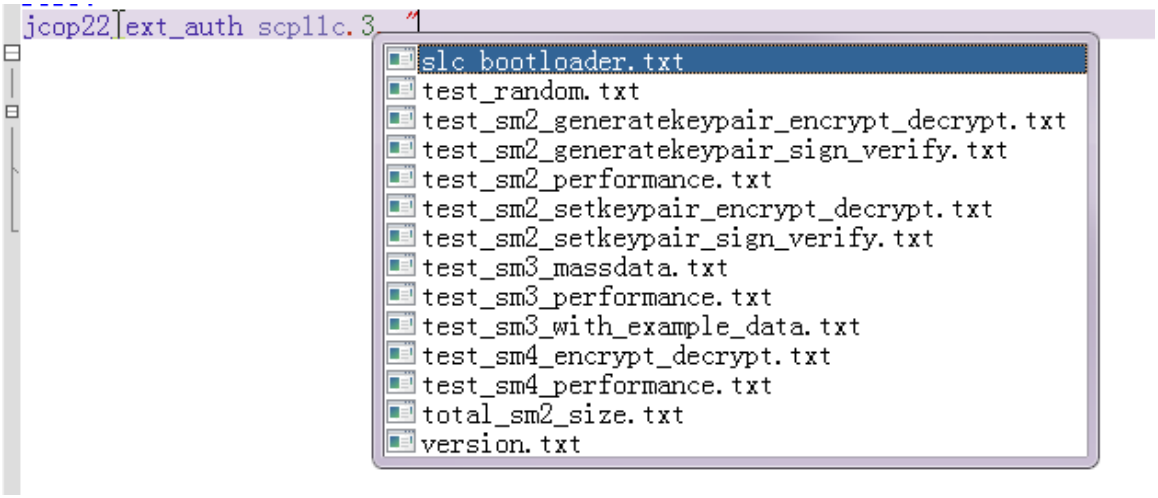
类似功能也增加到了 file_read、file_write 等函数内。



读卡器相关的函数



部分其他函数也加入了类似的功能



如果参数列表消失，可以按 F4 键呼出

注：光标放在“有效输入的最后”，才可以呼出菜单

注：必须输入一个左边的双引号，才可以呼出菜单

注：如果创建新脚本并且没有保存，则弹出的文件列表，以工具所在分区的根目录开始。

load_script "加油卡_svn_2 加油卡_svn_2320.enc 加油卡_svn_2320.txt	能够呼出时光标位置

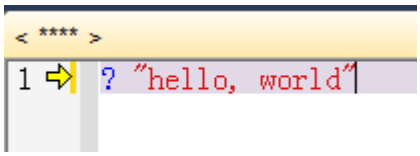
load_script "加油卡_svn_2	不能呼出时光标位置

第一个脚本

在屏幕上输出一个 hello, world。

在脚本编辑窗口输入

```
? "hello, world!"
```



按 F5 或 F10 或 F11

屏幕上会出现如下字符

```
//[]==================[]
//[]  hello, world!    []
//[]==================[]
```

并通过PC喇叭发出振铃声，提示您程序运行结束。

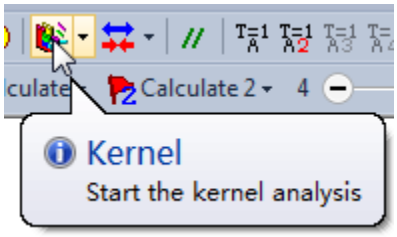
? 与 message具有相同效果，message函数可以接一个，也可以接多个参数，如下两条脚本的功能是一样的

```
message "hello, world!"
message "hello" "," " " "world" "!"
```

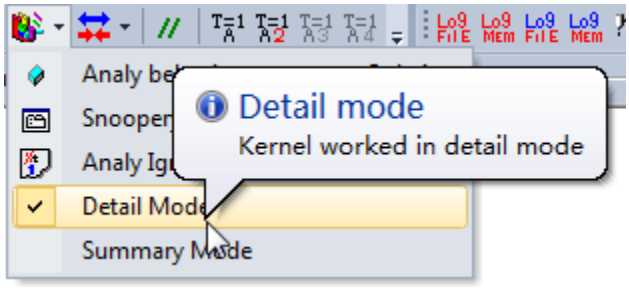
其具体的功能请见函数介绍章节。

输出设置

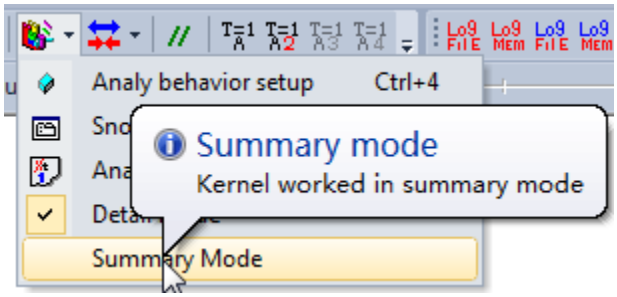
脚本工具只涉及功能测试，还可以通过一些简单的设置，来修改脚本工具的界面输出。

- 

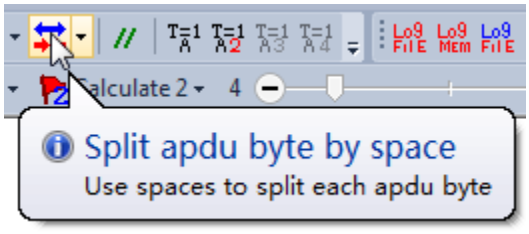
是否进行内核解析

- 

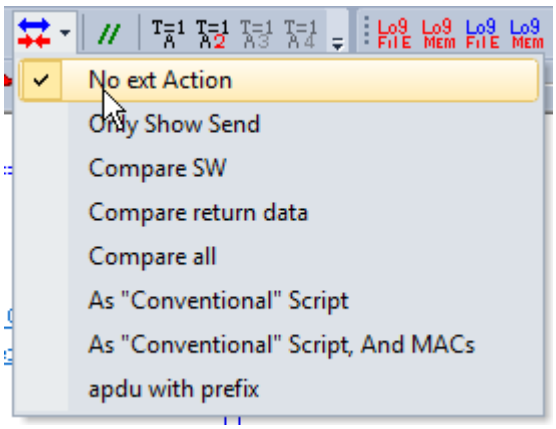
进行解析时，进行详细解析

- 

进行解析时，进行概要解析

- 

apdu输出时，是否使用空格分隔各字符

- 

apdu输出时，8种输出选择，默认且常用的是第一种。

脚本语法介绍

数据表示

未经说明，本文档中所有的数值都是16进制形式，比如E9表示值为0xe9的数值的16进制数。

完整 apdu 语句

apdu脚本一般都是每行一个独立语句，不支持独立语句分为多行，也不支持一行包含多个语句。

Apdu脚本语言，除非指定了进制，所以输入的数据都表示16进制，每两个连续的字符表示一个16进制数，如果某一行的字符串的统计结果是单数，那么最后一个字符将被丢弃。

如 00 01 02 03 04 05 06 07 08 09 aa表示0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0xaa

而00 01 02 03 04 05 06 07 08 09 a 则将丢弃最后单独的a，变成0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09，如果中间丢了一个字节，则后面的数据将发生半字节的错位。

静态脚本

静态脚本的组成

静态脚本就是由16进制ascii串组成的指令序列，例如 00 A4 00 00 02 3f 00，静态脚本将被直接转成16进制数据发送给智能卡，静态脚本并非只能接受纯16进制数据，也有多种表现形式，可以接用双引号引起来的字符串，比如以下几条脚本。

```
00a4000002 3f00
00a4000004 "3f00"
00a4000004 "汉字"
```

静态脚本中，双引号部分被视为字符串，’1’表示16进制的31。

前面脚本的执行结果

```
00A4000002 3F00
00A4000004 33663030
00A4000004 BABAD7D6
```

增强型静态脚本

增强型静态脚本是能够自动计算长度的脚本，现在有三种长度运算符“(”、“<”、“{”，括号使用要匹配。

被 () 包括起来的数据的长度将被计算成一个字节的长度

被 <> 包括起来的数据的长度将被计算成一个字节的长度，并加上 4 或 8 或 16(mac_length 的长度)

被 {} 包括起来的数据的长度将被计算成两个字节的长度

示例：

```
00a40000( 3f00 )
00a40000( "3f00" )
00a40000< "汉字" >
00000000 (<{}>)
00000000 ()
00000000 <>
00000000 {}
00000000 (())
```

结果：

```
00A4000002 3F00
00A4000004 33663030
00A4000008 BABAD7D6
0000000003 060000
0000000000
0000000004
0000000000 00
0000000001 00
```

增强型静态脚本可以直接执行，其中长度部分可以用“(”来代替一个字节的长度，其会自动算成“)”内部的hex字串长度

示例：

00a40000(d156000001010301)

“<”的长度

“<”在脚本的长度为 “<>”之间的数据的长度 + MAC长度，mac长度默认为4，可用

mac_length = 10进制mac长度

修改，有效范围是1-16，通常为4或8，个别情况可能会用到16。

注释

两个相连的斜杠//后面的字符被认为是注释，注释用于解释该程序是做什么的。//后面的的字符序列在编译时被忽略掉，它们可以在脚本中自由地使用，目的是为了使脚本更易于理解。注释可以出现在任何空格、制表符或换行符可以出现的地方。

Snooper Apdu脚本语言不支持块注释。

变量

变量无类型，统一为自带长度的16进制数据串，为兼容以前的脚本，以下各变量名保留，不能任意使用

类型	名称	作用	状态
返回状态变量	SW, SW1, SW2	存放返回状态	$sw $sw1 $sw2
随机数变量	rand	存放随机数	00 84 00 00 08

			if sw != 6D00 ? "pause" endif set rand rand = random(20)
加密密钥	cipherkey	存放加密密钥	cipherkey = dup(16, 40)
mac 长度	mac_length 或 maclength	用于“<”长度计算	mac_length = 8
response 长度	max_response_length	用于 get response 时一次最大取多少数据	max_response_length = 240
文件名	__FILE__	返回 ansi 编码的 16 进制脚本名称 如 c:\a.txt 返回 63 3A 5C 61 2E 74 78 74	x = $__file__
文件行数	__LINE__	返回 10 进制的脚本当前行	y = $__line__
预计算	int	得到一个 10 进制的结果，支持简单表达式与格式化	x = int(0x9999) x = int(0x9999, 304)
预计算	hex	得到一个 16 进制的结果	x = int(0x9999) y = hex($x)
预计算	float	得到一个 10 进制的浮点结果，此结果只能用于最终输出，不能参与计算	? float(0x9999 / 3.8)

标号与过程

一个非关键字的字符串后面跟随一个"："就构成了一个标号，一个标号与return组成构成了过程，过程必须被动调用。

使用标号的时候，可以用 goto 目的标号来跳转，而使用过程的时候，需要用 call 目的过程。

示例：

```
//标号示例
goto next1

? "start"
```

```
next1:
    ? "goto next1"

    //过程示例
    call next2

end

next2:
    ? "call next2"

    //返回
return
```

结果：

```
//--    //标号示例

//[]===============[]
//[]  goto next1    []
//[]===============[]
//--    //过程示例

//[]===============[]
//[]  call next2    []
//[]===============[]
//--    //返回
//Script File
//The script has finished running at line 12
```

注 1：

过程可以带调用参数，参数有两种声明方式

1） 在过程中使用 getpara 取得参数

2） 在过程声明中直接声明参数

两者也可以混用

例：

```
clear

reset

auto_response on
```

```
call aaa( 1111,,3333,,5555 )
call aaa2( 1111,,3333,,5555 )

end

aaa(a,b, c): 00 a4 04 00 00
    t = getpara
    e = getpara
    ? $a
    ? $b
    ? $c
    ? $t
    ? $e
return

aaa2: 00 a4 04 00 00
    a = getpara
    b = getpara
    c = getpara
    t = getpara
    e = getpara
    ? $a
    ? $b
    ? $c
    ? $t
    ? $e
return
```

~~注 2:~~ 已经删除这种做法

~~为理解方便，有的版本上 变量 = call 用户过程 这种写法时，也可以省略call关键字，但是这种写法要特别注意，用户自定义的过程，不能与内部函数重名，也不要与16进制立即数重复。且这种写法没有语法高亮，容易造成错觉。建议使用传统的方法~~

```
clear

x = 1122
x = call aaaa( 03, 04 )          // aaaa 本身是16进制数，但是在Call后面就没有问题
x = aaaa( 03, 04 )               // aaaa 本身是16进制数，且有同名的用户自定义过程，带有括号，认为是一个用户过程
x = aaaa 03                      // 这种写法不带括号，aaaa表示一个16进制立即数
```

```
x = 00 88 aaaa( 03 )          // 虽然aaaa是也是一个合法的过程，但是不在表达式第一位置，所以被当成16进制数
x = aaaa                      // aaaa当成16进制数
x = aaaa()                    // aaaa当成用户过程
x = the_good_user_proc( 11, 22 )  // 这个函数名没有歧义
x = call the_good_user_proc( 11, 22 ) // call 用户过程没有歧义

end

aaaa(input1, input2):
    value = add( $input1, $input2 )
return $value

the_good_user_proc(input1, input2):
    value = add( $input1, $input2 )
return $value
```

main 过程与 end

main 表明这是一个主过程，但是主过程的概念已经被淡化，仅仅 main 关键字还被保留，现在的 apdu 脚本中，从第一条开始都是主过程

end 是标明整个脚本结束，也用来划分主过程与子过程，一般编写脚本时，将子程序放在最后面，并且在主程序的最后用 end 表明整个程序结束。

脚本允许跨文件调用，0.0.4.8 以前的版本一般在主文件的最后，或子文件的最前面加 end。

0.0.4.8 版中如果主程序和子函数处于不同的脚本中，可以省略 end 语句。

数据串连接

在 apdu 脚本中，所有的表达式都要最终转换成 16 进制的运行时唯一的静态字符串，整个语句的最终表达结果就是所有静态字符串的顺序串联。

比如变量 oldkey 中的内容为 1122334455667788，执行 key = $oldkey 99ee，则 key 的内容就是

1122334455667788 99EE

示例：

```
oldkey = 1122334455667788

? $oldkey

key = $oldkey 99ee

? $key

oldkey = $oldkey $oldkey

? $oldkey
```

结果：

```
//oldkey = 1122334455667788

//[]====================[]
//[]  1122334455667788    []
//[]====================[]
//key = 1122334455667788 99EE

//[]=========================[]
//[]  1122334455667788 99EE    []
//[]=========================[]
//oldkey = 1122334455667788 1122334455667788

//[]====================================[]
//[]  1122334455667788 1122334455667788    []
//[]====================================[]
```

编写第一个实用脚本

以测试卡片取随机数的长度为例，典型的脚本是这样写的

```
clear                    // 清除屏幕的内容
```

```
reset                    // 复位卡片

auto_response on         // 自动取响应或重试

00 a4 04 00 ( 1122334455880101 )    // 选择应用
if sw != 9000            // 判断指令是否返回9000，不是9000则暂停
    pause
endif

00 84 00 00 04       // 尝试取4字节的随机数
if sw != 9000
    ? "此卡片  不支持  取4字节的随机数"     // 输出信息
    pause
endif
? "此卡片支持取4字节的随机数"            // 输出信息
set resp             // 将取得的随机数赋给变量 resp，resp可供后续使用


// 示例如何使用resp
random_first_byte = left( $resp, 1 )     // 使用left函数从resp中
                                         // 取得第一个字节，并赋
                                         // 给变量 random_first_byte

// 可以使用end来表示脚本结束，也可以不写
end
```

脚本执行日志

```
//reset--Card Status:The card has been reset and specific communication protocols have been
established.

//reset
3B8180018080

//Select File                            (defined by ISO/IEC 7816-4)
00A4040008 1122334455880101
9000

//Get Challenge                          (defined by ISO/IEC 7816-4)
0084000004
```

```
B4A8FA90
9000

//[]==============================[]
//[]  此卡片支持取4字节的随机数     []
//[]==============================[]
//resp = B4A8FA90
//--// 示例如何使用resp
//random_first_byte = B4
//--                                    // 取得第一个字节，并赋
//--                                    // 给变量 random_first_byte
//--// 可以使用end来表示脚本结束，也可以不写
//Script File  C:\Users\Desktop\0084.txt
//The script has finished running at line 26
```

控制流

if-else 语句

if 判断

if 语句是一个相当实用的判断，可以比较诸多的变量，使用方式也相当灵活，在使用 if 语句时，必须与一个 endif 配套使用，如果 if 与 endif 不配套，当 if 的条件为假时，会报错 endif 不匹配的错误，并停止脚本继续执行。

if 语句可以对复位信息，接收缓冲区，状态字，变量等进行判断。

if 语句可以多重嵌套。

if 语句不支持“与”,“或”等复杂逻辑。

判断条件可以是相等 = =，或是不等 ！=，**0.0.3.4 版本及以上，增加支持**>，>=，<，<=四种。

使用 if 时，变量要写在判断条件的前面，而比较值要写在判断表达式的后面，判断长度由真值决定。当判断中有不定的值时，可以用 * 代替，每个 * 表示半个 16 进制字节（此功能将来可能去掉）。

else 部分，它指定当 if 语句中的条件部分为假时所要采取的动作。其一般形式为：

```
if 表达式1
    语句1
else if 表达式2
    语句2
else if 表达式3
    语句3
else if 表达式4
    语句3
    ..................
else
    语句n
endif
```

在 if－else 的两个语句中有一个并且只有一个被执行。如果表达式的值为真，那么就执行语句 1，否则，执行语句 2。这两个语句均可以或者是单个语句或者语句序列。每个 if 分支仍然可以是一个 if 语句。

常用判断主要包括对变量内容进行判断，比如

```
if $pin == $oldpin
    //do something
endif
```

示例：

```
aa = 99

if $aa == 00
    ? "aa = 00"
else if $aa = 01
    ? "aa = 01"
else if $aa == 99
    ? "aa = 99"
else
    ? "aa 不知道是什么"
endif
```

结果：

```
//aa = 99
```

```
//[]============[]
//[]  aa = 99    []
//[]============[]
```

循环控制

do-loop 循环语句

APDU 脚本语言中的目前用三种循环语句，第一种是循环语句—do-loop，在循环体执行完后再测试终止条件，循环体至少要执行一次。do-loop 循环次数是 10 进制的。

do - loop 循环语句的语法是：

```
do 循环次数(10进制)
    语句
loop
```

在 do - loop 循环语句执行时，先要执行语句，然后再判断循环次数是否已经等于 0，如果值大于 0，那么就再次执行语句，如此等等。当循环次数的值变成 0 的的时候，就终止循环的执行。经验表明，使用 do - loop 语句的场合要比使用 for-next 少得多。然而， do - loop 循环语句有时还是很有价值的。

示例：

```
clear

do 0
    ? "果然循环了一次"
loop

do 1
    ? "我也循环了一次"
loop
```

结果：

```
//[]==================[]
//[]  果然循环了一次    []
```

```
//[]====================[]

//[]====================[]
//[]   我也循环了一次     []
//[]====================[]
```

while-loop 循环语句

while - loop 循环语句的语法是：

```
while 条件
    语句
loop
```

在 while - loop 循环语句执行时，先要执行判断语句，如果条件符合，则执行循环体， while - loop 循环语句比 do-loop 要实用一些。

示例：

```
clear

kk = 00

while $kk < 03
    kk = add( $kk, 01 )
    ? $kk
loop
```

结果：

```
//kk = 00
//kk = 01

//[]========[]
//[]   01    []
//[]========[]
//kk = 02

//[]========[]
//[]   02    []
//[]========[]
//kk = 03
```

```
//[]=======[]
//[]  03    []
//[]=======[]
```

do-loop 与 while-loop 循环可以有 20 层嵌套，while-loop 常用来做判断式循环，可以简化 goto 语句控制的循环。

for-next 循环语句

for循环语句:

```
for 表达式1 = 表达式2 to 表达式3 step 表达式4
    语句
next 表达式1
```

for循环语句的三个组成部分都是表达式。实际情况是，表达式1是循环变量，表达式2与表达式3是表达式1的作用区间，表达式1,2,3中任何一个都不可以省略，step与后面的表达式4可以省略。

当不出现step时，表示步长为1，当表达式2大于表达式3时，必须接step部分，且表达式4步长必须小于0。如果表达式4为0且表达式2与表达式3不等，for循环语句就是一个“无限”循环语句，这种语句要用其他手段（如exit for语句或return语句或end语句）才能终止执行。

for语句的表达2，3，4要求的都是10进制数，表达式1则是a-g,i-k,l-n中的一个。

示例：

```
clear

reset

for a = 0 to 5
    message $a
next a

number = ""

for i = 0 to 9
    number = $number $i
next i
? $number
```

```
for i = 9 to 0 step -1
    number = $number $i
next i
? $number

for i = 1 to 1 step 0
    ? "不是死循环"
next i

for i = 1 to 300 step 0
    ? "死循环"
next i
```

结果：

```
//Protocol T1
//reset--Card Status:The card has been reset and specific communication protocols have been established.

//reset
3B80800101

//[]========[]
//[]  00    []
//[]========[]

//[]========[]
//[]  01    []
//[]========[]

//[]========[]
//[]  02    []
//[]========[]

//[]========[]
//[]  03    []
//[]========[]

//[]========[]
//[]  04    []
```

```
//[]========[]

//[]========[]
//[]  05     []
//[]========[]
//number =
//number = 0000
//number = 00000001
//number = 000000010002
//number = 0000000100020003
//number = 00000001000200030004
//number = 000000010002000300040005
//number = 0000000100020003000400050006
//number = 00000001000200030004000500060007
//number = 000000010002000300040005000600070008
//number = 0000000100020003000400050006000700080009

//[]=============================================[]
//[]  000000010002000300040005000600070008000900  []
//[]=============================================[]
//number = 00000001000200030004000500060007000800090009
//number = 000000010002000300040005000600070008000900090008
//number = 00000001000200030004000500060007000800090009000800007
//number = 000000010002000300040005000600070008000900090008000070006
//number = 0000000100020003000400050006000700080009000900080007000600005
//number = 00000001000200030004000500060007000800090009000800070006000050004
//number = 000000010002000300040005000600070008000900090008000700060005000400003
//number = 0000000100020003000400050006000700080009000900080007000600050004000300002
//number = 00000001000200030004000500060007000800090009000800070006000500040003000200001
//number = 000000010002000300040005000600070008000900090008000700060005000400030002000100000

//[]=============================================================================================[]
//[]  000000010002000300040005000600070008000900090008000700060005000400030002000100000    []
//[]=============================================================================================[]

//[]===============[]
//[]  不是死循环    []
//[]===============[]

//[]===========[]
//[]  死循环    []
//[]===========[]
```

```
//[]============[]
//[]  死循环     []
//[]============[]
.............................
//[]============[]
//[]  死循环     []
//[]============[]
```

只做固定次数的循环，不包含初始化或重新初始化部分，使用do - loop循环语句最为自然。如果要做简单地初始化与增量处理，最好还是使用for语句，因为它可以使循环控制的语句更密切，而且它把控制循环的信息放在循环语句的顶部，易于程序理解。这在如下语句中表现得更为明显：

```
for a = 1 to 20
    00 b2 $a 04 b0
next a
```

这是apdu脚本语言在处理一个文件的前20个元素时的一种习惯性用法，类似于basic语言的for循环语句。但是，这种类比不够恰当，因为apdu脚本语言循环语句的当前值和终值在循环语句体内不可以改变，只能接受10进制形式的数值。在循环因某种原因终止时（用exit或循环自动结束）标变量a的值仍然保留，用goto语句也可以退出for - next循环，但是这样做会导致由于a的变量值仍然保留，所以再次使用a的时候会导致不可预料的错误，所以禁止使用goto语句退出for-next循环与do-loop循环。

exit for 语句

在for-next循环语句执行过程中，除了通过测试在循环顶部与底部正常退出外，有时从循环中直接退出会显得更加方便一些。exit for语句可以从for-next循环中退出来，exit for语句每次退出一层for-next循环。循环完毕或执行exit 会使循环值越过终止值。

示例：

```
clear

flag = 00
for i = 0 to 9
    if $i == 0004
        flag = 01
        exit for
    endif
```

```
        ? $i
    next i
    ? $i
    if $flag == 00
        ? "循环完了"
    else
        ? "没循环完，成功"
    endif
```

结果：

```
//flag = 00

//[]=========[]
//[]  0000    []
//[]=========[]

//[]=========[]
//[]  0001    []
//[]=========[]

//[]=========[]
//[]  0002    []
//[]=========[]

//[]=========[]
//[]  0003    []
//[]=========[]
//flag = 01

//[]=========[]
//[]  000A    []
//[]=========[]

//[]==================[]
//[]  没循环完，成功    []
//[]==================[]
```

hfor-hnext 循环与 exit hfor 组合

因为for-next的参数都是10进制的，带来了很大的不方便，所以又增加了hfor-hnext循环，配套的组

合为exit hfor语句。hfor-hnext循环的参数，都必须是16进制格式，诸如0,5这样的数值，要写成00,05。

for-next 与 hfor-hnext 最好不要嵌套使用

goto 语句与标号

APDU脚本设计语言提供了定义标号，可以用goto语句跳转到标号，以及可以毫无节制使用的goto语句的机制。

合理使用goto语句能够使代码逻辑更加清晰，在有些情况下使用goto语句可能比较合适。最常见的用法是在某些情况下跳过以前的代码，直接到某一行去执行。下面是使用goto语句的一个例子：

示例：

```
clear

reset

auto_response on
prompt on

00 a4 00 00 02 3f 00
if sw != 9000
    goto createmf
endif

// 删除mf

createmf:

    // 建立mf
    ...
    ...

return
```

结果：

无

如果错误处理比较重要并且在好几个地方都会出现错误，那么使用这种组织就比较灵活方便。标号的形式与变量名字相同，其后要跟一个冒号。标号可以用在任何语句的前面，但要与相应的goto语句位于同一文件中。标号的作用域是整个脚本。

示例：

```
if 操作1 失败
    goto 生成汇报
endif
if 操作2 失败
    goto 生成汇报
endif
if 操作3 失败
    goto 生成汇报
endif
if 操作4 失败
    goto 生成汇报
endif
if 操作5 失败
    goto 生成汇报
endif
if 操作6 失败
    goto 生成汇报
endif

生成汇报:
    ...
```

结果：

无

函数与程序结构

函数的基本知识

函数可以把大的计算任务分解成若干个较小的任务，脚本设计人员可以基于函数进一步构造脚本，而不需要重新编写一些代码，一个设计得当的函数可以把程序中不需要了解的具体操作细节隐藏起来，从而使整个程序结构更加清晰，并降低修改脚本的难度。

apdu 脚本语言在设计中考虑了函数的高效性与易用性这两个因素，脚本通常由多个较小的函数组成，而不是由较大的函数组成。一个完整的脚本可以保存在一个或多个源文件中。

一个合法的标号 +return，就构成了一个合法的函数，在 apdu 脚本中，函数可以重名，但是重名函数只有第一个会被调用。

示例：

```
clear
reset
set resp
call analy_atr
end

analy_atr:
    ts = mid( $resp, 0, 1 )
    if $ts == 3b
        ? "正向"
    else if $ts == 3f
        ? "反向"
    else
        ? "未知"
    endif
return
```

函数可以带参数，参数由call指令来传递，参数使用‘(’ 和 ‘)’来包括，多个参数使用 ‘,’进行分隔，在函数内部使用getpara关键字进行取值。

示例：

```
clear

reset

set resp

call analy_atr( aabbccdd, ddccbbaa )

end

analy_atr:
```

```
    tmp1 = pop
    tmp2 = pop

    ? $tmp1
    ? $tmp2

return
```

结果：

```
//reset--Card Status:The card has been reset and specific communication protocols have been
established.

//reset
3B888001443131435232 2E3070
//resp = 3B888001443131435232 2E3070
//tmp1 = DDCCBBAA
//tmp2 = AABBCCDD

//[]==============[]
//[]  DDCCBBAA    []
//[]  AABBCCDD    []
//[]==============[]
//The script has finished running at line 9
```

```
clear

reset
set resp

call analy_atr( aabbccdd, ddccbbaa )

end

analy_atr:
    tmp1 = getpara
    tmp2 = getpara
    ? $tmp1
    ? $tmp2
return
```

结果：

```
//Protocol T1
//reset--Card Status:The card has been reset and specific communication protocols have been established.

//reset
3B8B800186AA000122011311223344 53
//resp = 3B8B800186AA000122011311223344 53
//tmp1 = AABBCCDD
//tmp2 = DDCCBBAA

//[]==============[]
//[]  AABBCCDD    []
//[]  DDCCBBAA    []
//[]==============[]
//Script File
//The script has finished running at line 8
```

函数的返回值

0.0.5.3 之前的用户自定义函数不支持返回值，但是可以借用全局变量或 fifo 功能来实现返回值。

全局变量示例：

```
clear

reset

set resp

call analy_atr

? $t

end

analy_atr:

    t =  11223344
```

```
    return
```

结果：

```
//reset
3B8B800186AA00012201131122334453
//resp = 3B8B800186AA00012201131122334453
//t = 11223344

//[]==============[]
//[]  11223344    []
//[]==============[]
//Script File
//The script has finished running at line 11
```

FIFO 示例：

```
    clear

    reset

    set resp

    call analy_atr

    t = read_fifo

    ? $t

    end

    analy_atr:

        clear_fifo

        write_fifo 11223344

    return
```

结果：

```
//reset
3B888001443131435232 2E3070
```

```
//resp = 3B888001443131435232 2E3070
//t = 11223344

//[]==========[]
//[]  11223344   []
//[]==========[]
//The script has finished running at line 13
```

0.0.5.3 版之后，函数支持返回值，可以用 push 来压入多个返回值，也可以在 return 语句直接带返回值。

递归

一个过程或函数在其定义或说明中有直接或间接调用自身的一种方法，它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。递归的能力在于用有限的语句来定义对象的无限集合。一般来说，递归需要有边界条件、递归前进段和递归返回段。当边界条件不满足时，递归前进；当边界条件满足时，递归返回。

apdu 脚本语言也支持递归，但是因为 apdu 脚本语言中没有局部变量，所以在递归函数中要注意影响。而且由于 apdu 脚本语言的限制，递归不能超过 19 层。

示例：

```
aa = 05
cc = $aa

call fact

dd = hex2int( $cc )
? "5! = " $dd

end

fact:
    bb = sub( $aa, 01 )
    if $bb == 00
        return
    else
```

```
        cc = mul( $cc, $bb )
        aa = sub( $aa, 01 )
        call fact
    endif

return
```

结果：

```
//aa = 05
//cc = 05
//bb = 04
//cc = 14
//aa = 04
//bb = 03
//cc = 3C
//aa = 03
//bb = 02
//cc = 78
//aa = 02
//bb = 01
//cc = 78
//aa = 01
//bb = 00
//dd = 0120

//[]==============[]
//[]  5! = 0120    []
//[]==============[]
//The script has finished running at line 9
```

关键字

关键字分为四部分。

1. 前缀关键字

指令型关键字用来实现流程控制、特殊操作、信息提示、用户交互等功能。

2. 功能性关键字

包括函数与宏两部分，其中宏已经基本上被淘汰，建议在使用宏的地方全部使用相同名称的函数来代替。

功能性关键字将在函数部分进行详细说明

3. 辅助型关键字

用于实现语法通顺

4. 内部变量

用于实现语法通顺

辅助型关键字+以及内部变量

```
appletaid
cipherkey cardmanager
data default_selected
ecc_curve_id ecc_keypair
instanceaid installdata
maclength mac_length
rand result rsa_keypair
packageaid  privileges
sw sw1 sw2
```

前缀关键字

指令型关键字加在脚本的前面，用以标明脚本应该以何种方式进行运算或控制。

```
apdu2 auto_response
```

beep

call clear clear_fifo cold

do

eject end else endif error exit

for

hfor hnext

if insert install install_2

jcop22_ext_auth jcop22_ext_auth_kmc

jcop22_ext_auth_nonblock jcop22_ext_auth_kmc_nonblock

jcop22_ext_auth_2 jcop22_ext_auth_kmc_2

jcop22_ext_auth_nonblock_2 jcop22_ext_auth_kmc_nonblock_2

goto generate getpara

main message mac_auto_mac mac_auto_mac_2

next

loop load_script

on off

prompt pause pop

reset return reset2 read_fifo

set step sleep showapdu split select shutdown sm4_mac sm4_mac_2

upload upload_2

timer_begin timer_end to

with write_fifo

clear

清除屏幕

reset

复位当前设备，加入 cold 关键字，则表示冷复位。

reset2

复位第二设备，加入 cold 关键字，则表示冷复位。

第二设备复位成功，sw 为 9000，否则 sw 为 ffff

apdu2

向第二设备发送数据，一切数据都占用第一设备所使用的缓冲

auto_response

自动取 61XX 或自动重试 6CXX

auto_response 之后要使用 on/off 参数

使用 on 参数，表明自动取响应，使用 off 参数，则不自动取响应。

示例	clear reset auto_response on 00 a4 00 00 02 3f 00 if sw != 9000 pause endif

结果	//reset
	3B 7F 95 00 00 00 66 46 53 00 10 01 21 71 DF 00 00 81 90 00
	
	//SELECT
	00A4000002 3F00
	611A
	
	//GET RESPONSE
	00C000001A
	6F18840E315041592E5359532E4444463031A5068801019F0C00
	9000

prompt

prompt 之后要使用 on/off 参数

使用 on 参数，变量赋值时显示内容

showapdu

showapdu 之后要使用 on/off 参数

使用 on 参数，与卡片通讯时，通信内容会显示。

call

调用子程序

示例	clear call sub1 ? "success" end sub1: ? "in sub1" return
结果	//[]============[] //[] in sub1 [] //[]============[] //[]============[] //[] success [] //[]============[] //The script has finished running at line 7

示例	clear call sub1(1111, 2222) ? "success" end sub1: para2 = pop

```
    para1 = pop

    ? "in sub1"
    ? "para1 = " $para1
    ? "para2 = " $para2

return
```

结果

```
//para2 = 2222
//para1 = 1111

//[]=================[]
//[]  in sub1          []
//[]  para1 = 1111     []
//[]  para2 = 2222     []
//[]=================[]

//[]=============[]
//[]  success     []
//[]=============[]
//Script File
//The script has finished running at line 7
```

return

从函数中返回调用处的下一行，0.0.5.3 版之后，return 可以带返回值，返回值可以是静态数据，也可以是一个函数的返回值。

```
return 11223344
return $k
return des_des_mac( aa, bb, cc )
```

local

在子函数中，声明一个变量是局部变量，这样可以在一定程度了解决变量重名的问题。

```
local 变量名
```

示例

```
a   = 9999
b   = 8888

sum = call demo( $a, $b )
```

```
end

demo:
    local num1
    local num2

    num1    = getpara
    num2    = getpara

    local tmp
    tmp     = 33
    sum     = call a( $num1, $num2 )
    ? $tmp
    tmp     = 44
    diff    = call b( $sum, $num2 )
    ? $tmp

    push 999999     // like return 999999

return

a:
    local num1
    local num2

    num1    = getpara
    num2    = getpara

    local tmp

    tmp     = add( $num1, $num2 )


return $tmp


b:
    local num1
    local num2

    num1    = getpara
```

```
    num2    = getpara

    local tmp

    tmp     = sub( $num1, $num2 )

return $tmp
```

load_script

load_script "文件名"

将“文件名”加载到新的脚本窗口，被加载的一般是一段子程序。

示例1	load_script "0031_load_sub.txt" call sub1 end
子程序	sub1: ? "在子程序里面" return

结果	//[]==========[]
	//[] 在子程序里面 []
	//[]==========[]
	//Script File
	//The script has finished running at line 5

loadprofile 和 run_script

loadprofile "文件名"

或

run_script "文件名"

将“文件名”加载到新的脚本环境，并被执行，被加载的是一段个完整的脚本。

示例1	loadprofile "other.txt" end
子脚本	? "在子脚本里面"

结果	//[]==========[]
	//[] 在子脚本里面 []
	//[]==========[]
	//Script File
	//The script has finished running at line 3

pause

暂停脚本执行

sleep

sleep 10 进制毫秒数

示例	

	clear
	
	old = timer
	do 5
	  sleep 500
	  tt = timer
	  ee = hex2int($tt)
	  ? "系统自开机以来走过了 " $ee " 毫秒"
	  xx = sub($tt, $old)
	  xx = hex2int($xx)
	  ? "本次sleep的精度是 " $xx " 毫秒"
	  old = $tt
	loop

结果	//old = 0093E0FE
	//tt = 0093E301
	//ee = 09691905
	
	//[]==========[]
	//[] 系统自开机以来走过了 09691905 毫秒 []
	//[]==========[]
	//xx = 00000203
	//xx = 0515
	
	//[]==========[]
	//[] 本次sleep的精度是 0515 毫秒 []
	//[]==========[]
	//old = 0093E301
	//tt = 0093E504
	//ee = 09692420
	
	//[]==========[]
	//[] 系统自开机以来走过了 09692420 毫秒 []
	//[]==========[]
	//xx = 00000203
	//xx = 0515

```
//[]============================================[]
//[]  本次sleep的精度是         0515  毫秒     []
//[]============================================[]
//old = 0093E504
//tt = 0093E706
//ee = 09692934

//[]============================================[]
//[]  系统自开机以来走过了      09692934  毫秒     []
//[]============================================[]
//xx = 00000202
//xx = 0514

//[]============================================[]
//[]  本次sleep的精度是         0514  毫秒     []
//[]============================================[]
//old = 0093E706
//tt = 0093E909
//ee = 09693449

//[]============================================[]
//[]  系统自开机以来走过了      09693449  毫秒     []
//[]============================================[]
//xx = 00000203
//xx = 0515

//[]============================================[]
//[]  本次sleep的精度是         0515  毫秒     []
//[]============================================[]
//old = 0093E909
//tt = 0093EB1C
//ee = 09693980

//[]============================================[]
//[]  系统自开机以来走过了      09693980  毫秒     []
//[]============================================[]
//xx = 00000213
//xx = 0531

//[]============================================[]
//[]  本次sleep的精度是         0531  毫秒     []
//[]============================================[]
//old = 0093EB1C
```


end

结束脚本执行

insert

等待用户插卡，也可以使用 insert card

insert与eject可以在批量发卡时实现换卡功能。(目前已被界面脚本取代)

示例	clear count = 00 do 3 insert 00 84 00 00 08 if sw != 9000 pause endif count = add($count, 01) eject loop ? "您执行完了 " $count "次操作"

结果	//count = 00
	//--Please Insert Card
	
	//Select File (defined by ISO/IEC 7816-4)
	00A4040000
	9000
	//count = 01
	//--Please pull out the card
	//--Please Insert Card
	
	//Select File (defined by ISO/IEC 7816-4)
	00A4040000
	9000
	//count = 02
	//--Please pull out the card
	//--Please Insert Card

```
// Select File                (defined by ISO/IEC 7816-4)
00A4040000
9000
//count = 03
//--Please pull out the card

//[]========================[]
//[]  您执行完了 03次操作     []
//[]========================[]
```

eject

提示并等待拔出卡片，见 insert 部分。(目前已被界面脚本取代)

? 与 message

?普遍代替在脚本中使用 “?” 与 “message” 关键字等效

? “变量内容 = “ $变量内容

在if语句中使用 ? 可以提示参与比较的两个变量是否一致

```
if $aa != 64 30 90 87
    ?
    pause
endif
```

在屏幕上显示内容，分为带参数与不带参数两种情况。带参数的将在屏幕上显示输出，不带参数的可以进行显示数据差异。

在 if 与 endif 之间的一个不带任何参数的 message 或 “?”，用来比较刚刚参与比较的两个数据的差异，所以不带参数的 message 或 “?”应该使用在比较语句之后。

示例

```
clear

reset
if sw != 9000
    pause
endif
set resp

? "hello, the atr is " $resp

if $resp != 3b 00
    ?
    ? "这个复位信息不是3b 00"
    pause
endif
```

	? "结束"
结果	Protocol T1 Reset--Card Status:The card has been reset and specific communication protocols have been established. reset 3B8180018080 resp = 3B8180018080 []==================================[] [] hello, the atr is 3B8180018080 [] []==================================[] Compare 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ---- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0000 3B 81 80 01 80 80 ---- 3B 00 -- -- -- -- []==========================[] [] 这个复位信息不是3b 00 [] []==========================[] Pause at Line 14 The script has paused, press 'F5' to continue

示例	clear
	
	reset
	if sw != 9000
	pause
	endif
	set resp
	
	? "hello, the atr is " $resp
	
	if $resp == 3b 00
	? "这个复位信息是3b 00"
	else if $resp != 3b fd
	?
	? "这个复位信息不是3b fd"
	pause
	else
	pause
	endif
	
	? "结束"

结果	//Reset--Card Status:The card has been reset and specific communication protocols have been established.
	
	//reset
	3B8180018080
	//resp = 3B8180018080
	
	//[]==================================[]
	//[] hello, the atr is 3B8180018080 []
	//[]==================================[]
	//---------- Compare ----------
	//------ 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ---- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
	//0000 3B 81 80 01 80 80 ---- 3B FD -- -- -- --
	//
	//
	
	//[]==========================[]
	//[] 这个复位信息不是3b fd []
	//[]==========================[]
	//--------
	//------ Pause at Line 16
	//The script has paused, press 'F5' to continue

beep

提示用户某事件已经发生。

示例	clear beep
结果	如果 pc 喇叭可用，snooper 全局音效打开，可以听到两声蜂鸣。

clear_fifo

清除 fifo 内的数据，保证后续 fifo 操作都是最新的数据。

write_fifo

将数据写入 fifo 中，一般与 clear_fifo 和 read_fifo 使用。

write_fifo 是关键字

read_fifo

read_fifo 是一个函数

```
index = 00

for i = 0 to 20
    write_fifo $index
    index = add( $index, 04 )
next i
```

```
for i = 0 to 20
    index = read_fifo
    ? $index
next i
```

do

与 loop 配套使用，用于执行固定次数的循环，循环次数为 0 时，也会执行循环。

示例	clear do 3 ? "hello" loop
结果	//[]==========[] //[] hello [] //[]==========[] //[]==========[] //[] hello [] //[]==========[] //[]==========[] //[] hello [] //[]==========[]

loop

与 do 配套使用

if

判断比较，if 语句必须和 endif 配套，如果不配套，工具会自动指出哪个 if 不配套。

示例	aa = 99
	
	if $aa == 00
	    ? "aa = 00"
	else if $aa = 01
	    ? "aa = 01"
	else if $aa == 99
	    ? "aa = 99"
	else
	    ? "aa 不知道是什么"

	endif
结果	//aa = 99 //[]============[] //[] aa = 99 [] //[]============[]

else

在 if - else if - else - endif 语句中使用，else 部分存在时，没有匹配的事件时，将执行 else 部分动作。

示例	if $aa == 01
	  ? "01"
	else if $aa == 02
	  ? "02"
	endif
	
	? "else存在，只有if"
	aa = 00
	
	if $aa == 01
	  ? "01"
	else
	  ? "非01"
	endif
	
	? "else存在，有if和else if"
	aa = 00
	
	if $aa == 01
	  ? "01"
	else if $aa == 02
	  ? "02"
	else if $aa == 03
	  ? "03"
	else if $aa == 04
	  ? "04"
	else
	  ? "非01, 02, 03, 04"
	endif

结果	//[]==============[] //[] else不存在 [] //[]==============[]

```
//aa = 00

//[]====================[]
//[]  else存在，只有if    []
//[]====================[]
//aa = 00

//[]==========[]
//[]  非01    []
//[]==========[]

//[]============================[]
//[]  else存在，有if和else if    []
//[]============================[]
//aa = 00

//[]====================[]
//[]  非01, 02, 03, 04    []
//[]====================[]
```

endif

if 语句必须和 endif 配套

,如果不配套,工具会自动指出哪个 if 不配套。

见 if 部分。

for

与 next 配套使用，执行有条件循环，并且可以使用 exit for 从 for - next 循环中退出。

示例

```
clear

reset

for a = 0 to 5
    message $a
next a

set number = ""

for i = 0 to 9
```

```
    set number = $number $i
next i
message $number

for i = 9 to 0 step -1
    set number = $number $i
next i
message $number

for i = 1 to 1 step 0
    message "不是死循环"
next i

for i = 1 to 300 step 0
    message "死循环"
next i
```

结果

```
//[]========[]
//[]  00    []
//[]========[]

//[]========[]
//[]  01    []
//[]========[]

//[]========[]
//[]  02    []
//[]========[]

//[]========[]
//[]  03    []
//[]========[]

//[]========[]
//[]  04    []
//[]========[]

//[]========[]
//[]  05    []
//[]========[]
//number =
//number = 0000
//number = 00000001
```

```
//number = 000000010002
//number = 0000000100020003
//number = 00000001000200030004
//number = 0000000100020003000040005
//number = 000000010002000300040005006
//number = 00000001000200030004000500060007
//number = 000000010002000300040005000600070008
//number = 0000000100020003000400050006000700080009

//[]=============================================[]
//[]  000000010002000300040005000600070008000900090008    []
//[]=============================================[]
//number = 00000001000200030004000500060007000800090009000800070006
//number = 000000010002000300040005000600070008000900090008000700060005
//number = 00000001000200030004000500060007000800090009000800070006000500040003
//number = 0000000100020003000400050006000700080009000900080007000600050004000300020001
//number =
0000000100020003000400050006000700080009000900080007000600050004000300020001000000

//[]=============================================================================================
[]
//[]  000000010002000300040005000600070008000900090008000700060005000400030002000100000
[]
//[]=============================================================================================
[]

//[]===============[]
//[]  不是死循环    []
//[]===============[]

//[]============[]
//[]  死循环    []
//[]============[]

//[]============[]
//[]  死循环    []
//[]============[]
.....
```

next

与 for 配套使用

exit

用于退出 for 循环或 hfor 循环

示例	for i = 0 to 9
	    if $i == 0004
	        exit for
	    endif
	    ? $i
	next i
	
	if $i == 0009
	    ? "循环完了"
	else
	    ? "没循环完，成功"
	endif

结果	//[]==========[] //[] 0000 [] //[]==========[] //[]==========[] //[] 0001 [] //[]==========[] //[]==========[] //[] 0002 [] //[]==========[] //[]==========[] //[] 0003 [] //[]==========[] //[]====================[] //[] 没循环完，成功 [] //[]====================[]

hfor，hnext，exit hfor

与 for，next，exit for 相同,唯一不同的是 **hfor，hnext 所有的数值全是 16 进制**。

goto

跳转到标号

示例	timer_begin
	//生成密钥对
	00 46 00 00 02 04 00
	timer_end
	if sw != 9000
	  message "当前行" __line__
	  goto ERR
	endif
	
	message "++++++ac verify ok ++++++"
	end
	
	ERR:
	  message "----------ERROR--------"
	end

结果	//[]===== Script timming =====[]
	//[] Script Timming start []
	//[]=========================[]
	//--//生成密钥对
	
	//not defined instruction
	0046000002 0400
	6D00
	
	//[]= Script timed duration ==[]
	//[] Time 0 ms []
	//[]=========================[]
	
	//[]==============[]
	//[] 当前行 6 []
	//[]==============[]
	
	//[]============================[]
	//[] ----------ERROR-------- []
	//[]============================[]
	//Script File
	//The script has finished running at line 15

on error goto 错误处理标号

on error pause

当有很多错误时，可以先用此处理，此函数目前只支持通讯错误，目前主要用来测试掉电，一旦拔出卡或key就执行错误。

on error pause，取消错误处理陷阱

示例	clear
	
	reset
	
	auto_response on
	
	on error goto errhandle
	
	00 a4 04 00 00
	if sw != 9000
	  pause
	endif
	
	? "请拔出卡片"
	
	sleep 5000
	
	00 a4 04 00 00
	if sw != 9000
	  pause
	endif
	
	? "没有拔出卡片"
	
	end
	
	errhandle:
	  ? "卡片通讯异常了"
	
	end

结果1	没有拔出卡片的情况 //reset--Card Status:The card has been reset and specific communication protocols have been established.

	//reset
	3B8B800186AA00012201131122334453
	
	//Select File (defined by ISO/IEC 7816-4)
	00A4040000
	6F5C8408A000000003000000A504734A06072A864886FC6B01600C060A2A864886FC6B020201016309
	06072A864886FC6B03640B06092A864886FC6B040215650B06092B851086486402010366OC060A2B06
	0104012A026E01029F6501FF
	9000
	
	//[]==========[]
	//[] 请拔出卡片 []
	//[]==========[]
	
	//Select File (defined by ISO/IEC 7816-4)
	00A4040000
	6F5C8408A000000003000000A504734A06072A864886FC6B01600C060A2A864886FC6B020201016309
	06072A864886FC6B03640B06092A864886FC6B040215650B06092B851086486402010366OC060A2B06
	0104012A026E01029F6501FF
	9000
	
	//[]==========[]
	//[] 没有拔出卡片 []
	//[]==========[]
	//Script File
	//The script has finished running at line 25

结果2	拔出卡片的情况
	
	//reset--Card Status:The card has been reset and specific communication protocols have been established.
	
	//reset
	3B8B800186AA00012201131122334453
	
	//Select File (defined by ISO/IEC 7816-4)
	00A4040000
	6F5C8408A000000003000000A504734A06072A864886FC6B01600C060A2A864886FC6B020201016309
	06072A864886FC6B03640B06092A864886FC6B040215650B06092B851086486402010366OC060A2B06
	0104012A026E01029F6501FF
	9000
	
	//[]==========[]

```
//[]  请拔出卡片     []
//[]================[]

//[]======================== Smart Card Error
====================================[]
//[]  Error code 0x80100069 The smart card has been removed, so                    []
//[]  that further communication is not possible
[]
//[]=================================================================================
==[]

//--io error
//--00A4040000
//-------
//--------Communication error, from line 18 goto error handling

//[]====================[]
//[]  卡片通讯异常了     []
//[]====================[]
//Script File
//The script has finished running at line 30
```

On Error时会暂存当前环境中部分关键数据，所以On error的位置比较关键，下面这个示例，因为on error位置在外围，实际运行时会导致错误

错误示例

```
clear

reset

on error goto again

do 100

    insert
    // send apdu..........


again:
    eject

loop
```

推荐写法

```
clear
```

```
reset

do 100

    on error goto again      // 在这里设置error陷阱


    insert
    // send apdu.........

again:
    eject

loop
```

timer_begin
timer_end

两个计时函数

示例	timer_begin sleep 2000 timer_end
结果	//[]==========[x] //[] 计时开始 //[]==========[] //[]==========[x] //[] Time Elapsed 2031 ms //[]==========[]

des_des_mac 、des_3des_mac、full_3des_mac
des_des_mac_2 、des_3des_mac_2
sm4_mac、sm4_mac_2

```
des_des_mac 80ee0000< 11223344 >
用 cipherkey 对数据进行加密，并将结果填到明文后面
首先将数据80对齐，然后用icv分别对每组数据进行异或，再对异或结果进行加密，遵循前面des最后des/3des的原则。
注：此函数自动进行80对齐
注：要注意mac_length的长度。
```

自动计算指令后的 3des mac

示	clear

例

```
reset

auto_response on

// data[ 1 ]是传输密钥
set data[ 1 ] = 00112233445566778899aabbccddeeff
set cipherkey = $data[ 1 ]

set maclength = 4

00 84 00 00 08
set rand

des_3des_mac 84 e0 00 00 <62 16 82023800 83023F00 85020000 8608 0000240000000000>
if sw != 9000
    message "建立mf错误"
    pause
endif
```

结果

```
//reset
3B 9F 95 81 31 F0 9F 00 66 46 53 07 20 20 10 11 22 33 44 81 90 00 E4



//--// data[ 1 ]是传输密钥
//[PBOC/PKI] Get Challenge / [GSM-CDMA] Update SSD
0084000008
8E794DCFF99927CF
9000

//建文件
84E000001C 62168202380083023F008502000086080000240000000000014E2587A
9000
```

示例

```
clear

reset

auto_response on

// data[ 1 ]是传输密钥
set data[ 1 ] = 00112233445566778899aabbccddeeff 11223344556677ee
set cipherkey = $data[ 1 ]
```

```
set maclength = 4

00 84 00 00 08
set rand

full_3des_mac 84 e0 00 00 <62 16 82023800 83023F00 85020000 8608 0000240000000000>
if sw != 9000
    message "建立mf错误"
    pause
endif
```

结果

```
//reset
3B 8C 80 01 0C 77 F7 E9 02 46 53 20 15 10 01 52 09


//--// data[ 1 ]是传输密钥
//[PBOC/PKI] Get Challenge / [GSM-CDMA] Update SSD
0084000008
B084A7BA6873903E
9000

//建文件
84E000001C 62168202380083023F008502000086080000240000000000 9062ABF7
9000
```

jcop22_ext_auth

jcop22_ext_auth_2

这个函数进行GP认证，支持scp01, 02, 03, 11[b,c],
由于历史的原因，jcop22这个名字一直未改动，
更合理的叫法可能应该叫**scp_auth**或**gp_auth**

jcop22_ext_auth 认证模式, 静态密钥enc[, 密钥mac, 密钥dek]

- 认证模式：可选0，1，3，或scp11c.1，scp11c.3，或在后面加上“|”16进制P1P2值，注意竖线不能少
  - 如果竖线后面跟scp11c.x，则表示要进行一个scp11c认证
- 静态密钥：可以跟1，2或3个密钥，密钥顺序为enc，mac，dek，如果缺少某一个，则使用第一个的值
  - 如果要进行scp11c认证，则密钥可以跟一个settings密钥文件
  - 不带密钥文件的话，将查找需要的settings，找不到停止

修改后的jcop22_ext_auth函数，会在界面上显示一条命令，显示卡片3个过程密钥，供脚本使用。

```
//Initial Update                                (defined by GP)
```

```
8050000008 A02857278C0DECDF
00003FA9863EFCAA3336FF02000F79231C718F50FE5A8A429E518BD3
9000

//External Authenticate                                (defined by GP)
8482000010 853F9744E37DA6A25DFF54E031AC24A9
9000

//下面加重的部分，不是实际发送给卡片的，只显示在界面上，供脚本使用
//this command not send to card, only display to scrren for
//next script use, return data format
//0x10 SessionSENCKey, 0x10 SessionCMACKey, 0x10 SessionCDEKKey

//External Authenticate                                (defined by GP)
8482000010 853F9744E37DA6A25DFF54E031AC24A9
8D289AFE0AB9C45B1C76DEEA182966F46C273F308271B15D2
B960E304E544F1630625987D5408159F449EA504578BDCF
9000
//resp = 8D289AFE0AB9C45B1C76DEEA182966F46C273F308271B15D2
B960E304E544F1630625987D5408159F449EA504578BDCF
```

为支持80 50指令p1值为非00，可以在认证级别后面用’|’加上1字节的16进制数p1值。

例

```
enckey  = 404142434445464748494a4b4c4d4e4f
mackey  = $enckey
dekkey  = $enckey
jcop22_ext_auth 0, 404142434445464748494a4b4c4d4e4f, 404142434445464748494a4b4c4d4e4f,
404142434445464748494a4b4c4d4e4f
jcop22_ext_auth 0, $enckey, $mackey, $dekkey
jcop22_ext_auth 1, $enckey, $mackey
jcop22_ext_auth 3, $enckey
jcop22_ext_auth 0|01, 404142434445464748494a4b4c4d4e4f, 404142434445464748494a4b4c4d4e4f,
404142434445464748494a4b4c4d4e4f
jcop22_ext_auth 0, $enckey, $mackey, $dekkey
jcop22_ext_auth 1|0103, $enckey, $mackey
jcop22_ext_auth 3, $enckey
jcop22_ext_auth scp11c|01
jcop22_ext_auth scp11c.1, "88888.settings"
jcop22_ext_auth scp11c.3, "999.settings"
```

jcop22_ext_auth_kmc

jcop22_ext_auth_kmc_2

```
jcop22_ext_auth_kmc 认证模式, kmc密钥
jcop22_ext_auth_kmc 0, 401414243444546474849 4a4b4c4d4e4f
```

jcop22_ext_auth_kmc 0|16进制P1P2值, 4014142434445464748494a4b4c4d4e4f
与 jcop22_ext_auth 功能基本一样。

jcop22_ext_auth_nonblock

jcop22_ext_auth_nonblock_2

jcop22_ext_auth_kmc_nonblock

jcop22_ext_auth_kmc_nonblock_2

这4个函数与不带nonblock的函数功能类似，区别在于这4个函数，如果认证不通过，并不会导致脚本停止运行，会继续运行，由后续脚本处理。

mac_auto_mac

mac_auto_mac_2

java卡自动计算明文+mac或密文+mac 脚本需要处理，cla置为X4，P3 为实际数据长度 + 8 clear reset auto_response on select cardmanager $cardmanager jcop22_ext_auth 3, 4041424344454647484 94a4b4c4d4e4f mac_length = 08 mac_auto_mac 80F28000 < 4F00 >
自0.0.5.0起，此函数支持scp01

自动计算与 java 卡交互的指令

jcop22_ext_auth 10 进制认证级别，静态密钥

示例	jcop22_ext_auth 3, 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 00a4000002 b001 mac_auto_mac 04e2000010 1111111111111111 mac_auto_mac 04e2000018 22222222222222222222222222222222
结果	//INITIALIZE UPDATE 或 INITIALIZE FOR ED/EP 8050000008 6880D08C01962BE200 00007118034742911734FF020003CA8210D4D251F7847BA9DB2A5AB6 9000 //[GSM-CDMA] Confrim SSD / [PBOC]EXTERNAL AUTHENTICATE 8482030010 887AC03700B2CECDA4485E55B60B4D00 9000

```
//SELECT
00A4000002 B001
9000

//APPEND RECORD
04E2000018 09402AC5DF6C80531EFC8E8B4D56C0F94ED2CCB6EA167CD9
9000
```

```
select cardmanager
select_2 cardmanager
    select cardmanager a000000003000000
    select cardmanager "1234"

delete
delete_2
    delete d156 0000 0101 0301
    delete "Applet"

upload
upload_2
    上传 java 包
    upload "java_applet.cap" [, isd] [, 10进制upload数据长度 ]

install
install_2
    安装实例
    install packageaid= "Applet", appletaid="Applet1", instanceaid=d156000001010301,
    installdata=xxxxxxxx, default_selected, privileges = xx
    安装 java 包，此功能有若干参数，如 packageaid, appletaid, instanceaid,installdata，是否默认选中，应用特权等等。

    安装时带上 default_selected 参数，则默认被选中，使用时要保证卡片没有被默认选中的应用。
    如果不带 privileges 参数，则 privileges 参数默认 00，此选项可以与 default_selected 共用，也可以代替 default_selected
```

示例

```
clear

? "author -  Snooper"
? "date   -  2015-12-24 11:16:27"
```

```
reset

auto_response on
prompt on

packageaid_01 = get_package_aid( "basic.cap" )
if $packageaid_01 != 111111111111
    ?
    pause
endif
packageaid_02 = get_package_aid( "logic.cap" )
if $packageaid_02 != 111111111112
    ?
    pause
endif
packageaid_03 = get_package_aid( "applet.cap" )
if $packageaid_03 != 111111111113
    ?
    pause
endif
install_package_aid = $packageaid_03
appletaid = get_applet_aid( "applet.cap", 0 )
instanceaid = 00000000000000 01

00 a4 04 00 00
if sw != 9000
    pause
endif
set resp
isd = gettlv( $resp, 6f 84 )

mac_length = 8
select cardmanager $isd
if sw != 9000
    pause
endif

jcop22_ext_auth 0, 404142434445464748494a4b4c4d4e4f
if sw != 9000
    ?
    pause
endif
```

```
delete $instanceaid

delete $packageaid_03
delete $packageaid_02
delete $packageaid_01

upload "basic.cap"
upload "logic.cap"
upload "applet.cap"

install packageaid = $ install_package_aid, appletaid =
$ appletaid, instanceaid = $ instanceaid, default_selected

end
```

结果

```
//reset
3B 6B 00 FF 65 46 53 06 10 07 71 C6 80 61 17



//--//上传 java 卡 test 应用脚本
//[-]=========================[X]
//|执行 test 发卡流程
//[-]=========================[-]

//SELECT
00A4040008 A000000003000000
6112

//GET RESPONSE
00C0000012
6F108408A000000003000000A5049F6501FF
9000

//INITIALIZE UPDATE 或 INITIALIZE FOR ED/EP
8050000008 1CDD4E4C126F7966
611C

//GET RESPONSE
00C000001C
00007118035201911734FF0200003D029C31C78965761E370F98D126
9000

//[GSM-CDMA] Confrim SSD / [PBOC]EXTERNAL AUTHENTICATE
```

```
8482000010 70A403717520B0507C86E981E4901FFB
9000

//--//delete instance, 这是install的
//DELETE APPLET/PACKAGE
80E400000A 4F087465737430303101
6A88

//--//delete applet, 这是packageaid
//DELETE APPLET/PACKAGE
80E4000009 4F0774657374303031
6A88

//------解析CAP文件，遇到组件 Component_Header
//------解析CAP文件，遇到组件 Component_Directory
//------解析CAP文件，遇到组件 Component_Applet
//------解析CAP文件，遇到组件 Component_Import
//------解析CAP文件，遇到组件 Component_Class
//------解析CAP文件，遇到组件 Component_ConstantPool
//------解析CAP文件，遇到组件 Component_Method
//------解析CAP文件，遇到组件 Component_Export
//------解析CAP文件，遇到组件 Component_StaticField
//------解析CAP文件，遇到组件 Component_Descriptor
//------解析CAP文件，遇到组件 Component_Debug
//------解析CAP文件，遇到组件 结束

//--整理cap内容

//[PBOC] Install / [PKI] Read Public Key
80E6020014 0774657374303031 08A000000030000000000000
6101

//GET RESPONSE
00C0000001
00
9000

//UPLOAD
80E80000C0 C482007.....................
6101

//GET RESPONSE
00C0000001
```

```
00
9000

//UPLOAD
80E880012A 0100020003800A0113.....................
6101

//GET RESPONSE
00C0000001
00
9000

//[PBOC] Install / [PKI] Read Public Key
80E60C001F 0774657374303031077465737430303 10...
9000

//[-]=========================[X]
//|test 安装成功
//[-]=========================[-]

//结束，当前行 33
```

compare

比较两串数字的不同，compare 数据1，数据2，[参数输出模式]
没有 模式 这个参数，或者 模式 参数为0，表示宽松显示，否则紧凑显示

参数为 0 时，输出结果使用宽松模式，参数为 1 时，输出结果为紧凑模式。
输出模式可以省略，默认为 0

数据 1，数据 2 内有相同的部分，则用绿色输出，不同部分，使用红色输出。

示例

```
clear

compare 1122, 3344
compare 1122, 11, 0
compare 1122, 112233, 1
compare 1122, 3322, 1
```

结果

```
//------------------------ Compare ------------------------
//------ 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ----  00 01 0
// 0000  11 22                                            ----  33 44
//
//
//------------------------ Compare ------------------------
//------ 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ----  00 01 0
// 0000  11 22                                            ----  11 --
//
//
//------------------------ Compare ------------------------
//------ 000102030405060708090A0B0C0D0E0F ---- 000102030405060708090A0B
// 0000  1122--                            ---- 112233
//
//
//------------------------ Compare ------------------------
//------ 000102030405060708090A0B0C0D0E0F ---- 000102030405060708090A0B
// 0000  1122                              ---- 3322
//
```

socket_message（已淘汰）

socket_message “ip:port”, “message”

例如 socket_message “127.0.0.1:9998”, “next card”

generate rsa_keypair（已淘汰）

generate rsa_keypair 10进制输入模长, 输入的E, N的变量名, D的变量名, P的变量名, Q的变量名, DP的变量名, DQ的变量名, qinv的变量名

generate rsa_keypair 512, 010001, myn, myd, myp, myq, mydp, mydq, myu

变量若不存在，会自动分配

generate ecc_keypair（已淘汰）

generate ecc_Keypair 任意指定私钥, 公钥索引

注：生成 ecc 密钥对之前，请使用 ecc_curve_id = 曲线 id 选择 ecc 曲线。

示例

```
ecc_curve_id = 411
generate ecc_keypair 11223344, 公钥

? "公钥 = " $公钥
```

结果

```
//[]============================================================================
==========================[]
//[]  公钥 =
B8AEC1A2A646ECA9BD96C861748E61BA6868FD26A47990A0DCF7C76BC8F6C4B38CCE536A77FE6805BBC74
E05A4281A65    []
```

```
//[]============================================================================
==========================[]
```

generate sm2_keypair（已淘汰）

generate sm2_Keypair 公钥, 私钥
生成的密钥对将被保存到变量名中

示例	generate sm2_keypair 公钥, 私钥 ? “公钥 = “ $公钥 ? “私钥 = “ $私钥
结果	//[]== ==========================[] //[] 公钥 = 9BF956C1F2F00B89F832E99A61EB14CE4DD3C1C6A42177452533AF3F0CD6EDAF0C1339DD50416650BC7E0 8500980933D057D895EC35C43DABD2B0A0063C02232 [] //[] 私钥 = E94B8CF70D484E1EC1CB0F2925497E883D84D8EB03EE904233BED6BD57A37DCC [] //[]== ==========================[]

示例 1

```
generate rsa_keypair 64, 010001, n, d, p, q, dp, dq, u

? "n  = " $n
? "d  = " $d
? "p  = " $p
? "q  = " $q
? "dp = " $dp
? "dq = " $dq
? "u  = " $u
```

示例2 ecc

```
ecc_curve_id = 411
generate ecc_keypair 11223344, 公钥

? "公钥 = " $公钥
```

示例3 sm2

```
generate sm2_keypair 公钥, 私钥

? "公钥 = " $公钥
? "私钥 = " $私钥

明文 = random( 40 )
? "明文 = " $明文

密文 = sm2_pub_encode( $明文, $公钥 )
? "密文 = " $密文

明文2 = sm2_pri_decode( $密文, $私钥 )
if $明文 != $明文2
    ?
    pause
endif

hash = dup( 32, 01 )
签名 = sm2_sign( $hash, $私钥 )
r = left( $签名, 32 )
s = right( $签名, 32 )

? $签名
? $r
? $s

验签 = sm2_sign_verify( $公钥, $hash, $r, $s )
if $验签 != 01
    ?
    pause
endif

? "搞定"
```

main（已淘汰）

基本无用，与以前的脚本兼容

sendtext（已淘汰）

主要用来发送一个以$开头的字符串

split

实现多个“message”语句或多个 “?”语句的分行输出

示例

```
? ″  fdda time ″
? ″ select ppse aid      ″ $__       ″    uS″
? ″ select pboc aid      ″ $aa       ″    uS″
? ″ gpo                  ″ $cc       ″    uS″
? ″ read                 ″ $x0       ″    uS″
? ″ read                 ″ $x1       ″    uS″
? ″ read                 ″ $x2       ″    uS″
? ″ read                 ″ $x3       ″    uS″
? ″ read                 ″ $x4       ″    uS″
? ″ read                 ″ $x5       ″    uS″
? ″ read                 ″ $x6       ″    uS″
? ″ read                 ″ $x7       ″    uS″
? ″ total                ″ $xx       ″    uS″

split

? ″qpboc offline trade ok″
```

结果

```
//[]=======================================[]
//[]    fdda time                          []
//[]   select ppse aid     026718    uS    []
//[]   select pboc aid     029841    uS    []
//[]   gpo                 149345    uS    []
//[]   read                013441    uS    []
//[]   read                034332    uS    []
//[]   read                030067    uS    []
//[]   read                029193    uS    []
//[]   read                042570    uS    []
//[]   read                00    uS        []
//[]   read                00    uS        []
//[]   read                00    uS        []
//[]   total               355507    uS    []
//[]=======================================[]

//[]============================[]
//[]  qpboc offline trade ok    []
//[]============================[]
```

push

向堆栈内压入一个数据，数据可以是变量，也可以是一个函数的返回值。

```
    push 11223344
    push $k
    push des_des_mac( aa, bb, cc )
```

inc_indent 和 dec_indent

用于控制代码缩进，适当使用可使代码结构更加清晰

```
    inc_indent
        call cbor_int( 01 )
        call cbor_text( $s_website )
    dec_indent

    inc_indent
        call cbor_int( 02 )
        call cbor_byte( $g_in_client_hash )
    dec_indent
```

undef

用于控制删除一个已经存在的变量，可配合 var_exist("x")来使用

```
    z   = 4444444488

    x   = 9988

    eee = eeeeeeeeeeeeeeeeeeeeeeeeeeeeee
    y   = var_exist( "x" )

    if $y == 01
        ? "x is defined"
        ? $eee
    else
        ? "x not define"
        ? $eee
    endif
```

函数

函数用来计算某操作，这一版 apdu 脚本语言中，**函数不支持嵌套、不支持在一行内使用多个函数、不支持在常量表达式后面使用**。

错误示例 1：

函数嵌套

```
a = aes128_decode_cbc( random( 8 ), 0000, 0000 )
```

错误示例 2：

多个函数

```
a = aes128_decode_cbc( 0000, 0000, 0000 ) sha1_hash( 11223344 )
```

错误示例 3：

在常量后面使用函数

```
a = 0000 sha1_hash( 11223344 )
```

函数与宏的返回结果一般存储在某个变量中，变量自身可以做为函数运算的参数，这一部分将在 SET 关键字进行详细介绍。

使用 dummy 接口，可以做到在一行 apdu 内同时执行多个函数，函数可以嵌套。

函数的一般用法是 **变量 = 函数(参数)**

set 关键字

set关键字在大部分时候可以忽略，具体忽略条件为，当set后面是一个完整的表达式时，比如 set t = aabbccdd 这个语句，”t = aabbccdd”是一个完整的表达式，此时set关键字可以忽略。

以下的表格忽略关键字

rand	将apdu接收数据或用户指定数据设为随机数
set rand	
rand = 字符串	
rand = 11223344	
ecc_curve_id	选择ecc曲线
cipherkey	发送类似des_des_mac为前缀关键字等带mac的计算apdu时，要设立一个cipherkey，并取随机数为ICV，对输入数据进行加密才行。
cipherkey = 0102030405060708090a0b0c0d0e0f10	
cipherkey = "xxxx"	
cipherkey = $data[9]	
mac_length	set mac_length = 十进制长度 注：此设置影响 ”<” 的计算和mac结果的长度
变量名 = $变量名 1	
变量名 = 函数	

set 变量, offset, length
set 变量, offset
set 变量
set 变量[index], offset, length
set 变量[index], offset
set 变量[index]

set 函数列表

加解密部分

des_encode_ecb

des_encode_ecb	变量名 = des_encode_ecb(数据, 密钥)

des_decode_ecb

des_decode_ecb	变量名 = des_decode_ecb(数据, 密钥)

3des_encode_ecb

3des_encode_ecb	变量名 = 3des_encode_ecb(数据, 密钥)

3des_decode_ecb

3des_decode_ecb	变量名 = 3des_decode_ecb(数据, 密钥)

3des_24k_encode_ecb

3des24_encode_ecb

3des_24k_encode_ecb	变量名 = 3des_24k_encode_ecb(数据, 密钥)

3des_24k_decode_ecb

3des24_decode_ecb

3des_24k_decode_ecb	变量名 = 3des_24k_decode_ecb(数据, 密钥)

des_encode_cbc

des_encode_cbc	变量名 = des_encode_cbc(随机数, 数据, 密钥)
	本函数不进行80对齐

des_decode_cbc

des_decode_cbc	变量名 = des_decode_cbc(随机数, 数据, 密钥)
	本函数不进行80对齐

3des_encode_cbc

3des_encode_cbc	变量名 = 3des_encode_cbc(随机数，数据，密钥)
	本函数不进行80对齐

3des_decode_cbc

3des_decode_cbc	变量名 = 3des_decode_cbc(随机数，数据，密钥)
	本函数不进行80对齐

3des_24k_encode_cbc

3des24_encode_cbc

3des_24k_encode_cbc	变量名 = 3des_24k_encode_cbc(随机数，数据，密钥)
	本函数不进行80对齐

3des_24k_decode_cbc

3des24_decode_cbc

3des_24k_decode_cbc	变量名 = 3des_24k_decode_cbc(随机数，数据，密钥)
	本函数不进行80对齐

aes128_encode_ecb

aes128_encode_ecb	变量名 = aes128_encode_ecb(数据，密钥)
	本函数不进行80对齐

aes128_decode_ecb

aes128_decode_ecb	变量名 = aes128_decode_ecb(数据，密钥)
	本函数不进行80对齐

aes128_encode_cbc

aes128_encode_cbc	变量名 = aes128_encode_cbc(随机数，数据，密钥)
	本函数不进行80对齐

aes128_decode_cbc

aes128_decode_cbc	变量名 = aes128_decode_cbc(随机数,数据，密钥)
	本函数不进行80对齐

aes192_encode_ecb

aes192_encode_ecb	变量名 = aes192_encode_ecb(数据，密钥)
	本函数不进行80对齐

aes192_decode_ecb

aes192_decode_ecb	变量名 = aes192_decode_ecb(数据，密钥)
	本函数不进行80对齐

aes192_encode_cbc

aes192_encode_cbc	变量名 = aes192_encode_cbc(随机数, 数据, 密钥)
	本函数不进行80对齐

aes192_decode_cbc

aes192_decode_cbc	变量名 = aes192_decode_cbc(随机数,数据, 密钥)
	本函数不进行80对齐

aes256_encode_ecb

aes256_encode_ecb	变量名 = aes256_encode_ecb(数据, 密钥)
	本函数不进行80对齐

aes256_decode_ecb

aes256_decode_ecb	变量名 = aes256_decode_ecb(数据, 密钥)
	本函数不进行80对齐

aes256_encode_cbc

aes256_encode_cbc	变量名 = aes256_encode_cbc(随机数, 数据, 密钥)
	本函数不进行80对齐

aes256_decode_cbc

aes256_decode_cbc	变量名 = aes256_decode_cbc(随机数,数据, 密钥)
	本函数不进行80对齐

chacha20

chacha20	变量名 =chacha20(明文或密文, 32字节密钥,1-4字节16进制count, nonce)

```
clear
key = oem_strset( 32, 00, 01 )
count = 01
nonce = 00 00 00 00 00 00 00 4a 00 00 00 00
kk = ""
kk = $kk 4c 61 64 69 65 73 20 61 6e 64 20 47 65 6e 74 6c
kk = $kk 65 6d 65 6e 20 6f 66 20 74 68 65 20 63 6c 61 73
kk = $kk 73 20 6f 66 20 27 39 39 3a 20 49 66 20 49 20 63
kk = $kk 6f 75 6c 64 20 6f 66 66 65 72 20 79 6f 75 20 6f
kk = $kk 6e 6c 79 20 6f 6e 65 20 74 69 70 20 66 6f 72 20
kk = $kk 74 68 65 20 66 75 74 75 72 65 2c 20 73 75 6e 73
kk = $kk 63 72 65 65 6e 20 77 6f 75 6c 64 20 62 65 20 69
kk = $kk 74 2e

ciphertext = chacha20( $kk, $key, $count, $nonce )
```

```
if $ciphertext != 6e 2e 35 9a 25 68 f9 80 41 ba 07 28 dd 0d 69 81 e9
7e 7a ec 1d 43 60 c2 0a 27 af cc fd 9f ae 0b f9 1b 65 c5 52 47 33 ab
8f 59 3d ab cd 62 b3 57 16 39 d6 24 e6 51 52 ab 8f 53 0c 35 9f 08 61
d8 07 ca 0d bf 50 0d 6a 61 56 a3 8e 08 8a 22 b6 5e 52 bc 51 4d 16 cc
f8 06 81 8c e9 1a b7 79 37 36 5a f9 0b bf 74 a3 5b e6 b4 0b 8e ed f2
78 5e 42 87 4d
    ?
    pause
endif
ciphertext = chacha20( $ciphertext, $key, $count, $nonce )
if $ciphertext != $kk
    ?
    pause
endif

end
```

poly1305_init

poly1305_init	变量名 = poly1305_init(32字节密钥)
	固定返回00

poly1305_update

poly1305_update	变量名 = poly1305_update(数据)
	固定返回00

poly1305_finish

poly1305_finish	变量名 = poly1305_finish()
	固定返回16字节的mac poly1305_key = dup(16, 00) poly1305_key = $poly1305_key 36 e5 f6 b5 c5 e0 60 70 f0 ef ca 96 22 7a 86 3e text_to_mac = "" text_to_mac = $text_to_mac 41 6e 79 20 73 75 62 6d 69 73 73 69 6f 6e 20 74 text_to_mac = $text_to_mac 6f 20 74 68 65 20 49 45 54 46 20 69 6e 74 65 6e text_to_mac = $text_to_mac 64 65 64 20 62 79 20 74 68 65 20 43 6f 6e

```
74 72
text_to_mac = $text_to_mac 69 62 75 74 6f 72 20 66 6f 72 20 70 75 62
6c 69
text_to_mac = $text_to_mac 63 61 74 69 6f 6e 20 61 73 20 61 6c 6c 20
6f 72
text_to_mac = $text_to_mac 20 70 61 72 74 20 6f 66 20 61 6e 20 49 45
54 46
text_to_mac = $text_to_mac 20 49 6e 74 65 72 6e 65 74 2d 44 72 61 66
74 20
text_to_mac = $text_to_mac 6f 72 20 52 46 43 20 61 6e 64 20 61 6e 79
20 73
text_to_mac = $text_to_mac 74 61 74 65 6d 65 6e 74 20 6d 61 64 65 20
77 69
text_to_mac = $text_to_mac 74 68 69 6e 20 74 68 65 20 63 6f 6e 74 65
78 74
text_to_mac = $text_to_mac 20 6f 66 20 61 6e 20 49 45 54 46 20 61 63
74 69
text_to_mac = $text_to_mac 76 69 74 79 20 69 73 20 63 6f 6e 73 69 64
65 72
text_to_mac = $text_to_mac 65 64 20 61 6e 20 22 49 45 54 46 20 43 6f
6e 74
text_to_mac = $text_to_mac 72 69 62 75 74 69 6f 6e 22 2e 20 53 75 63
68 20
text_to_mac = $text_to_mac 73 74 61 74 65 6d 65 6e 74 73 20 69 6e 63
6c 75
text_to_mac = $text_to_mac 64 65 20 6f 72 61 6c 20 73 74 61 74 65 6d
65 6e
text_to_mac = $text_to_mac 74 73 20 69 6e 20 49 45 54 46 20 73 65 73
73 69
text_to_mac = $text_to_mac 6f 6e 73 2c 20 61 73 20 77 65 6c 6c 20 61
73 20
text_to_mac = $text_to_mac 77 72 69 74 74 65 6e 20 61 6e 64 20 65 6c
65 63
text_to_mac = $text_to_mac 74 72 6f 6e 69 63 20 63 6f 6d 6d 75 6e 69
63 61
text_to_mac = $text_to_mac 74 69 6f 6e 73 20 6d 61 64 65 20 61 74 20
61 6e
text_to_mac = $text_to_mac 79 20 74 69 6d 65 20 6f 72 20 70 6c 61 63
65 2c
text_to_mac = $text_to_mac 20 77 68 69 63 68 20 61 72 65 20 61 64 64
72 65
text_to_mac = $text_to_mac 73 73 65 64 20 74 6f
```

	b = poly1305_init($poly1305_key) b = poly1305_update($text_to_mac) b = poly1305_finish()

数据处理部分

utf8_string

utf8_string	变量名 = utf8_string(字符串)
	将数据（一般为字符串）转换成utf8格式的数据

utf16_little_string

utf16_little_string	变量名 = utf16_little_string(字符串)
	将数据（一般为字符串）转换成utf16_little格式的数据（windows标准格式）

utf16_big_string

utf16_big_string	变量名 = utf16_big_string(字符串)
	将数据（一般为字符串）转换成utf16_big格式的数据（与windows标准格式高低位相反）

ansi_string

ansi_string	变量名 = ansi_string(字符串)
	无用，与前面几个参数配套用

shl

shl	变量名 = shl(数据，10进制移位次数)
	数据左移

shr

shr	变量名 = shr(数据，10进制移位次数)
	数据右移

rol

rol	变量名 = rol(数据，10进制移位次数)
	数据循环左移

ror

ror	变量名 = ror(数据，10进制移位次数)
	数据循环右移

mid

mid	变量名 = mid(数据，10进制offset，10进制length)
	如果数据后面不接参数，将所有的数据保存到变量中 函数会根据 数据 的内容来决定是否能够取出来数据，如果不能取出来，那么变量的长度为0 Note:也可以只接offset

hmid

hmid	变量名 = hmid(数据，16进制offset，16进制length)
	如果数据后面不接参数，将所有的数据保存到变量中 函数会根据 数据 的内容来决定是否能够取出来数据，如果不能取出来，那么变量的长度为0 Note:也可以只接offset

left

left	变量名 = left(数据，10进制length)

hleft

hleft	变量名 = hleft(数据，16进制length)

right

right	变量名 = right(数据，10进制length)

hright

hright	变量名 = hright(数据，16进制length)

rightmid

rightmid	变量名 = rightmid(数据，10进制offset，10进制length)
	数据的最右边的offset为0，向左数到offset处，然后向右取length个数据，以如下数据为例 rightmid(0102030405060708090a, 3, 2)

01	02	03	04	05	06	07	08	09	0a
9	8	7	6	5	4	3	2	1	0
						开始处			
						取出	结果		

hrightmid

hrightmid	变量名 = hrightmid(数据，16进制offset，16进制length)
	数据的最右边的offset为0，向左数到offset处，然后向右取length个数据，以如下数据为例 hrightmid(0102030405060708090a，03，02)

01	02	03	04	05	06	07	08	09	0a
9	8	7	6	5	4	3	2	1	0
						开始处			
						取出	结果		

dup

dup	变量名 = dup(10进制重复次数，16进制数据)
	与$dup功能相同，$dup有问题，可以用dup

hdup

hdup	变量名 = hdup(16进制重复次数，16进制数据)

random

random	变量名 = random(10进制长度)

hrandom

hrandom	变量名 = hrandom(16进制长度)

reverse_block_byte

reverse_block_byte	变量名 = reverse_block_byte(数据)

read_fifo

read_fifo	变量名 = read_fifo()

fixed80

fixed80	变量名 = fixed80(数据)
	数据以80 00 00 。。。 8字节对齐

fixed80_16

fixed80_16	变量名 = fixed80_16(数据)
	数据以80 00 00 。。。 16字节对齐

add

add	变量名 = add(数据1, 数据2)
	注：此计算按数据1的长度为结果长度，超过数据1的长度，则使用新的长度 如 add(ffff, 01)，则结果为010000。

sub

sub	变量名 = sub(数据1, 数据2)

datalen

datalen	变量名 = datalen(数据)
	返回2字节长16进制的数据的长度

bcdadd

bcdadd	变量名 = bcdadd(数据1, 数据2)
	bcd数据相加

bcdsub

bcdsub	变量名 = bcdsub(数据1, 数据2)
	bcd数据相减

bcdmul

bcdmul	变量名 = bcdmul(数据1, 数据2)
	bcd数据相乘

bcddiv

bcddiv	变量名 = bcddiv(数据1, 数据2)
	bcd数据相除

bcdmod

bcdmod	变量名 = bcdmod(数据1, 数据2)
	bcd数据求余

strlen

strlen	变量名 = strlen(数据)
	与datalen相似，不过当len小于0x100时，返回一个字节的数据

xor

xor	变量名 = xor(数据1, 数据2)

	结果长度为数据2的长度

xor2

xor2	变量名 = xor2(数据1，数据2)
	结果长度为最长数据长度

and

and	变量名 = and(数据1，数据2)
	结果长度为数据2的长度

and2

and2	变量名 = andx(数据1，数据2)
	结果长度为最长数据长度

or

or	变量名 = or(数据1，数据2，…，数据10)
	结果长度为最长数据长度

hex2int

hex2int	变量名 = hex2int(16进制数据)
	输入16进制数据，输出16进制格式的10进制数据，比如set dec = hex2int(14)，则dec内容是20，可用于在使用10进制数据的地方。

int2hex

int2hex	变量名 = int2hex(10进制数据)
	输入10进制数据，输出16进制格式数据

mul

mul	变量名 = mul（16进制数据1，16进制数据2）
	mul（16进制数据1，16进制数据2）两个16进制数相乘，返回数据的长度不小于数据1的长度

div

div	变量名 = div（16进制数据1，16进制数据2）
	div（16进制数据1，16进制数据2）两个16进制数相除，返回数据的长度不小于数据1的长度

mod

mod	变量名 = mod（16进制数据1，16进制数据2）
	mod（16进制数据1，16进制数据2）求余数，返回数据长度不大于数据2的长度

itoa, atoi

itoa	变量名 = itoa（16进制数据）
atoi	变量名 = atoi（16进制数据）
	为兼容一些脚本而保留的函数，输入什么返回什么。输入多长返回多长。

compressed_bcd

compressed_bcd	变量名 = compressed_bcd(非压缩格式的bcd)
	比如输入 01 02 03 04 05，结果是12 34 50

uncompressed_bcd

uncompressed_bcd	变量名 = uncompressed_bcd(非压缩格式的bcd)
	比如输入 aa bb cc，结果是0a 0a 0b 0b 0c 0c

gettlv

gettlv	变量名 = gettlv(要查找的tlv，要查找中的tag，最后一个tag的编号)

	要查找的tag，可以使用模板，比如tag为70，则返回70这个tag的内容，再如tag为705f20，则查找tag中，70这个tlv中的5f20这个tlv中的V值。
	最后一个tag的编号要求16进制，从00开始
	比如
	//cardaccess =
	3170300D060804007F000702020202010130 0F060A04007F000702020302010201013012060A04007F000702020402020201020201 0D3012060A04007F000702020402010201020201 0D3012060A04007F000702020401020201020201 003012060A04007F000702020401010201020201 00
	要使用脚本
	xx = gettlv($cardaccess, 31 30, 02)
	得到
	//xx = 060A04007F000702020402020201020201 0D

gettlv_single

gettlv_single	变量名 = gettlv_single(要查找的tlv，要查找中的tag，最后一个tag的编号)
	用法与gettlv类似，不过tag与长度都是1字节。此函数未经过详细测试。

gettlv_scp11c

gettlv_scp11c	变量名 = gettlv_scp11c(要查找的tlv，要查找中的tag，最后一个tag的编号)
	用法与gettlv类似，不过对部分做特殊处理。此函数未经过详细测试。

get_dol_tag

get_dol_tag	变量名 = get_dol_tag(dol_数据，16进制索引)
	索引从00开始，如果索引给的超过了dol中的实际范围，那么返回结果的长度为0，返回结果为空。
示例	dol = 9F 66 04 9F 02 06 9F 03 06 9F 1A 02 95 05 5F 2A 02 9A 03 9C 01 9F 37 04 xxx = "" hfor a = 00 to 10

```
    tag = get_dol_tag( $dol, $a )
    if $tag == ""
        exit hfor
    else if $tag == 9f02
        xxx = $xxx 00000000 0100
    else if $tag == 9f03
        xxx = $xxx 000000000000
    else if $tag == 95
        xxx = $xxx 0000000000
    else if $tag == 9a
        eee = datetime()
        eee = hmid( $eee, 01, 03 )
        xxx = $xxx $eee
    else if $tag == 9c
        eee = datetime()
        eee = hright( $eee, 03 )
        xxx = $xxx $eee
    else if $tag == 9f37
        eee = hrandom( 04 )
        xxx = $xxx $eee
    else if $tag == 9f1a
        xxx = $xxx 0156
    else if $tag == 5f2a
        xxx = $xxx 0156
    else
        ? "不认识的tag"
    endif
hnext a
```

结果

```
// DOL = 9F66049F02069F03069F1A0295055F2A029A039C019F3704
// XXX =
// TAG = 9F66
//[-]========================[X]
//|不认识的tag
//[-]========================[-]

// TAG = 9F02
// XXX = 000000000100
// TAG = 9F03
// XXX = 000000000100000000000000
// TAG = 9F1A
// XXX = 00000000010000000000000000156
// TAG = 95
// XXX = 0000000001000000000000000156000000000000
```

	// TAG = 5F2A
	// XXX = 00000000010000000000000001560000000000156
	// TAG = 9A
	// EEE = 20131120164111
	// EEE = 131120
	// XXX = 000000000100000000000000015600000000000156131120
	// TAG = 9C
	// EEE = 20131120164111
	// EEE = 164111
	// XXX = 0000000001000000000000000156000000000001561311201641111
	// TAG = 9F37
	// EEE = 0DF28D8D
	// XXX = 000000000100000000000000015600000000000156131120164111 0DF28D8D
	// TAG =

一些封装的行业函数

get_package_aid

get_package_aid	变量名 = get_package_aid (“文件名”)

get_applet_aid

get_applet_aid	变量名 = get_applet_aid (“文件名”, applet在包内的索引)
	索引从0开始 xx = get_applet_aid(“tt.cap”, 0) 索引可以忽略，兼容以前的脚本。

diversify

diversify	变量名 = diversify(数据，密钥)
	支持多级分散

oem_subkey

oem_subkey	同 diversify

oem_encrdata

oem_encrdata	变量名 = oem_encrdata(key, data, mode)
	将数据前面添加长度，后边补上80，然后按key进行des或3des加密

	oem_encrdata(key, data, mode) 金融，加油数据加密，密钥直接加密/解密，不需要取随机数，不需要计算过程密钥 mode=0，加密 mode=1，解密

oem_desmac

oem_desmac	变量名 = oem_desmac(key, data, rand, mode)
	自动补80，然后自动按des或3des算mac oem_desmac(key,data,rand,mode) mode bit 0位为0时，金融，加油计算MAC，当数据长度为8的整数倍时仍然补80，最低位为1时，长度为8的整数倍时不补80. mode bit 1位为0时，产生mac为4字节，bit1为1时，产生mac为8字节。 0 强制补80，4字节mac 1 长度为8时，不自动补80，4字节mac 2 强制补80，8字节mac 3 长度为8时，不自动补80，8字节mac

oem_desjs

oem_desjs	变量名 = oem_desjs(key, data, mode)
	ecb模式按des或3des计算 DES,3DES计算，如果密钥长度为8进行单DES计算，如果密钥长为16进行3DES计算，mode=00 加密 mode=01 解密

oem_xortac

oem_xortac	变量名 = oem_xortac(密钥，数据)
	oem_xortac (密钥，数据)，将密钥进行xor之后，对数据进行mac计算，主要用于算tac。

oem_strset

oem_strset	变量名 = oem_strset(10进制总数a，16进制数据b，16进制步长c)
	oem_strset(10进制总数a，16进制数据b，16进制步长c)，把数据b复制a份，并连接在一起构成新的数据，每复制一次b增加c大小，如果 16进制步长C没有写，默认为00。

calc_afl

calc_afl	变量名 = calc_afl(afl的值)
	将 afl 置换成 bit 图，供某些程序使用，返回数据固定为 10 字节，所以要求 afl 中的数据必须在 1-8 号文件之间，记录个数也在 1-8 号之间。 返回数据 afl_start （1字节，从0开始） afl_长度（1字节），afl位图（8字节） 80 df 02 00 用于 fdda 00

	80 df 02 01 用于 fdda 01 80 df 02 20 用于 sm fdda 00 80 df 02 21 用于 sm fdda 01

calc_cdol_len

calc_cdol_len	变量名 = calc_cdol_len (cdol的值)
	80 df 02 02 用于 cdol1 80 df 02 03 用于 cdol2

calc_log_by_pdol

calc_log_by_pdol	变量名 = calc_log_by_pdol (pdol的值, log的值)
	80 df 02 04 用于接触式 pdol 与日志 80 df 02 05 用于非接触式 pdol 与日志 80 df 02 06 用于接触式 pdol 与圈存日志 80 df 02 07 用于非接触式 pdol 与圈存日志

calc_pdol_len

calc_pdol_len	变量名 = calc_pdol_len (pdol的值)
	80 df 02 08 用于接触式 pdol 80 df 02 09 用于非接触式 pdol

calc_log_by_cdol

calc_log_by_cdol	变量名 = calc_log_by_cdol (cdol的值, log的值)
	80 df 02 0a 用于接触式 cdol1 与日志 80 df 02 0b 用于非接触式 cdol2 与日志 80 df 02 0c 用于接触式 cdol1 与圈存日志 80 df 02 0d 用于非接触式 cdol2 与圈存日志

calc_ddol_len

calc_log_by_cdol	变量名 = calc_ddol_len (ddol的值)
	80 df 02 0e 用于 ddol 的值

HASH 部分

sha1_hash

sha1_hash	变量名 = sha1_hash(数据)

sha1_hash_nonfill

sha1_hash_nonfill	变量名 = sha1_hash_nonfill(数据)
	按非填充模式计算sha1 hash值，数据长度必须为64的整数倍，结果长度为20

sha1_hash_lastblock

sha1_hash_lastblock	变量名 = SHA1_hash_lastblock(10进制前面数据长度，20字节的非填充hash值，小于64字节的最后一块数据)
	根据部分数据计算sha1填充数据的最终结果

sha1_hash_init

sha1_hash_init	变量名 = sha1_hash_init(数据) 建议为64字节的整数倍

sha1_hash_update

sha1_hash_update	变量名 = sha1_hash_update(数据) 建议为64字节的整数倍

sha1_hash_dofinal

sha1_hash_dofinal	变量名 = sha1_hash_dofinal(数据) 长度任意

sha256_hash

sha256_hash	变量名 = sha256_hash(数据)

sha256_hash_nonfill

sha256_hash_nonfill	变量名 = sha256_hash_nonfill(数据)
	按非填充模式计算sha256 hash值，数据长度必须为64的整数倍，结果长度为32

sha256_hash_lastblock

sha256_hash_lastblock	变量名 = SHA256_hash_lastblock(10进制前面数据长度，32字节的非填充hash值，小于64字节的最后一块数据)
	根据部分数据计算sha256填充数据的最终结果

sha384_hash

sha384_hash	变量名 = sha384_hash(数据)

sha384_hash_nonfill

sha384_hash_nonfill	变量名 = sha384_hash_nonfill(数据)
	按非填充模式计算sha384 hash值，数据长度必须为128的整数倍，结果长度为64

sha384_hash_lastblock

sha384_hash_lastblock	变量名 = SHA384_hash_lastblock(10进制前面数据长度，64字节的非填充hash值，小于128字节的最后一块数据)
	根据部分数据计算sha384填充数据的最终结果

sha512_hash

sha512_hash	变量名 = sha512_hash(数据)

sha512_hash_nonfill

sha512_hash_nonfill	变量名 = sha512_hash_nonfill(数据)
	按非填充模式计算sha512 hash值，数据长度必须为128的整数倍，结果长度为64

sha512_hash_lastblock

sha512_hash_lastblock	变量名 = SHA512_hash_lastblock(10进制前面数据长度，64字节的非填充hash值，小于128字节的最后一块数据)
	根据部分数据计算sha512填充数据的最终结果

md5_hash

md5_hash	变量名 = md5_hash(数据)

md5_hash_nonfill

md5_hash_nonfill	变量名 = md5_hash_nonfill(数据)
	按非填充模式计算md5 hash值，数据长度必须为64的整数倍，结果长度为16

md5_hash_lastblock

md5_hash_lastblock	变量名 = md5_hash_lastblock(10进制前面数据长度，64字节的非填充hash值，小于64字节的最后一块数据)
	根据部分数据计算md5填充数据的最终结果

sha224_hash

sha224_hash	变量名 = sha224_hash(数据)

sha224_hash_init

sha224_hash_init	变量名 = sha224_hash_init(数据)

sha224_hash_update

sha224_hash_update	变量名 = sha224_hash_update(数据)

sha224_hash_dofinal

sha224_hash_dofinal	变量名 = sha224_hash_dofinal(数据)

sha3_224_hash

sha3_224_hash	变量名 = sha3_224_hash(数据)

sha3_224_hash_init

sha3_224_hash_init	变量名 = sha3_224_hash_init(数据)

sha3_224_hash_update

sha3_224_hash_update	变量名 = sha3_224_hash_update(数据)

sha3_224_hash_dofinal

sha3_224_hash_dofinal	变量名 = sha3_224_hash_dofinal(数据)

sha256_hash

sha256_hash	变量名 = sha256_hash(数据)

sha256_hash_init

sha256_hash_init	变量名 = sha256_hash_init(数据)

sha256_hash_update

sha256_hash_update	变量名 = sha256_hash_update(数据)

sha256_hash_dofinal

sha256_hash_dofinal	变量名 = sha256_hash_dofinal(数据)

sha384_hash

sha384_hash	变量名 = sha384_hash(数据)

sha384_hash_init

sha384_hash_init	变量名 = sha384_hash_init(数据)

sha384_hash_update

sha384_hash_update	变量名 = sha384_hash_update(数据)

sha384_hash_dofinal

sha384_hash_dofinal	变量名 = sha384_hash_dofinal(数据)

sha512_hash

sha512_hash	变量名 = sha512_hash(数据)

sha512_hash_init

sha512_hash_init	变量名 = sha512_hash_init(数据)

sha512_hash_update

sha512_hash_update	变量名 = sha512_hash_update(数据)

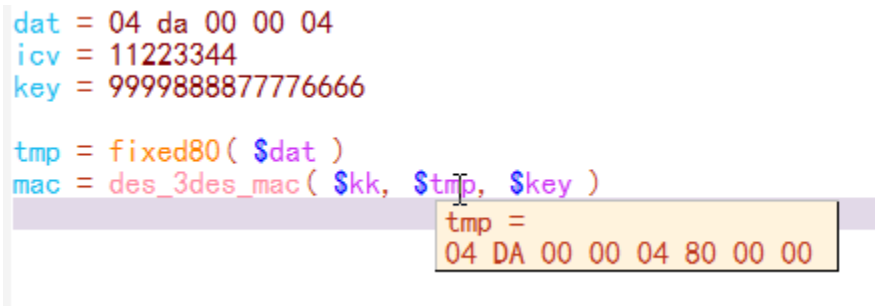
sha512_hash_dofinal

sha512_hash_dofinal	变量名 = sha512_hash_dofinal(数据)

mac 部分

des_des_mac

des_des_mac	变量名 = des_des_mac(随机数，数据，密钥)

	本函数不进行80对齐，使用时按需求决定 数据是否要提前用fixed_80填充 

des_3des_mac

des_3des_mac	变量名 = des_3des_mac(随机数，数据，密钥)
	本函数不进行80对齐，见des_des_mac

full_3des_mac

full_3des_mac	变量名 = full_3des_mac(随机数，数据，密钥)
	本函数不进行80对齐，见des_des_mac

非对称部分

rsa_nd_decode

rsa_nd_decode	变量名 = rsa_nd_decode(10进制密钥对长度，密钥N，密钥D，数据)

rsa_crt_decode

rsa_crt_decode	变量名 = rsa_crt_decode(10进制密钥对长度，密钥P，密钥Q，密钥dP，密钥dQ，密钥U,数据)

rsa_pub_encode

rsa_pub_encode	变量名 = rsa_pub_encode(10进制密钥对长度，密钥E，密钥N，数据)

ecc_sign_verify

ecc_sign_verify	变量名 = ecc_sign_verify(公钥，hash值，R值，S值)
	公钥长度应该是两倍的R或S的长度，以公钥为192 bit为例，实际的公钥长度为48字节，R,S都是24字节 ecc使用之前，要先选择ecc曲线。 此版本返回01成功，返回00失败

ecc_sign

ecc_sign	变量名 = ecc_sign(hash值，私钥)
	返回数据是R和S，以公钥为192 bit为例，实际的返回数据长度为48字节，R,S

	都是24字节 ecc使用之前，要先选择ecc曲线。

ecc_pub_encode

ecc_pub_encode	变量名 = ecc_pub_encode (数据，公钥，随机数)
	ecc加密时，明文的长度应该是8的整数倍，如果不满足此要求，会自动补齐成8的整数倍 ecc加密时，数据不能是全00，也不能大于（小P） ecc使用之前，要先选择ecc曲线。

ecc_pri_decode

ecc_pri_decode	变量名 = ecc_pri_decode (数据，私钥)
	ecc使用之前，要先选择ecc曲线。

国密算法部分

sm2_pub_encode

sm2_pub_encode	变量名 = sm2_pub_encode(数据，公钥)

sm2_pri_decode

sm2_pri_encode	变量名 = sm2_pri_encode(数据，私钥)

sm2_ecc_dh

sm2_ecc_dh	变量名 = sm2_ecc_dh(私钥1，公钥2)

sm2_sign

sm2_sign	变量名 = sm2_sign(hash, 私钥)
	返回R和S

sm2_sign_verify

sm2_sign_verify	变量名 = sm2_sign_verify(公钥， hash, R, S)
	此版本返回01成功，其他值失败

sm2_generate_pub_by_pri

sm2_generate_pub_by_pri	变量名 = sm2_generate_pub_by_pri(私钥)
	返回公钥值

sm3_hash

sm3_hash	变量名 = sm3_hash(数据)
	实现的是sm3_256功能，返回32字节数据

sm3_hash_nonfill

sm3_hash_nonfill	变量名 = sm3_hash_nonfill(数据)
	实现的是sm3_256功能，返回32字节数据

sm3_hash_lastblock

sm3_hash_lastblock	sm3_hash_lastblock(十进制前面数据长度，非填充hash值，最后一包数据)
	实现的是sm3_256功能，返回32字节数据

sm4_encode_ecb

sm4_encode_ecb	变量名 = sm4_encode_ecb(数据，密钥)

sm4_decode_ecb

sm4_decode_ecb	变量名 = sm4_decode_ecb(数据，密钥)

sm4_encode_cbc

sm4_encode_cbc	变量名 = sm4_encode_cbc(随机数，数据，密钥)

sm4_decode_cbc

sm4_decode_cbc	变量名 = sm4_decode_cbc(随机数，数据，密钥)

sm4_mac

sm4_mac	变量名 = sm4_mac(随机数，数据，密钥)
	本函数不进行80对齐，见des_des_mac

系统参数部分

apdutime

apdutime	变量名 = apdutime()
	取得系统以微秒数为单位的apdu收发时间

timer

timer	变量名 = timer()
	取得系统以毫秒数为单位的时间

pop

pop	变量名 = pop()
	在call 过程时，可以附带参数，在过程内可以用pop函数逆序弹出。 如果相应数据不存在，则返回数据为空

getpara

getpara	变量名 = getpara()
	在call 过程时，可以附带参数，在过程内可以使用getpara正序读出。 如果相应数据不存在，则返回数据为空

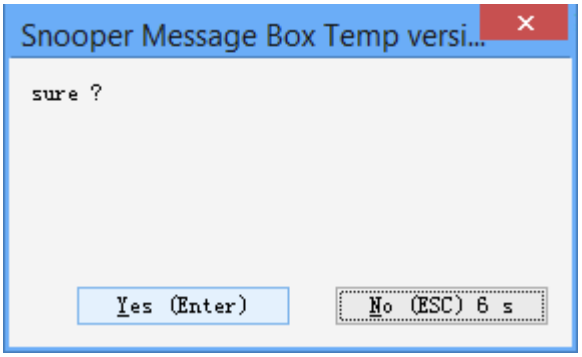
getprotocol

getprotocol	变量名 = getprotocol()
	T0下返回00，非T0下返回01
get_protocol	变量名 = get_protocol()
	T0下返回00，非T0下返回01

guess_protocol

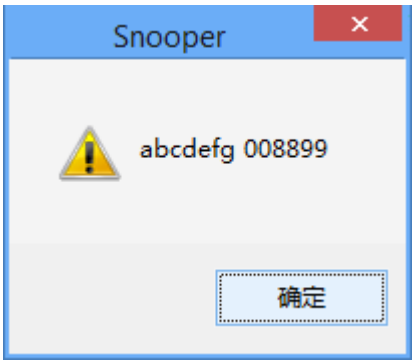
guess_protocol	变量名 = guess_protocol()
	接触式下返回00，非接触下返回01，按读卡器名称猜测出来的结果。

yes_no

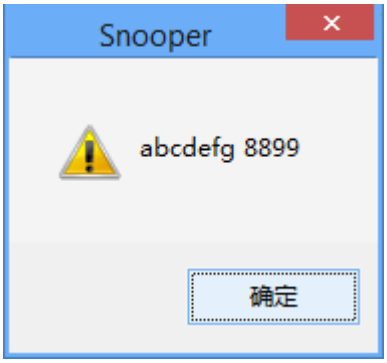
yes_no	变量名 =yes_no(“提示字符串”,1默认yes_0默认no,10进制自动选择时间)
	10进制默认值，0表示默认No，1表示默认选择Yes 10进制自动选择时间，设置了超时秒数之后，倒计时自动返回默认的值。 空格选择当前有焦点的按钮，yes被选中返回1，no被选中返回0 ESC键返回 0，回车选择 1。 

messagebox

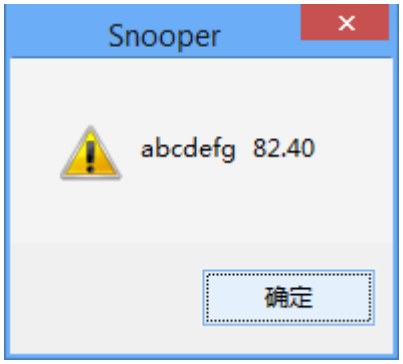
messagebox	变量名 = messagebox(“提示字符串”)
	弹出一个消息框，提示特定操作 a = messagebox("abcdefg " 008899)

messagebox_number

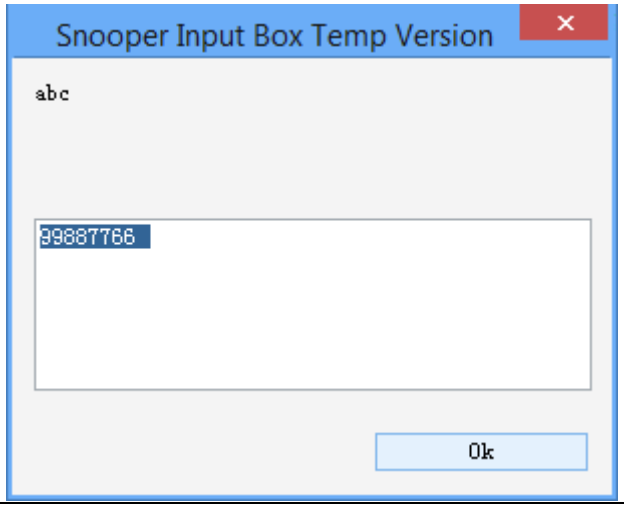
messagebox_number	变量名 = messagebox_number(“提示字符串”)
	弹出一个消息框，提示特定操作，当有00在某数据开始时，会去掉00 a = messagebox_number("abcdefg " 008899) 

messagebox_amount_hex

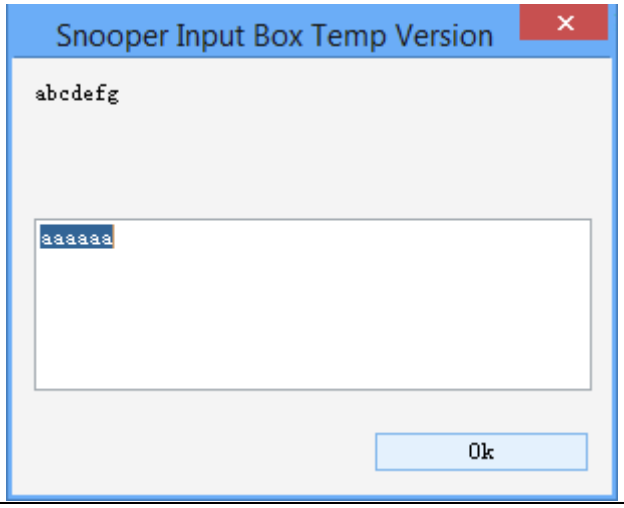
messagebox_amount_hex	变量名 = messagebox_amount_hex(“提示字符串”, 16进制数据)
	弹出一个消息框，提示特定操作，当有00在某数据开始时，会去掉00 a = messagebox_amount_hex("abcdefg ", 2030) 

getinput

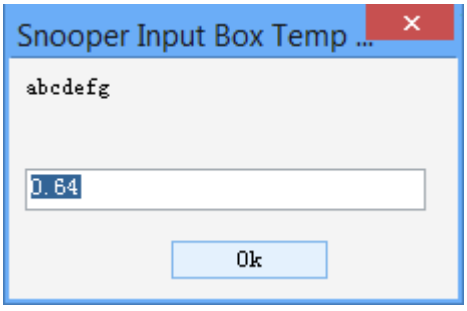
getinput	变量名 = getinput (“提示字符串”, 16进制默认值)	

	16进制默认值 	

getinput_utf16_string

getinput_utf16_string	变量名 = getinput_utf16_string（“提示字符串”， “默认字符串”）	
	目前参数2只支持字符串格式，返回结果是utf16 little endian格式。 	

getinput_amount_hex

getinput_amount_hex	变量名 = getinput_amount_hex（“提示字符串”， 16进制数据_会被显示成10进制带小数点格式）	
		

dummy

dummy	变量名 = dummy(下一版本函数表达式)

	下一版本函数表达式，很多都与现在这一套相同，只支持16进制，为兼容10进制数，可以用int(10进制数)来代替16进制数。

get_first_device_type

get_first_device_type	变量名 = get_first_device_type()
	用以取得第一设备的类型，个别函数依赖于设备，提供此接口可供脚本判断 部分设备类型

get_second_device_type

get_second_device_type	变量名 = get_second_device_type()
	用以取得第二设备的类型，个别函数依赖于设备，提供此接口可供脚本判断 部分设备类型

基本不用部分

fileread

fileread	变量名 = fileread(“文件名”, 16进制长度, 16进制偏移)

epass3003_checksum

epass3003_checksum	变量名 = ePass3003_Checksum(“文件名”)

rockey6_crc32

rockey6_crc32	变量名 = rockey6_crc32(数据, 异或串)
	实现rockey6的带crc32的功能。

leftpack

leftpack	变量名 = leftpack(数据, 10进制长度)
	leftpack(数据, 10进制长度)，返回数据长度等于10进制的数据长度，原文不足时，左边补00

not_morethen

not_morethen	变量名 = not_morethen（16进制数据1, 16进制数据2）
	not_morethen（16进制数据1, 16进制数据2）如果数据1大于数据2，那么返回数据2，否则返回数据1

not_lessthen

not_lessthen	变量名 = not_lessthen（16进制数据1, 16进制数据2）
	not_lessthen（16进制数据1, 16进制数据2）如果数据1小于数据2，那么返回

	数据2，否则返回数据1

chenqi_crc_file

chenqi_crc_file	变量名 = chenqi_crc_file(16进制两字节icv，文件名)

chenqi_crc_mem

chenqi_crc_mem	变量名 = chenqi_crc_mem(16进制两字节icv，数据)

connectless_crc_ab_file

connectless_crc_ab_file	变量名 = connectless_crc_ab_file(16进制两字节icv，文件名)

deletefile

deletefile	变量名 = deletefile(“xxx.txt”)
	删除脚本路径下的xxx.txt

copyfile

copyfile	变量名 = copyfile(“..\app\ccc.txt”, “xxx.txt”)
	拷贝 ..\app\ccc.txt 到本路径下，并命名为xxx.txt 目前只支持 src 相对路径， dst 纯文件名。

calc_ftsafe_serial

calc_ftsafe_serial	变量名 = calc_ftsafe_serial(7字节时间, 2字节序列号, 1字节生产序列号)
	结果为8字节

odd

odd	变量名 = odd(输入数据)
	01 24 56 7A BD EF 38 9C 奇调整为 01 25 57 7A BC EF 38 9D

even

even	变量名 = even(输入数据)
	01 24 56 7A BD EF 38 9C 偶调整为 00 24 56 7B BD EE 39 9C

file_readall (readallfile)

file_readall	变量名 = file_readall(“文件名”)
	将整个文件直接读进来并作为一个结果返回给变量

file_length(filelength)

file_length	变量名 = file_length(“文件名”)
	取得4字节16进制的文件的长度，如果文件不存在，返回00000000。

file_append(appendfile)

file_append	变量名 = file_append(“文件名”, 16进制数)
	写文件，如果写入00，由文件内容是一个字节的0，如果写入20，文件内容是一个字节的空格。继续向后增加 如果文件名本身是变量，要求是ansi格式的 注意： 1） 如果在参数1处直接使用文件名，可以使用”c:\a.txt”，或”kkk.txt”之类 2） 如果在参数1处使用变量，要特别注意’\’的转义功能，建议参考如下示例 clear log_file_name = datetime() log_file_name = num2txt($log_file_name) log_file_name = "d:\\" $log_file_name ".txt" aa = random(1) if $aa != 00 call append_random_error($__LINE__, "随机数不正确") pause endif end append_random_error: error = pop current_line = pop() current_line = num2txt($current_line) s = "文件 " $__FILE__ " " $current_line " 行 " a = file_append($log_file_name, "\r\n") a = file_append($log_file_name, $s) a = file_append($log_file_name, $error) return

file_write(writefile)

writefile	变量名 = file_write(“文件名”, 16进制偏移量, 16进制数)
	写文件，如果写入00，由文件内容是一个字节的0，如果写入20，文件内容是

	一个字节的空格。

num2txt

num2txt	变量名 = num2txt(输入数据)
	输入是00，输出是”00”，也就是3030，无空格。

txt2num

txt2num	变量名 = txt2num (输入数据)
	输入是”00”， 也就是3030，输出是00

reverse_byte_nibble

reverse_byte_nibble	变量名 = reverse_byte_nibble (输入数据)
	输入是1234，输出是2143

direct_write_hid_64x

direct_write_hid_64x	变量名 = direct_write_hid_64x (64字节倍数的数据)
	返回00写成功，返回非00失败

direct_read_hid_64x

direct_read_hid_64x	变量名 = direct_read_hid_64x ()
	返回数据长度固定为64字节，不由脚本指定

readcd

readcd	变量名 = readcd(4字节16进制块号)
	返回数据长度固定为2048字节，不由脚本指定 注：配套前缀关键字 writecd writecd2

readcd_fc

readcd_fc	变量名 = readcd_fc(4字节16进制块号)
	返回数据长度固定为2048字节，不由脚本指定 注：配套前缀关键字 writecd_fc

其他部分

comset

comset	变量名 = comset(“字符串”)
	例如 comset(“com1, 115200, 8, n, 1”)

comsend

comsend	变量名 = comset(16进制数据，10进制输出长度)
	例如 comsend(11223344, 8) 或comsend(112233445566, 0)

	或comsend(11223344)

xorsum

xorsum	变量名 = xorsum(16进制数据)
	对输入数据进行依次xor计算，返回数据长度固定为1字节。

addsum

addsum	变量名 = addsum(16进制数据)
	对输入数据进行依次add计算，返回数据长度固定为4字节。

comtimeout

comtimeout	变量名 = comtimeout(10进制毫秒数)
	aa = comset(“com1, 9600, 8, n, 1”) bb = comtimeout(4000) cc = comsend(11223344, 08)　　// 4秒超时

reset

reset	变量名 = reset()

	用来判断复位是否成功，不会弹出窗口
	aa = reset()
	if $aa == 00
	　? “成功”
	　reset
	　set resp
	else
	　? “失败”
	　pause
	endif

drawimage

drawimage	变量名 = drawimage(utf16 小端格式字符串)
	aa = drawimage(“x 100 y 110 o 100”) 注：在此函数中直接使用字符串，就是utf16 小端格式字符串，不用做其他转换，如果使用$ xxx 方式，要保证 xxx 的内容是合法格式。 示例: 资质认证机构 = drawimage("x 105 y 105 c ff0000 o 102 o 101 o 100 o 99 o 98 f 35 x 41 y 41 i 130 s \"银\" x 25 y 63 i 112 s \" \" x 19 y 75 i 103 s \" \" x 16 y 89 i 95 s \" \" x 15 y 102 i 86 s \"行\" x 18 y 129 i 69 s \" \" x 23 y 142 i 60 s \" \" x 29 y 154 i 51 s \" \" x 38 y 165 i 42 s \"卡\" x 59 y 182 i 25 s \" \" x 71 y 188 i 16 s \" \" x 84 y 192 i 8 s \" \" x 98 y 194 i 0 s \"检\" x 125 y 192 i 342 s \" \" x 138 y 188 i 334 s \" \" x 150 y 182 i 325 s \" \" x 161 y 174 i 316 s \"测\" x 180 y 154 i 299 s \" \" x 186 y 142 i 290 s \" \" x 191 y

	129 i 281 s \" \" x 194 y 116 i 273 s \"中\" x 193 y 89 i 254 s \" \" x 190 y 75 i 246 s \" \" x 184 y 63 i 237 s \" \" x 177 y 51 i 228 s \"心\" - 105 135 87 80 - 76 114 122 80 - 87 80 133 114 - 122 80 105 134 - 133 114 76 114 ")

drawqrcode

drawqrcode	变量名 = drawqrcode(ansi字符串)
	aa = drawqrcode(“x 100 y 110 o 100”) 注：在此函数中直接使用字符串，就是ansi格式字符串，不用做其他转换，如果使用$ xxx 方式，要保证 xxx 的内容是合法格式。 示例: 资质认证机构 = drawqrcode("abcdefg")

scanqrcode

scanqrcode	变量名 = scanqrcode(二维码图片文件名)
	aa = scanqrcode(“snooper.png”) 注：在此函数中在脚本路径中查找二维码图片 示例: x = drawqrcode("snooper.png")

open_second_reader(openreader)

open_second_reader	变量名 = open_second_reader(16进制从1开始pcsc读卡器序列号)

	在脚本中打开第二个读卡器，参数是读卡器列表中的从1开始的索引号
	
	aa = open_second_reader(03) // 使用读卡器列表中第三个读卡器
	注：返回00，成功，其他值失败

open_second_reader_by_name(openreader_by_name)

open_second_reader_by_name	变量名 = open_second_reader_by_name(“读卡器的名称的全部或部分”)
	在脚本中打开第二个读卡器，参数是读卡器名称的一部分 aa = open_second_reader_by_name("sdi011g contactless") // 使用第一个设备名称中出现sdi011g contactless的读卡器。 注：返回00，成功，其他值失败 注：使用读卡器名称打开读卡器，可靠性较低。

send_odd_byte

send_odd_byte	变量名 = send_odd_byte(数据)
	aa = send_odd_byte(aabbcc) 注：目前无返回值

socket_connect

socket_connect	变量名 = socket_connect(字符串格式的地址)
	aa = socket_connect(“192.168.0.1:9999”) 返回0成功 非0失败

socket_io

socket_io	变量名 = socket_io(数据)

	aa = socket_io(11223344)
	
	//a = socket_connect("127.0.0.1:9999")
	a = socket_connect("192.168.15.15:5000")
	//a = socket_connect("192.168.0.1:9999")
	if $a != 00
	pause
	endif
	
	// b是从服务器端接收到的结果
	b = socket_io("12345678")
	
	c = socket_close()

socket_close

socket_close	变量名 = socket_close()
	aa = socket_close()

file_read_as_var

file_read_as_var	变量名 = file_read_as_var(“文件名”)
	aa = file_read_as_var(“a.txt”) 假设a.txt的内容为 卡片序列号 ： 86000021500099999901 ASN号：0021500099999901 卡号：2150999999999901 MF主控密钥 : 29DF2B5E2B9AF20A7F670332CEDA7D52

	MF维护密钥 : BCE534B4966E72C0EA88679FEB37BAEC
	小额应用主控密钥 : E96A50C64990E545C82FD8C0279DAEC9
	小额应用维护密钥 : DDDBA6115B3D585BD12118EBDDDD4BEF
	则会自动变量
	//卡片序列号 = 86000021500099999902
	//asn号 = 0021500099999902
	//卡号 = 2150999999999902
	//mf主控密钥 = 155890A1BE84400787247206A16D7175
	//mf维护密钥 = 7B98C8DA5DD3B384063B9CD4FCFF11BC
	//小额应用主控密钥 = 8C9E4BC4D66D9DA9D3330D48369A753E
	//小额应用维护密钥 = 4D0D41F79BCF2378304A75E48D27D242
	//小额应用圈存密钥01 = BAE31BF15C6F2579EADE9A8FB069869A

M1 部分（使用专用的读卡器—RF1201 读卡器）

keya_or_keyb 为 00，01，02 时表示 KEYA
其他值表示 KEYB

tk_loadkey

tk_loadkey	变量名 = tk_loadkey(keya_or_keyb, secno_00_to_0f, key)
	例如 tk_loadkey(00, 00, ffffffffffff) 返回01表示成功，00表示失败

tk_beep

tk_beep	变量名 = tk_beep()
	固定返回1字节00

tk_authcard

tk_authcard	变量名 = tk_authcard(keya_or_keyb, secno_00_to_0f)
	返回01表示成功，00表示失败

tk_readblock

tk_readblock	变量名 = tk_readblock(address_00_to_3f, keya_or_keyb)
	成功返回16字节数据 失败返回值为空

tk_writeblock

tk_writeblock	变量名 = tk_writeblock(address_00_to_3f, keya_or_keyb, data)
	返回01表示成功，00表示失败

tk_readvalue

tk_readvalue	变量名 = tk_readvalue(address_00_to_3f, keya_or_keyb)
	成功返回4字节数据值 失败返回值为空

tk_writevalue

tk_writevalue	变量名 = tk_writevalue(address_00_to_3f, keya_or_keyb, value)
	返回01表示成功，00表示失败

tk_increment

tk_increment	变量名 = tk_increment(address_00_to_3f, value)
	返回01表示成功，00表示失败

tk_decrement

tk_decrement	变量名 = tk_decrement(address_00_to_3f, value)
	返回01表示成功，00表示失败

tk_transfer

tk_transfer	变量名 = tk_transfer(address_00_to_3f)
	返回01表示成功，00表示失败

tk_restore

tk_restore	变量名 = tk_restore(address_00_to_3f)
	返回01表示成功，00表示失败

M1 部分（使用专用的读卡器—RK501 读卡器）

keya_or_keyb 为 00，01，02 时表示 KEYA
其他值表示 KEYB

rk501_loadkey

rk501_loadkey	变量名 = rk501_loadkey(keya_or_keyb, secno_00_to_0f, key)
	例如 rk501_loadkey(00, 00, fffffffffffff) 返回01表示成功，00表示失败

rk501_beep

rk501_beep	变量名 = rk501_beep()
	固定返回1字节00

rk501_authcard

rk501_authcard	变量名 = rk501_authcard(keya_or_keyb, secno_00_to_0f)
	返回01表示成功，00表示失败

rk501_readblock

rk501_readblock	变量名 = rk501_readblock(address_00_to_3f, keya_or_keyb)
	成功返回16字节数据 失败返回值为空

rk501_writeblock

rk501_writeblock	变量名 = rk501_writeblock(address_00_to_3f, keya_or_keyb, data)
	返回01表示成功，00表示失败

rk501_readvalue

tk_readvalue	变量名 = tk_readvalue(address_00_to_3f, keya_or_keyb)
	成功返回4字节数据值 失败返回值为空

rk501_writevalue

rk501_writevalue	变量名 = rk501_writevalue(address_00_to_3f, keya_or_keyb, value)
	返回01表示成功，00表示失败

rk501_increment

rk501_increment	变量名 = rk501_increment(address_00_to_3f, value)
	返回01表示成功，00表示失败

rk501_decrement

rk501_decrement	变量名 = rk501_decrement(address_00_to_3f, value)
	返回01表示成功，00表示失败

rk501_transfer

rk501_transfer	变量名 = rk501_transfer(address_00_to_3f)
	返回01表示成功，00表示失败

rk501_restore

rk501_restore	变量名 = rk501_restore(address_00_to_3f)
	返回01表示成功，00表示失败

0.0.5.2 新增加部分

setup_m1_accessbit

setup_m1_accessbit	变量名 = setup_m1_accessbit(value)
	设置或分析m1权限位

8583_mac_ecb

8583_mac_ecb	变量名 = 8583_mac_ecb(data, key)
	8583 mac ecb计算

asn1c_der_encode

asn1c_der_encode	变量名 = asn1c_der_encode(r, s)
	返回der加密后的数据，长度由r的长度指定

asn1c_der_decode

asn1c_der_decode	变量名 = asn1c_der_decode(r_s_der, 16进制r或s的长度)

	返回der解密后的数据，前面是r，后面是s
	
	示例:
	r = random(32)
	s = random(32)
	
	der = asn1c_der_encode($r, $s)
	
	tmp = asn1c_der_decode($der, 20)
	
	if $tmp != $r $s
	?
	pause
	endif
	
	//r = 85876F232E45499513253F33FF5416B88C94DF06E9EAD775DB968B6B0248402D
	//s = 5E9191F412985A44670ED579136403A3BC3B4E93B81C6C3617698622E3939A6F
	//der = 304502210085876F232E45499513253F33FF5416B88C94DF06E9EAD775DB968B6B0248402D02205E9191F412985A44670ED579136403A3BC3B4E93B81C6C3617698622E3939A6F
	//tmp = 85876F232E45499513253F33FF5416B88C94DF06E9EAD775DB968B6B0248402D5E9191F412985A44670ED579136403A3BC3B4E93B81C6C3617698622E3939A6F

open_first_reader

open_first_reader	变量名 = open_first_reader(16进制从01开始的索引)
	输入01表示打开读卡器列表中的第一个读卡器

open_first_reader_by_name

open_first_reader_by_name	变量名 = open_first_reader_by_name(“读卡器名称”)
	打开第一个匹配的读卡器，输入参数可以比列表中的名称短，比如列表中存

	在 “ccid 0” 只需要输入”ccid”即可

open_second_reader

open_second_reader	变量名 = open_second_reader(16进制从01开始的索引)
	输入01表示打开读卡器列表中的第一个读卡器 第二读卡器复位使用 reset2 发送apdu使用 apdu2 **注：此函数代替原来的openreader**

open_second_reader_by_name

open_second_reader_by_name	变量名 = open_second_reader_by_name(“读卡器名称”)
	打开第一个匹配的读卡器，输入参数可以比列表中的名称短，比如列表中存在 “ccid 0” 只需要输入”ccid”即可 **注：此函数代替原来的openreader_by_name**

示例：

```
clear

// 索引方式
// 打开读卡器列表中第一个读卡器
a = open_first_reader( 01 )
if $a != 00
    ?
    ? "脚本中打开第一个读卡器失败"
    pause
endif

reset

00 a4 04 00 00
if sw != 9000
    pause
endif

a = messagebox( "换到第二个读卡器上面" )
a = open_first_reader( 02 )
```

```
if $a != 00
    ?
    ? "脚本中打开第一读卡器失败"
    pause
endif

reset

00 a4 04 00 00
if sw != 9000
    pause
endif

a = messagebox( "换到 Identiv uTrust 4700 F CL Reader 读卡器上面" )

// 读卡器名称方式
a = open_first_reader_by_name( "Identiv uTrust 4700 F CL Reader" )
if $a != 00
    ?
    ? "脚本中打开 Identiv uTrust 4700 F CL Reader 读卡器失败"
    pause
endif

reset

00 a4 04 00 00
if sw != 9000
    pause
endif

a = messagebox( "换到 Identiv uTrust 4700 F Contact Reader 读卡器上面" )

// 读卡器名称方式
a = open_first_reader_by_name( "Identiv uTrust 4700 F Contact Reader" )
if $a != 00
    ?
    ? "脚本中打开 Identiv uTrust 4700 F Contact Reader  读卡器失败"
    pause
endif

reset

00 a4 04 00 00
```

```
if sw != 9000
    pause
endif

// 双读卡器演示

a = open_first_reader_by_name( "Identiv uTrust 4700 F Contact Reader" )

reset
00 a4 04 00 00
if sw != 9000
    pause
endif

a = open_second_reader_by_name( "Phoenix reader   9600 BPS COM4" )
reset2

apdu2 00 a4 04 00 00
if sw != 9000
    pause
endif
```

calc_cdk_by_kmc

calc_cdk_by_kmc	变量名 = cakc_cdk_by_kmc(initial_update_返回数据, kmc)
	返回key enc + mac + dek，共0x30字节

示例：

```
reset

00 a4 04 00 00

key = 40414243444546474849 4a4b4c4d4e4f

80 50 00 00 08 1122334455667788
set resp
a = calc_cdk_by_kmc( $resp , $key )

enc = mid( $a, 0, 16 )
mac = mid( $a, 16, 16 )
dek = mid( $a, 32, 16 )
```

```
jcop22_ext_auth 0, $enc, $mac, $dek
```

0.0.5.3 新增加部分

此版本函数，签名，验签，返回 00 表示成功，非 0 值失败。

new_ecc_initialize(p, a, b, Gx, Gy, n, h, hex_len_in_byte)

```
p = fffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
a = 0000000000000000000000000000000000000000000000000000000000000000
b = 0000000000000000000000000000000000000000000000000000000000000007
Gx = 79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
Gy = 483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
n = fffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
h = 01
hex_len_in_byte = 20

r = new_ecc_initialize( $p, $a, $b, $gx, $gy, $n, $h, $hex_len_in_byte )
```

new_ecc_generate_keypair()

new_ecc_check_pubkey(pubkey)

```
keypair = new_ecc_generate_keypair()
len = datalen( $keypair )
if $len > 01
    pri = mid( $keypair, 0, 32 )
    pub = mid( $keypair, 32 )
else
    // error
    pause
endif
r = new_ecc_check_pubkey( $pub )
```

new_ecc_get_pubkey(prikey)

```
pub1 = new_ecc_get_pubkey( $pri )
if $pub != $pub1
    ?
```

```
    pause
endif
```

new_ecc_ecdh_agreement(prikey, other_pubkey)

```
pri1 = random( 32 )
pri2 = random( 32 )
pub1 = new_ecc_get_pubkey( $pri1 )
pub2 = new_ecc_get_pubkey( $pri2 )

x = new_ecc_ecdh_agreement( $pri1, $pub2 )
```

new_ecc_sign(prikey, hash, specific_random)

new_ecc_verify(pubkey, hash, r_and_s)

```
hash = random( 32 )
r_s = new_ecc_sign( $pri1, $hash )

kk = new_ecc_verify( $pub1, $hash, $r_s )
```

new_ecc_point_add(p, q)

new_ecc_point_double(p)

new_ecc_j2a(p)

new_ecc_kp(k, p)

new_ecc_kp_add_lq(k, p, l, q)

new_ecc_check_point(point)

new_ecc_computer_y(pc_02_or_03, x)

```
a = new_ecc_point_add( $pub1, $pub2 )

x = new_ecc_check_pubkey( $a )
```

```
a = new_ecc_point_double( $pub1 )
x = new_ecc_check_pubkey( $a )

a = new_ecc_j2a( $pub1 )

x = random( 32 )
a = new_ecc_kp( $x, $pub1 )
x = new_ecc_check_pubkey( $a )

l = random( 32 )
k = random( 32 )
a = new_ecc_kp_add_lq( $k, $pub1, $l, $pub2 )
x = new_ecc_check_point( $a )

x = mid( $pub1, 0, 32 )
y = new_ecc_computer_y( 02, $x )
```

new_ecc_ecdh_gm_map(k, h)

执行 K*G+h 操作

big_add(a, b)

big_sub(a, b)

big_mul(a, b)

big_mod_add(a, b, n)

big_mod_sub(a, b, n)

big_mod_mul(a, b, n)

big_mod_mul_montgomery(a, b, n) // 目前要求 n 的长度是 4 字节的倍数

big_mod_inv(e, n)

big_mod_inv_2_32(4_byte_e)

big_mod_exp(m, d)

big_exp(a, b)

big_bcd(a, b)

prime_test(prime)

prime_gen(byte_len_in_hex_format)

prime_gen_multi(byte_len_in_hex_format, thread_num)

prime_gen_bit(bit_len_in_hex_format)

prime_gen_bit_multi(bit_len_in_hex_format, thread_num)

getnextprime(current_number)

getprevprime(current_number)

```
clear

a = 058980
b = 0a
c = big_mul( $a, $b )

c = 01
hfor l = 01 to 10
    c = big_mul( $c, $l )
```

```
hnext 1

d = expr10( 0x$c )

// a = big_mod_mul( $a, $b, $n )

k = mul( 8899, aabb )
k = big_mul( 8899, aabb )

n =
B305E4AD9B9D923E9ED2F72B3C419AB2759B1E14F11C3897DF91733C056E24C68848D79CD4DA1B4F5B82FC5DD5E1
44A6DF6BBEBC0353704C046CB5DD782959C2CC0D91473F9F87C2C46664EA16A82B0B2959BE6F2DCAB7A34668789B
856395D23F3DEB7ECBDECEF454313A712C52FCA4603C4406CA3A5F7A74DFFFB0655B5E8BF011B4DCBD9D64694150
EE59C1CA1AE2C22792BDFA2301E19126CEF34B8AFAB6260438A6A215D753E951DAECE82D1E17DFC28CA357D689BB
2432B06FF25489AA9DCEE3CECC4304353CE2E2832079990BE045B1DD7FEE7396C3E91B4DC7027996F17DD4642793
BD4ECC833E51169C0ABC88A52DD2B530DA2AE960578E2F2AA0F1

c =
AF2B763EA10A137BFAB9FAA326575E33C75388970D84482BD30161B51DAFA810066E93853016A7F8EB84FA33423D
F23269B0C14C3BF70E5ADD6201359F70739DD273C43F7B83603B073FF93F338A1F09A2843D8C1F4DADBE0F49FBA0
DA44D4C11629F5D2CB2CE2AE88424B37862E5BAE0049B3AF940411467CDD38EACBB3F54043EB69C53884FE6758A3
4781382E37ABE9DF01C19D54F4C34C5D7F976B551A66C00D296A91FCA2931E144F48A66771AFC72D3DC3F20FAB07
2B00A89058F577DC2750E24F5F5D227D703B2D7810955CD744BC9B04055B3DB2C70533A0D0E26B1FA65751937748
A766613E98D81667BEE995B8D34A7CCC9DF7C7348647B3E0415D

d =
633F9828BB461F26C24C79252A11C0DEC11E8F0DD321EF2A1B92FA8CC301EE3776A4D38C29E10514F8C9E4796D32
A729D0615E6BC37845A08F2F699FEF9715214E9912FAAA061A70DB0B5D54FDDC9B886393526536C0D5101ADF9E45
B22CA489DBEFCB180F9465ADC4E349E0264F97AAD58373BF43EC1AB4BC08D72B06F621B28AEC74D80CCF9BF7633F
CE0C5647D15F7423767FA12CDB0552E8673D5F72283106DB7BA0003B0A68492B6826D4ABA08DB269767432E7923C
9E957382F941DD3AD442BBC31D24EDEFD4BA7AD49E618B69CA23430121D0C99DB74A46906A60CD1022364D50EC25
D68C71F6C84C40E00EBDE6BF879BB0B36826BC080EEDBA236E65

// c + d mod n:
res = big_mod_add( $c, $d, $n )
if $res !=
5F6529B9C0B2A0641E337C9D1427846012D6F98FEF89FEBE0F02E905DB437180F4CA8F74851D91BE88CBE24ED98F
54B55AA660FBFC1BE3AF6824B4F816DE2EFC54FF45F2E5E9F2E91DE4F1AA1ABE8F86DCBDD1822843CB2AE3C1214B
070DE378B2DBD56C0EE27967F8F45AA6802AF6B47590E3680DB5CC80C40610656D4EB866DEC629C087B735F57A92
2733CCABEE289BDAE583445ECDE70E1F17E17F3C47E1A0E46C63F021D5A77DEDDC8292E5F42599D42794CD20B388
A5636BA35FE2CB6C5DC4BA43B03F0C380812C5C98E7D4F352E9A2C27A73D93B9BA665EE37440BE98D70FCA803BDA
C0A406B222D340ABC2EAF3D32D13784F2BF399DC3DA73ED90ED1
    ?
```

```
    pause
endif

// d - c mod n:
res = big_mod_sub( $d, $c, $n )
if $res !=
671A0697B5D99DE9666575AD3FFBFD5D6F66248BB6B9DF9628230C13AAC06AEDF87F17A3CEA4786B68C7E6A400D5
F99E461C5BDB8AD4A791B63A1E47C84FFB464832E0026E2241F89831C8FFE0FAA789EA68D348453DDEF551FE1B40
5D4B659B0503C0C4104651F390D23919CC7438A1357604167A2268E8B40B9DF0A09D8AFE3712BFEF91E801F94BED
74E4DFE3B4964C6C077BFDFAE82397B1B6993FA808806CD28ADC10543F291468F3CB16714CF5CAFEC55398AE70F0
97C77B6292A0EF094AC0BD428A0ACFA7A1622FDFAE45C79E65AC59DA9C63FF81B42E2E3D6080DB876D5CD0219C70
EC74DD3B6DC541145A90D9ABE223E917A459DE33E034356DCDF9
    ?
    pause
endif


// c * d mod n:
res = big_mod_mul( $c, $d, $n )
if $res !=
7431283183E453A86B561B6679C165873C622FDAD0F50A6BF64379AFE2E52359D4EC366F48059EF06CC626F00E28
61E0E65F1E12AB10297AF14DF49F75D82FAC1DEAB2B04787E6716F12F2DD9D2697DF63F668638EBF0E2A9CA3A707
02B90EEEE720F3579E71BBF7A4C668583DB5269769D4F5EF42DB5A59958E44C3F5F98AAAA8CA516BDD102AFCEAB0
CD16E16B4D6339ED64B987C6A79813D9C14C5BA1DE06EA514FE95287D53BD804B92E6362E545990F0319929C1102
3814E6B6E99C4A5F4894D35012840996D5FA634A0B069A800A70C104E7E21EFF35EA87CC5A8D150B7B38773C6DFC
AF329A2B27AD3513CDCC85F6A738486482FE9243FF1BFC08B716
    ?
    pause
endif


// c ^ d mod n:
res = big_mod_exp( $c, $d, $n )
if $res !=
90819B56518680B50B8F555F38B4A36404A54BB3B6D84A14D4E3FB3BC879B6080E450B585E24009537F1B2C22EA4
E589DD182B217C558E86F2A5C301BA4A1E3CFAF835FD647F5D4D35015B611BE101F403FA70297E7245539F52DEBB
78DD0F685EE3EFE6A4C2ADD0E28AD7D16EFA390E19E8CD332498B0A35B202AC9CC542E53B80F065BA53A7B9EFD13
62531F30C45855BEF7519D41736314A0DB783F5E6F6C48AC5FE3D5958D81F16EB95B8227BAA2CC52CF9480FB32DC
9F819A1315C7392AEC4BF4ADF9DAE982388FC15D9F1961206DBB21AF9134B8A0403F01E29E35E24AC93DA8754623
73ADDB7D24EB3D662D4DDD0C3CB9C1CBDD957FA2EF67ED374253
    ?
    pause
endif
```

```
e = 00010001

p =
C5594D6F383F7D5B781B28773776065A506E1028474D5C7F153A577F107F237E117F7C2B4A560E39096D684E066D
053F1B7F5A76383F107F2F6D6D3D3E7F5E78697F385D761B463B4375611B1F5F083F244E423F043D2A7E263E5F1F
442B536F6C7D5C3E315B4D3E2864284F02753D773423116D71352D576E7D775F797D65AA

// modinv(e, p):
res = big_mod_inv( $e, $p )
if $res !=
40152488E609EFC9E0F168AD75ABF49A251DE84F3C0009F6164D9095E9246FC1032B93857729048B4EDD610E5D15
CFAB5361825D7B1C375E2567721E7E09DC025BB77CF51461FCE7479B7901D6A9044384A588EB6F93F1156F93B74D
AED426CEBABAE65627AECF214CFA61006EFAE50FAD3569F650D984B4AC633F7D183538EB
    ?
    pause
endif

////////////////////////////////////////////
// 蒙哥马利模乘:
// c * d mod n:
res = big_mod_mul_montgomery( $c, $d, $n )
if $res !=
1E188AA1CF3A711A947461EA621CC7CC3FE1F5ED229753511C525623A28B2E058502355336336D8167835B74726A
9CF5B7E26FD0CD318F17621F61D6642BD8AB76EB75BBFCC1A9023FC5B0148185476AAD34FBCC986BAC54EB35F2CB
238507FEC266B4E49853A6CC860A0B5E111160E050E2F85DE38A1E0B2B315D51ABF644DF69FF5BA2607C3D4D2920
B41BEC1F0E54469B78BD24CC6A0C01B086377D1A5D7E5A2F02B774DD69225A94D0158744EF8BE7DF66DB5D47AFF5
C012E578FE6FF21AF42CFAD9724867E05546728F9E33EB7F6EC7C8FF7072F7318EB79D8821DC8EBD27436A8C771A
46657C31377B6616EDD7A665653388C28E23FF3913F1D8B630A8
    ?
    pause
endif
```

```
e       = prime_gen( 20 )
fn      = random( 42 )
inv     = big_mod_inv( $e, $fn )


ppp     = big_mul( $e, $inv )
qqq     = big_div( $ppp, $e )
ppp1    = big_mul( $qqq, $e )
```

```
compare $ppp, $ppp1
r       = big_mod( $ppp, $fn )
```

new_rsa_genstd(bit_len_in_hex, e)

```
res = new_rsa_genstd( 0400, 010001 )
n = mid( $res, expr10( 64 * 0 ), 128 )
d = mid( $res, expr10( 64 * 2 ), 128 )
```

new_rsa_genstd_multi(bit_len_in_hex, e, thread_num)

```
res = new_rsa_genstd( 0400, 010001, 10 )
n = mid( $res, expr10( 64 * 0 ), 128 )
d = mid( $res, expr10( 64 * 2 ), 128 )
```

new_rsa_gencrt(bit_len_in_hex, e)

```
res = new_rsa_gencrt( 0400, 010001 )
n = mid( $res, expr10( 64 * 0 ), 128 )
d = mid( $res, expr10( 64 * 2 ), 128 )
p = mid( $res, expr10( 64 * 4 ), 64 )
q = mid( $res, expr10( 64 * 5 ), 64 )
dp = mid( $res, expr10( 64 * 6 ), 64 )
dq = mid( $res, expr10( 64 * 7 ), 64 )
qinv = mid( $res, expr10( 64 * 8 ), 64 )
```

new_rsa_gencrt_multi(bit_len_in_hex, e, thread_num)

```
res = new_rsa_gencrt( 0400, 010001, 10 )
n = mid( $res, expr10( 64 * 0 ), 128 )
d = mid( $res, expr10( 64 * 2 ), 128 )
p = mid( $res, expr10( 64 * 4 ), 64 )
q = mid( $res, expr10( 64 * 5 ), 64 )
dp = mid( $res, expr10( 64 * 6 ), 64 )
dq = mid( $res, expr10( 64 * 7 ), 64 )
qinv = mid( $res, expr10( 64 * 8 ), 64 )
```

new_rsa_std_sign(bit_len_in_hex, n, d, plain)

new_rsa_crt_sign(bit_len_in_hex, p, q, dp, dq, qinv, plain)

new_rsa_verify(bit_len_in_hex, e, n, plain, signature)

```
xx = new_rsa_crt_sign( 0400, $p, $q, $dp, $dq, $qinv, $kk )
yy = new_rsa_verify( 0400, 010001, $n, $kk, $xx )

xx = new_rsa_std_sign( 0400, $n, $d, $kk )
yy = new_rsa_verify( 0400, 010001, $n, $kk, $xx )
```

new_rsa_pub_encrypt(bit_len_in_hex, e, n, cipher)

new_rsa_std_decrypt(bit_len_in_hex, n, d, plain)

new_rsa_crt_decrypt(bit_len_in_hex, p, q, dp, dq, qinv, plain)

```
tt = new_rsa_pub_encrypt( 0400, 010001, $n, $kk )

res1 = new_rsa_crt_decrypt( 0400, $p, $q, $dp, $dq, $qinv, $tt )
```

new_rsa_calculate_other_by_pqe(bit_len_in_hex, e, p, q)

new_rsa_calculate_other_by_nde(bit_len_in_hex, e, n, d)

```
e       = prime_gen( hex( 512 / 8 )  )

nd      = new_rsa_genstd( 0400, $e )
nn      = mid( $nd, 0, 128 )
dd      = mid( $nd, 128 )

all     = new_rsa_gencrt( 0400, $e )
len     = 0064
n       = mid( $all, 0, int( $len * 2 ) )
d       = mid( $all, int( $len * 2 ), int( $len * 2 ) )
p       = mid( $all, int( $len * 4 ), $len )
q       = mid( $all, int( $len * 5 ), $len )
dp      = mid( $all, int( $len * 6 ), $len )
dq      = mid( $all, int( $len * 7 ), $len )
```

```
qinv    = mid( $all, int( $len * 8 ), $len )

x       = dummy( clear_fifo() )
xx      = dummy( rsa_calculate_other_by_pqe( 0400, $e, $p, $q ) )
n1      = dummy( read_fifo() )
d1      = dummy( read_fifo() )
dp1     = dummy( read_fifo() )
dq1     = dummy( read_fifo() )
qinv1   = dummy( read_fifo() )

if $n != $n1
    ?
    pause
endif

x       = new_rsa_calculate_other_by_pqe( 0400, $e, $p, $q )
n2      = mid( $x, 0, int( $len * 2 ) )
d2      = mid( $x, int( $len * 2 ), int( $len * 2 ) )
dp2     = mid( $x, int( $len * 4 ), $len )
dq2     = mid( $x, int( $len * 5 ), $len )
qinv2   = mid( $x, int( $len * 6 ), $len )
if $n != $n2
    ?
    pause
endif
```

new_sm2_generate_keypair()

new_sm2_get_pubkey(prikey)

new_sm2_sign(prikey, hash, specific_random)

new_sm2_verify(pubkey, hash, r_and_s)

new_sm2_encrypt_123(pubkey, plain)

new_sm2_decrypt_123(prikey, cipher)

new_sm2_encrypt_132(pubkey, plain)

new_sm2_decrypt_132(prikey, cipher)

new_sm2_getz(pubkey, id)

```
clear

// 生成密钥对
a = new_sm2_generate_keypair()
pri = mid( $a, 0, 32 )
pub = mid( $a, 32 )

// 由私钥生成公钥
pub2 = new_sm2_get_pubkey( $pri )
if $pub != $pub2
    ?
    pause
endif

// 原来的生成公钥
pub3 = sm2_generate_pub_by_pri( $pri )
if $pub != $pub3
    ?
    pause
endif

// 加密成123格式
k = dup( 50, ee )   // random( 50 )
a1 = new_sm2_encrypt_123( $pub, $k )
b1 = new_sm2_decrypt_123( $pri, $a1 )
if $b1 != $k
    ?
```

```
    pause
endif

// 加密成132格式
a2 = new_sm2_encrypt_132( $pub, $k )
b2 = new_sm2_decrypt_132( $pri, $a2 )
if $b2 != $k
    ?
    pause
endif

// 123转132格式
xlen = datalen( $a1 )
c1 = mid( $a1, 0, 64 )
c2 = mid( $a1, 64, expr10( 0x$xlen - 96 ) )
c3 = right( $a1, 32 )
newa1 = $c1 $c3 $c2
newb1 = new_sm2_decrypt_132( $pri, $newa1 )

// 用原来的算法解密
x1 = sm2_pri_decode( $a1, $pri )
x2 = sm2_pri_decode( $a2, $pri )
x3 = sm2_pri_decode( $newa1, $pri )

// 用原来的算法加密
old_cipher = sm2_pub_encode( $k, $pub )
// 用新的算法解密
// new_plain = new_sm2_decrypt_132( $pri, $old_cipher )
new_plain = new_sm2_decrypt_123( $pri, $old_cipher )
if $new_plain != $k
    ?
    pause
endif
new_plain = sm2_pri_decode( $old_cipher, $pri )
if $new_plain != $k
    ?
    pause
endif


// 新函数签名测试
hash = random( 32 )
r_s = new_sm2_sign( $pri, $hash ) // or new_sm2_sign( $pri, $hash, $random )
```

```
r = left( $r_s, 32 )
s = right( $r_s, 32 )

// 旧函数验签
sign_verify_result = sm2_sign_verify( $pub, $hash, $r, $s )
if $sign_verify_result != 01
    ?
    pause
endif

// 旧函数签名测试
r_s = sm2_sign( $hash, $pri )
sign_verify_result = new_sm2_verify( $pub, $hash, $r_s )
if $sign_verify_result != 00
    ?
    pause
endif
```

ripemd128_hash(data)

ripemd160_hash(data)

aes128_mac

aes128_mac	变量名 = aes128_mac(随机数，数据，密钥)
	本函数不进行80对齐，见des_des_mac

aes192_mac

aes192_mac	变量名 = aes192_mac(随机数，数据，密钥)
	本函数不进行80对齐，见des_des_mac

aes256_mac

aes256_mac	变量名 = aes256_mac(随机数，数据，密钥)
	本函数不进行80对齐，见des_des_mac

file 函数族

file_read	变量名 = file_read (“文件名”, 16进制偏移量，16进制长度)
file_write	变量名 = file_write (“文件名”, 16进制偏移量，16进制数据)
file_readall	变量名 = file_readall (“文件名”)
file_writeall	变量名 = file_writeall (“文件名”, 16进制数据)
file_append	变量名 = file_append (“文件名”, 16进制数据)
file_length	变量名 = file_length (“文件名”)

0.0.5.4 新增加部分

加入关键字

jcop22_ext_auth_2
jcop22_ext_auth_kmc_2
mac_auto_mac_2
upload_2
delete_2
select_2
des_des_mac_2
des_3des_mac_2

功能与原来的一样，对第二设备进行操作。

0.0.5.5 新增加部分

关于ZY_RF_Monitor的使用，请见Snooper.Spec-008-ZY_RF_Monitor_使用手册

map_m1_keya

map_m1_keya	变量名 = map_m1_keya(M1密钥keya)

map_m1_keyb

map_m1_keyb	变量名 = map_m1_keyb(M1密钥keyb)
	keya = ffffffffffff keyb = ffffffffffff m1key = call calc_m1_key($keya, $keyb) if $m1key != 0B 54 57 07 45 FE 3A E7 ? pause endif keya = a0a1a2a3a4a5 keyb = b0b1b2b3b4b5 m1key = call calc_m1_key($keya, $keyb) if $m1key != 8C 7F 46 D7 6C E0 12 66 ? pause

```
endif

keya = 4D 3A 99 C3 51 DD
keyb = 1A 98 2C 7E 45 9A

mlkey = call calc_ml_key( $keya, $keyb )
if $mlkey != D8 09 C4 6A 74 84 A1 34
    ?
    pause
endif

end

calc_ml_key:
    keya1 = getpara
    keyb1 = getpara

    dkeya = map_ml_keya( $keya1 )
    dkeyb = map_ml_keyb( $keyb1 )

    key1 = reverse_block_byte( $dkeya )
    key2 = reverse_block_byte( $dkeyb )

    x = 3des_encode_ecb( 0000000000000000, $key1 $key2 )
    x = reverse_block_byte( $x )

return $x
```

reader_dialog

reader_dialog	变量名 = reader_dialog() 变量名 = reader_dialog(读卡器窗口从01开始的16进制索引) 变量名 = reader_dialog(“读卡器名称”) 返回00成功 返回其他值失败
	a = reader_dialog() a = reader_dialog(03) a = reader_dialog("scm")

close_first_device

close_first_device	变量名 = close_first_device() 返回空 注：返回值以后可能被修改
	a = messagebox("请使用接触界面") a = close_first_device() a = reader_dialog()

0.0.5.6 新增加部分

prime_gen

prime_gen	变量名 = prime_gen(16进制素数字节长度)

	prime = prime_gen(40)
	
	res = prime_test($prime)
	if $res != 00
	    ?
	    pause
	endif

	//prime = FE8477E417A6579A5B7266B30E0C5F4F68FE1E701904360F39EA09046A350392008732B16B0201655D1B0ED13A9230BF36911DD1742037A12967506F698363DF //res = 00

big_mod_inv_2_32

big_mod_inv_2_32	变量名 = big_mod_inv_2_32(4字节整数)

	a = 11223397
	b = big_mod_inv_2_32($a)
	c = mul($a, $b)
	c1 = big_mul($a, $b)
	if $c != 00000001
	    ?
	    pause
	endif

	//a = 11223397
	//b = 693B7C27
	//c = 00000001
	//c1 = 070B025A00000001

connectless_crc16_ab

connectless_crc16_ab	变量名 = connectless_crc16_ab(icv, data)
	a = connectless_crc16_ab(6363, e040) if $a != 3db5 ? pause endif a = connectless_crc16_ab(ffff, e040) if $a != daa4 ? pause endif

obe_crc16

obe_crc16	变量名 = obe_crc16(data)
	a = obe_crc16(D4C1423731374E5500000000FFFFFFFF) if $a != 2c0e ? pause endif

unpack80

unpack80	变量名 = unpack80(data) 此函数检查数据格式，去掉以80 00 00结尾的数据，如果不存在类似的尾部数据，则返回原值
	a = unpack80(aabb) a = unpack80(aabb80) a = unpack80(( )80) a = unpack80(00000000) a = unpack80(0000008000)

	//a = AABB
	//a = AABB
	//a = 00
	//a = 00000000
	//a = 000000

pack80_len

pack80_len	变量名 = pack80_len(data, 16进制对齐长度)
	a = pack80_len(1122, 20) b = unpack80($a)

	c = pack80_len(112233, 04) d = pack80_len(11223344, 04)
	此函数按16进制长度进行数据对齐，如果数据本身符合长度要求，则不再添加80 00 00 对齐

pack00_len

pack00_len	变量名 = pack00_len(data, 16进制对齐长度)
	a = pack00_len(1122, 20) c = pack80_len(112233, 04) d = pack80_len(11223344, 04)
	此函数按16进制长度进行数据对齐，如果数据本身符合长度要求，则不再添加00 00 00 对齐

oem_sm4_encrdata

oem_sm4_encrdata	变量名 = oem_sm4_encrdata(key, data, mode)
	将数据前面添加长度，后边补上80，然后用sm4做ecb加密 oem_sm4_encrdata(key, data, mode) 金融，加油数据加密，密钥直接加密/解密，不需要取随机数，不需要计算过程密钥 mode=0，加密 mode=1，解密

oem_sm4mac

oem_sm4mac	变量名 = oem_sm4mac(key, data, rand, mode)
	自动补80，然后按sm4算mac oem_sm4mac(key,data,rand,mode) mode bit 0位为0时，金融，加油计算MAC，当数据长度为8的整数倍时仍然补80，最低位为1时，长度为8的整数倍时不补80. mode bit 1位为0时，产生mac为4字节，bit1为1时，产生mac为8字节。 0 强制补80，4字节mac 1 长度为8时，不自动补80，4字节mac 2 强制补80，8字节mac 3 长度为8时，不自动补80，8字节mac

oem_sm4js

oem_sm4js	变量名 = oem_desjs(key, data, mode)
	sm4做ecb模式计算 mode=00 加密 mode=01 解密

memset

memset	变量名 = memset(dest, offset, value, len)
	value长度可以超过1字节

memcpy

memcpy	变量名 = memset(dest, offset, value)

memxor

memxor	变量名 = memxor(dest, offset, value)

memor

memor	变量名 = memor(dest, offset, value)

memand

memand	变量名 = memand(dest, offset, value)

des_subkey

des_subkey	变量名 = des_subkey(数据，密钥)
	支持多级分散

des_diversify

des_diversify	变量名 = des_diversify(数据，密钥)
	支持多级分散

sm4_subkey

sm4_subkey	变量名 = sm4_subkey(数据，密钥)
	支持多级分散

sm4_diversify

sm4_diversify	变量名 = sm4_diversify(数据，密钥)
	支持多级分散

0.0.5.7 新增加部分

new_sm2_keyexchange

new_sm2_key exchange	变量名 = new_sm2_exchange(角色（00表示发起方，01表示响应方），己方私钥，己方临时私钥，对方公钥，对方临时公钥，己方ID，对方ID)
	a = new_sm2_generate_keypair() self_pri = left($a, 32) self_pub = right($a, 64) a = new_sm2_generate_keypair()

```
self_tmp_pri     = left( $a, 32 )
self_tmp_pub     = right( $a, 64 )

a                = new_sm2_generate_keypair()
other_pri        = left( $a, 32 )
other_pub        = right( $a, 64 )

a                = new_sm2_generate_keypair()
other_tmp_pri    = left( $a, 32 )
other_tmp_pub    = right( $a, 64 )

self_id          = dup( 16, 11 )
other_id         = dup( 16, 22 )

keyab            = new_sm2_keyexchange( 00, $self_pri, $self_tmp_pri, $other_pub,
$other_tmp_pub, $self_id, $other_id )

keyba            = new_sm2_keyexchange( 01, $other_pri, $other_tmp_pri, $self_pub,
$self_tmp_pub, $other_id, $self_id )

if $keyab != $keyba
    ?
    pause
endif

self_pri         =
7D511F0A173101AD4B1E204956946F6C6E1B0A087C9F5AFC405751CD74C87429
self_pub         =
13F17A6A022AFFE04523A1BABE9A65C3BB3B90EDB57404F1D4CC80C048E57BB507EF820388F0771
D2C5AA92DAE53B75732C8A774F838E1095FC2E4E836F7FB29

self_tmp_pri     =
16332BBB07B929915707348063EF6B922A7578C7203D405514CB2B5A6A1F6D93
self_tmp_pub     =
D15AA0A9EB024EE12AF054D6BF25A8887CF6C5CC9910DD535C6E8D2DAD0DD2707A378F38876B376
75B985591E403028FA8E10BD0020F7F27608CA4FDF2A95745

other_pri        =
581C3B64332F12386F874C02278249A0531906B8715F44B16A1F3155230C76CD
other_pub        =
1D9E4517A0483378B59CCD8CD65D5A6F180FA85850102E0CFCE0FD6E27B38C4D7434456860CF929
2B0E29505F7CE8A23031F6997EFBEF90C18FAA1EB596A2F94
```

```
other_tmp_pri    =
2B3B097C799A6F9333BA3D785CC26B37031456B663EB605F0C5233C718E337D8
other_tmp_pub    =
5FFE69C7756B2B14B3CC131A10AB0E0DAD9C5778F89E52CD5C94E354D8F80592ACD3B97849C18FB
FAA38E3456C60C5E7CC5581691681DE57FEDFE1E720165CB2

keyab            = new_sm2_keyexchange( 00, $self_pri, $self_tmp_pri, $other_pub,
$other_tmp_pub, $self_id, $other_id )

keyba            = new_sm2_keyexchange( 01, $other_pri, $other_tmp_pri, $self_pub,
$self_tmp_pub, $other_id, $self_id )

if $keyab != CF41314ECF7C0FEDB67CC92D62C5434C0E33C07264120F85990E5B98EC88B4F5
    ?
    pause
endif

if $keyba != CF41314ECF7C0FEDB67CC92D62C5434C0E33C07264120F85990E5B98EC88B4F5
    ?
    pause
endif
```

otp_sm3

otp_sm3 | 变量名 = otp_sm3(key, t, tc, c, q)

```
//
{0X12,0X34,0X56,0X78,0X90,0XAB,0XCD,0XEF,0X12,0X34,0X56,0X78,0X90,0XAB,0XCD,0XEF},
// 1313655030, 1234, "5678", 854787

key = 1234567890abcdef 1234567890abcdef
t = expr16( 1313655030 )
tc = 01
c = expr16( 1234 )
q = "5678"

a = otp_sm3( $key, $t, $tc, $c, $q )

the_correct_result = expr16( 854787, 4 )
if $a != $the_correct_result
    ?
    pause
endif

//
```

```
{0X12,0X34,0X56,0X78,0X90,0XAB,0XCD,0XEF,0X12,0X34,0X56,0X78,0X90,0XAB,0XCD,0XEF},
// 1313998979, 1234, "5678", 814095
t = expr16( 1313998979 )
tc = 01
c = expr16( 1234 )
q = "5678"

a = otp_sm3( $key, $t, $tc, $c, $q )

the_correct_result = expr16( 814095, 4 )
if $a != $the_correct_result
    ?
    pause
endif
```

otp_sm4

otp_sm4	变量名 = otp_sm4(key, t, tc, c, q)

```
//
{0X12,0X34,0X56,0X78,0X90,0XAB,0XCD,0XEF,0X12,0X34,0X56,0X78,0X90,0XAB,0XCD,0XEF},
// 1340783053, 1234, "5678", 446720

key = 1234567890abcdef 1234567890abcdef
t = expr16( 1340783053 )
tc = 01
c = expr16( 1234 )
q = "5678"

a = otp_sm4( $key, $t, $tc, $c, $q )

the_correct_result = expr16( 446720, 4 )
if $a != $the_correct_result
    ?
    pause
endif

//
{0X12,0X34,0X56,0X78,0X90,0XAB,0XCD,0XEF,0XAB,0XCD,0XEF,0X12,0X34,0X56,0X78,0X90},
// 1340783416, 5621, "3698", 49845
key = 1234567890abcdef abcdef1234567890
t = expr16( 1340783416 )
tc = 01
c = expr16( 5621 )
```

```
q = "3698"

a = otp_sm4( $key, $t, $tc, $c, $q )

the_correct_result = expr16( 49845, 4 )
if $a != $the_correct_result
    ?
    pause
endif
```

pkcs_encrypt_pad

pkcs_encrypt_pad	变量名 = pkcs_encrypt_pad(待加密数据, 16进制模长)
	返回填充后的明文

pkcs_encrypt_unpad

pkcs_encrypt_unpad	变量名 = pkcs_encrypt_unpad(填充后的明文)
	返回去掉填充的原始数据

pkcs_encrypt_pad_ff

pkcs_encrypt_pad_ff	变量名 = pkcs_encrypt_pad_ff(待加密数据, 16进制模长)
	返回填充后的明文，使用ff填充

pkcs_encrypt_unpad_ff

pkcs_encrypt_unpad_ff	变量名 = pkcs_encrypt_unpad_ff(填充后的明文)
	返回去掉填充的原始数据，与pkcs_encrypt_unpad相比，对填充的要求不同

pkcs_sign_pad

pkcs_sign_pad	变量名 = pkcs_sign_pad(待签名数据, 16进制模长)
	返回填充后的明文

pkcs_sign_unpad

pkcs_sign_unpad	变量名 = pkcs_sign_unpad(填充后的明文)
	返回去掉填充的原始数据

show_ansi_string

show_ansi_string	变量名 = show_ansi_string(数据, 转义标识)

show_utf8_string

show_utf8_string	变量名 = show_utf8_string(数据, 转义标识)

show_utf7_string

show_utf7_string	变量名 = show_utf7_string(数据, 转义标识)

show_utf16_little_string

show_utf16_little_string	变量名 = show_utf16_little_string(数据, 转义标识)

show_utf16_big_string

show_utf16_big_string	变量名 = show_utf16_big_string(数据, 转义标识)

show_string

show_string	变量名 = show_string(数据,代码页, 转义标识)

0.0.5.8 新增加部分

new_sm2_ecc_kdf

new_sm2_ecc_kdf	变量名 = new_sm2_ecc_kdf(Z, 16进制k长度)

u2f_escape_command

u2f_escape_command	变量名 = u2f_escape_command(1字节控制字符, apdu指令)

open_process_lock

open_process_lock	变量名 = open_process_lock(字符串格式互斥量名字)
	a = open_process_lock("abc") aa = 88 b = release_process_lock("abc") 进程锁可以跨进程使用，同一个工具的不同脚本窗口无效，open与release之间的操作被视为一个整体执行，其他进程则等待此进程。

release_process_lock

release_process_lock	变量名 = release_process_lock(字符串格式互斥量名字)

u2f_escape_send

u2f_escape_send	变量名 = u2f_escape_send(tpdu数据)

u2f_escape_recv

u2f_escape_recv	变量名 = u2f_escape_recv()

getbit

getbit	变量名 = getbit(数据, 16进制从0开始字节索引, 高位, 低位)
	xx = getbit(aabbccdd, 02, 07, 00)

getbit_right

getbit_right	变量名 = getbit_right(数据, 16进制从0开始字节索引, 高位, 低位)
	xx = getbit_right(aabbccdd, 02, 07, 00)

aes128_encode_ofb

aes128_encode_ofb	变量名 = aes128_encode_ofb(icv, data, key)

aes128_decode_ofb

aes128_decode_ofb	变量名 = aes128_decode_ofb(icv, data, key)

aes192_encode_ofb

aes192_encode_ofb	变量名 = aes192_encode_ofb(icv, data, key)

aes192_decode_ofb

aes192_decode_ofb	变量名 = aes192_decode_ofb(icv, data, key)

aes256_encode_ofb

aes256_encode_ofb	变量名 = aes256_encode_ofb(icv, data, key)

aes256_decode_ofb

aes256_decode_ofb	变量名 = aes256_decode_ofb(icv, data, key)

sm4_encode_ofb

sm4_encode_ofb	变量名 = sm4_encode_ofb(icv, data, key)

sm4_decode_ofb

sm4_decode_ofb	变量名 = sm4_decode_ofb(icv, data, key)

aes128_encode_ctr

aes128_encode_ctr	变量名 = aes128_encode_ctr(icv, data, key)

aes128_decode_ctr

aes128_decode_ctr	变量名 = aes128_decode_ctr(icv, data, key)

aes192_encode_ctr

aes192_encode_ctr	变量名 = aes192_encode_ctr(icv, data, key)

aes192_decode_ctr

aes192_decode_ctr	变量名 = aes192_decode_ctr(icv, data, key)

aes256_encode_ctr

aes256_encode_ctr	变量名 = aes256_encode_ctr(icv, data, key)

aes256_decode_ctr

aes256_decode_ctr	变量名 = aes256_decode_ctr(icv, data, key)

sm4_encode_ctr

sm4_encode_ctr	变量名 = sm4_encode_ctr(icv, data, key)

sm4_decode_ctr

sm4_decode_ctr	变量名 = sm4_decode_ctr(icv, data, key)

gbk_gb18030

gbk_gb18030	变量名 = gbk_gb18030(gbk)

utf8_gb18030

utf8_gb18030	变量名 = utf8_gb18030(utf8)

utf16_little_gb18030

utf16_little_gb18030	变量名 = utf16_little_gb18030(utf16_little)

gb18030_utf16_little

gb18030_utf16_little	变量名 = gb18030_utf16_little_gb18030(gb18030)

utf16_big_gb18030

utf16_big_gb18030	变量名 = utf16_big_gb18030(utf16_big)

aes128_cmac

aes128_cmac	变量名 = aes128_cmac(icv, data, key)

aes192_cmac

aes192_cmac	变量名 = aes192_cmac(icv, data, key)

aes192_cmac

aes192_cmac	变量名 = aes192_cmac(icv, data, key)

sm4_cmac

sm4_cmac	变量名 = sm4_cmac(icv, data, key)

ansi_utf16_little

ansi_utf16_little	变量名 = ansi_utf16_little(ansi_format_string)

ansi_utf16_big

ansi_utf16_big	变量名 = ansi_utf16_big(ansi_format_string)

utf8_utf16_little

utf8_utf16_little	变量名 = utf8_utf16_little(utf8_format_string)

utf8_utf16_big

utf8_utf16_big	变量名 = utf8_utf16_big(utf8_format_string)

aes128_encode_cfb

aes128_encode_cfb	变量名 = aes128_encode_cfb(icv, data, key, slen)

aes128_decode_cfb

aes128_decode_cfb	变量名 = aes128_decode_cfb(icv, data, key, slen)

aes192_encode_cfb

aes192_encode_cfb	变量名 = aes192_encode_cfb(icv, data, key, slen)

aes192_decode_cfb

aes192_decode_cfb	变量名 = aes192_decode_cfb(icv, data, key, slen)

aes256_encode_cfb

aes256_encode_cfb	变量名 = aes256_encode_cfb(icv, data, key, slen)

aes256_decode_cfb

aes256_decode_cfb	变量名 = aes256_decode_cfb(icv, data, key, slen)

sm4_encode_cfb

sm4_encode_cfb	变量名 = sm4_encode_cfb(icv, data, key, slen)

sm4_decode_cfb

sm4_decode_cfb	变量名 = sm4_decode_cfb(icv, data, key, slen)

des_encode_cfb

des_encode_cfb	变量名 = des_encode_cfb(icv, data, key, slen)

des_decode_cfb

des_decode_cfb	变量名 = des_decode_cfb(icv, data, key, slen)

3des_encode_cfb

3des_encode_cfb	变量名 = 3des_encode_cfb(icv, data, key, slen)

3des_decode_cfb

3des_decode_cfb	变量名 = 3des_decode_cfb(icv, data, key, slen)

3des24_encode_cfb

3des24_encode_cfb	变量名 = 3des24_encode_cfb(icv, data, key, slen)

3des24_decode_cfb

3des24_decode_cfb	变量名 = 3des24_decode_cfb(icv, data, key, slen)

des_encode_ofb

des_encode_ofb	变量名 = des_encode_ofb(icv, data, key)

des_decode_ofb

des_decode_ofb	变量名 = des_decode_ofb(icv, data, key)

3des_encode_ofb

3des_encode_ofb	变量名 = 3des_encode_ofb(icv, data, key)

3des_decode_ofb

3des_decode_ofb	变量名 = 3des_decode_ofb(icv, data, key)

3des24_encode_ofb

3des24_encode_ofb	变量名 = 3des24_encode_ofb(icv, data, key)

3des24_decode_ofb

3des24_decode_ofb	变量名 = 3des24_decode_ofb(icv, data, key)

des_encode_ctr

des_encode_ctr	变量名 = des_encode_ctr(icv, data, key)

des_decode_ctr

des_decode_ctr	变量名 = des_decode_ctr(icv, data, key)

3des_encode_ctr

3des_encode_ctr	变量名 = 3des_encode_ctr(icv, data, key)

3des_decode_ctr

3des_decode_ctr	变量名 = 3des_decode_ctr(icv, data, key)

3des24_encode_ctr

3des24_encode_ctr	变量名 = 3des24_encode_ctr(icv, data, key)

3des24_decode_ctr

3des24_decode_ctr	变量名 = 3des24_decode_ctr(icv, data, key)

des_cmac

des_cmac	变量名 = des_cmac(icv, data, key)

3des_cmac

3des_cmac	变量名 = 3des_cmac(icv, data, key)

3des24_cmac

3des24_cmac	变量名 = 3des24_cmac(icv, data, key)

0.0.5.9 新增加部分

rockey4_encode

rockey4_encode	变量名 = rockey4_encode(data, key)

rockey4_decode

rockey4_decode	变量名 = rockey4_decode(data, key)

cbor 函数组

cbor_init(index, cbor_data)
cbor_stream_get(index)
	用于cbor编码、解码，index是cbor编码对象索引，有效范围是00-03。
	简单用法
	a = cbor_init(00, "")
	// 或
	a = cbor_init(00, 有效数据)
	
	使用cbor_stream_get(index)取得相应的值

cbor_stream_analy(cbor_data) cbor_stream_to_encode(index, cbor_data) cbor_stream_to_decode(index, cbor_data)	调用init之后，可以调用几个函数进行分析、或生成编码

cbor_get_current(index)	生成解码脚本，可逐项取得对应的值
cbor_write_positive(index, value) cbor_write_negative(index, value) cbor_write_any(index, value) cbor_write_map_start(index, value) cbor_write_byte_string(index, value) cbor_write_text_string(index, value) cbor_write_array_start(index, value) cbor_write_bool(index, true_of_false)	cbor编码函数

blakehash 函数组

blake224_hash (data) blake256_hash (data) blake384_hash (data) blake512_hash (data)	

blake224_hash_init (data) blake224_hash_update (data) blake224_hash_dofinal (data) blake256_hash_init (data) blake256_hash_update (data) blake256_hash_dofinal (data) blake384_hash_init (data) blake384_hash_update (data) blake384_hash_dofinal (data) blake512_hash_init (data) blake512_hash_update (data) blake512_hash_dofinal (data)	

oid_encrypt

oid_encrypt	变量名 = oid_encrypt(字符串格式的oid)
	x = oid_encrypt("1.2.840.113549.2.5") //x = //----------MD5 //--2A864886F70D0205

oid_decrypt

oid_decrypt	变量名 = oid_decrypt(oid)

	返回utf16-little格式的字符串

	x = oid_encrypt("1.2.840.113549.2.5")
	y = show_utf16_little_string(oid_decrypt($x))
	
	//x =
	//————————————MD5
	//——2A864886F70D0205
	//y =
	//————————————1.2.840.113549.2.5
	
	//————————————MD5
	
	//[]======================[]
	//[] 1.2.840.113549.2.5 []
	//[]======================[]

crc32

crc32	变量名 = crc32(4字节crc初始化向量, 待crc的数据) x = crc32(00000000, 74) x = crc32($x, 68656D653A206A656B796C6C2D7468656D652D736C6174650A)

rc4_crypt

rcr_crypt	变量名 = rcr_crypt(数据，密钥) x = rc4_crypt(1122, 1122) y = rc4_crypt($x, 1122)

getopenfilename

getopenfilename	变量名 = getopenfilename(“图片文件\0*.jpg”)

getsavefilename

getsavefilename	变量名 = getsavefilename(“图片文件\0*.jpg”)

CreateProcess

CreateProcess	变量名 = CreateProcess(EXE, Para, b_wait_to_quit)

0.0.5.9 版----删除记录

1) 删除了superlong关键字，对应功能可以写在一行
2) 去掉了 pause 关键字的部分功能，pause 只支持暂停
3) 删除了runtime_replace关键字，不再支持此功能

0.0.6.0 新增加部分

libusb 函数族，返回 00 成功，非 00 失败

libusb_init	
libusb_exit	
libusb_finddevice_by_interface	vid, pid, interface_class
libusb_opendevice	
libusb_closedevice	
libusb_claim_interface	interface_u1
libusb_control_transfer	request_type_u1, request_u1, value_u2, index_u2, read_len_u2
libusb_control_transfer_raw	raw_data
libusb_bulk_transfer	endpoint, data
iap2_checksum	data
libusb_set_timeout	time_ms
libusb_reset_device	
libusb_set_config	config_u1

PLC 机械控制函数族，返回 00 成功，非 00 失败

fn_openplc	port_num, baudrate
fn_closeplc	
fn_runlamp_control	onflag_01_or_00, fixtype_t409c_01_t409e_02
fn_arm_zero	armid_based_0, speed_hz, low_speed_hz, direct_00_indirect_01, isplus
fn_arm_motion	armid_based_0, speed_hz, low_speed_hz, direct_00_indirect_01, ispuleses, rate, iscam, isplus
fn_calculate_pulse_num	micro_step, unit_range, goal_distance

记号控制函数族，可用于实现全局变量功能，被注册的记号在脚本窗口打开期间有效

mark_test	记号名
mark_set	记号名, 记号值 用于注册或更新记号内容
mark_get	记号名 取得记号内容
mark_del	记号名 删除某个记号
mark_remove_all	删除所有记号

```
clear

x = mark_test( "already_reset" )

if $x == 00
    // already_reset is not exist
    x = mark_set( "already_reset", 88 )
    x = mark_set( "eee", 88 )
    zzz = 00
else
    zzz = add( $zzz, 01 )
endif

x = mark_get( "already_reset" )
if $x == ""
    ?
    pause
endif

x = mark_del( "eee" )

x = mark_test( "eee" )
if $x != 00
    ?
    pause
endif
```

json 函数族（目前只支持读取功能）

json_load	“json文件名” 返回00表示成功 其他值表示失败

	x json_load($settings_name)
	if $x != 00
	?
	? "找不到配置文件"
	x = show_ansi_string($settings_name)
	pause
	endif
	
	// 加载成功会列出json的所有节点的名字与内容概要

	//----int KeyVersion = 24 (0x18) //----int Option = 2 (0x02) //------string SCP11c.CardGroupID = 01020304 //--------string SCP11c.Curve.k = 01 //------int SCP11c.Digest = 33 (0x21) //------string SCP11c.HostID = 8080808080808080 //------bool SCP11c.IncludeHostID = true //------int SCP11c.KID_CA_KLOC = 16 (0x10) //------string SCP11c.KPR = //------int SCP11c.KeyLength = 16 (0x10) //------string SCP11c.Key_DEK = //----string SCPType = _ //----int sLevel = 0 (0x00)
json_dump	显示json文件的内容与概要
json_get	“json节点名”
	scp11c_oce_1_0 = json_get("scp11c.oce[1][0]")
	注意： 这里取得的内容与原始内容有差异 ● 对于int型，如果文件中的内容是0x11，取出来的是00000011，要注意长度 ● 对于string型，如果文件中的内容是”1234”，取出来的是31323334，要使用parse函数转换成二进制

h2s

h2s	h2s函数是之前的num2txt的别名，这样的做法是因为代码中使用h2s进行转换，有了这样的别名用起来比较熟悉
num2txt	变量名 = num2txt(输入数据)
	输入是00，输出是”00”，也就是3030，无空格。

parse

parse	parse函数是之前的txt2num的别名，这样的做法是因为代码中使用parse进行转换，有了这样的别名用起来比较熟悉
txt2num	变量名 = txt2num (输入数据)
	输入是”00”， 也就是3030，输出是00

fthub_init

fthub_open

fthub_close

fthub_init	x = fthub_init()
	为简单起见，只支持一个hub
fthub_open	x = fthub_open(16进制端口号，00-07)

	为简单起见，只支持一个hub
fthub_close	x = fthub_close(16进制端口号，全部关闭)
	为简单起见，只支持一个hub

```
x    = fthub_init()
if $x != 00
    ?
    pause
endif

num = 00
for i = 0 to 400
    x    = fthub_open( $num )
    if $x != 00
        ?
        pause
    endif

    sleep 100
    num = add( $num, 01 )
    if $num > 07
        num = 00
    endif

    x    = fthub_close( $num )        // 全部关闭

    if $x != 00
        ?
        pause
    endif

next i
```

unix_utc

utc_unix

unix_utc	x = unix_utc(11228844)
	4字节unix时间戳
utc_unix	y = utc_unix(790210062300) 或 y = utc_unix(19790210062300)
	6或7个字节的utc时间

fthub_init_2

fthub_open_2

fthub_close_2

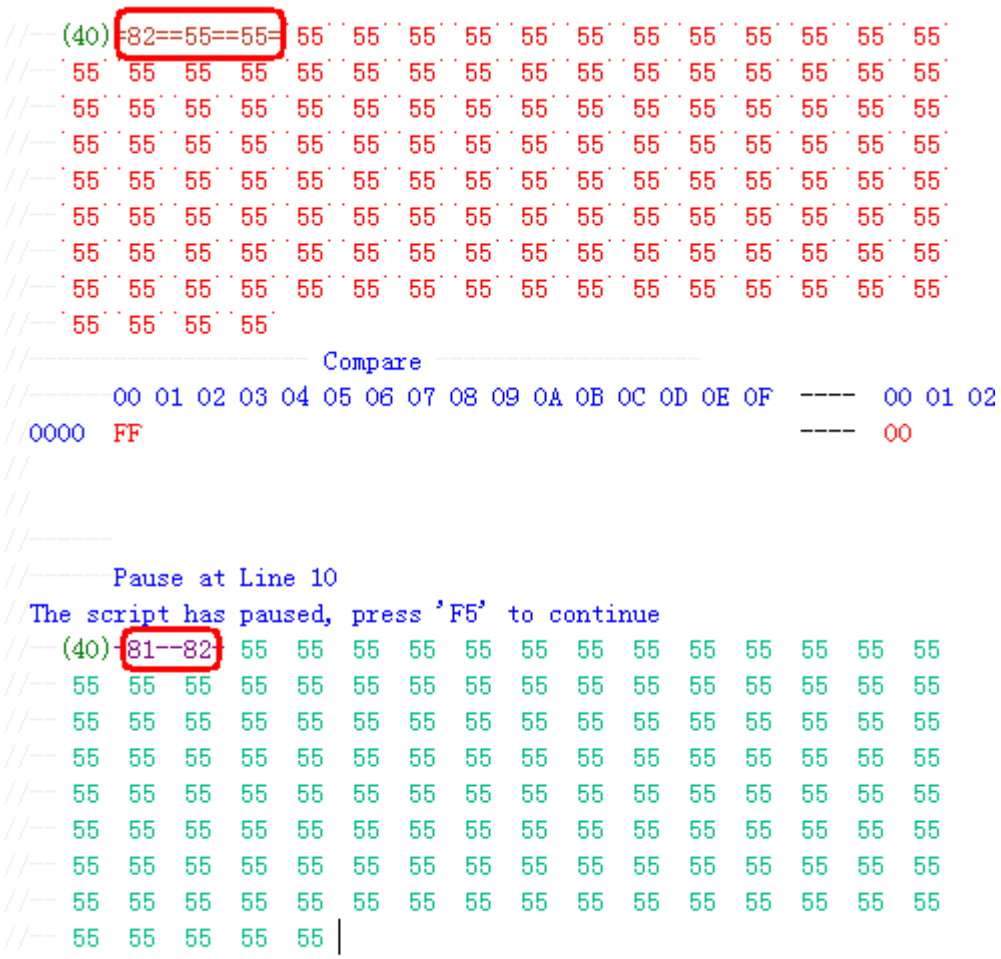
fthub_init_2	与fthub_init相同，此函数支持带电源的hub
fthub_open_2	与fthub_open相同，此函数支持带电源的hub
fthub_close_2	与fthub_close相同，此函数支持带电源的hub

check_tlv

check_dcep_tlv

check_tlv

```
y = check_tlv( $input )

clear
prompt off

t = dup( 130, 55 )
x = 40( $t )
y = check_tlv( $x )
if $y != 00
    ?
    pause
endif

t = dup( 130, 55 )
x = 40 81( $t )
y = check_tlv( $x )
if $y != 00
    ?
    pause
endif
```

	 此函数检查tlv，如果正确返回00，如果不正确，返回非00
check_dcep_tlv	y = check_dcep_tlv($input) 此函数与check_tlv类似

aes128_encode_ccm

aes128_decode_ccm

aes192_encode_ccm

aes192_decode_ccm

aes256_encode_ccm

aes256_decode_ccm

aes_128_encode_ccm	key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
	nonce = 00 00 00 03 02 01 00 A0 A1 A2 A3 A4 A5
	Adata = 00 01 02 03 04 05 06 07
	// Adata =
	len_adata = datalen($adata)
	m = 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E
	
	x = aes128_encode_ccm($nonce, $adata, $m, $key, 08)
	
	cipher = mid($x, int(16 + 0x$len_adata), 23)
	返回数据分为4个部分
	● 前0x10个字节为B0
	● 外部传入的adata，与输入数据长度相同
	● 密文，与明文长度相同
	● mac，长度由外面指定

aes_128_decode_ccm	y = aes128_decode_ccm($nonce, $adata, $cipher, $key, 08)

0.0.6.1 新增加部分

aes128_encode_gcm

aes128_decode_gcm

aes192_encode_gcm

aes192_decode_gcm

aes256_encode_gcm

aes256_decode_gcm

aes_128_encode_gcm	Key = f5a2b27c74355872eb3ef6c5feafaa740e6ae990d9d48c3bd9bb8235e589f010 IV = 58d2240f580a31c1d24948e9 Tag = 15e051a5e4a5f5da6cea92e2ebee5bac tem_xx = aes256_encode_gcm($iv, "", "", $key, 10) if $tem_xx != $tag ? pause endif

```
xx          = aes256_decode_gcm( $iv, , , $key, $tag )
if $xx != ""
    ?
    pause
endif

Key         = 11754cd72aec309bf52f7687212e8957
IV          = 3c819d9a9bed087615030b65
Tag         = 250327c674aaf477aef2675748cf6971
tem_xx      = aes128_encode_gcm($IV, , , $Key, 10)
if $tem_xx != $tag
    ?
    pause
endif

xx          = aes128_decode_gcm( $iv, , , $key, $tag )
if $xx != ""
    ?
    pause
endif


mac_length  = 04
Key         = 0e00c76561d2bd9b40c3c15427e2b08f
IV          =
492cadaccd3ca3fbc9cf9f06eb3325c4e159850b0dbe98199b89b7af528806610b6f63
998e1eae80c348e74cbb921d8326631631fc6a5d304f39166daf7ea15fa1977f101819
adb510b50fe9932e12c5a85aa3fd1e73d8d760af218be829903a77c63359d75edd91b4
f6ed5465a72662f5055999e059e7654a8edc921aa0d496
PT          =
fef03c2d7fb15bf0d2df18007d99f967c878ad59359034f7bb2c19af120685d78e32f6
b8b83b032019956ca9c0195721476b85
AAD         =
d8f1163d8c840292a2b2dacf4ac7c36aff8733f18fabb4fa5594544125e03d1e6e5d6d
0fd61656c8d8f327c92839ae5539bb469c9257f109ebff85aad7bd220fdaa95c022dbd
0c7bb2d878ad504122c943045d3c5eba8f1f56c0
CT          =
4f6cf471be7cbd2575cd5a1747aea8fe9dea83e51936beac3e68f66206922060c697ff
a7af80ad6bb68f2cf4fc97416ee52abe
Tag         = e20b6655

tem_xx      = aes128_encode_gcm($IV, $PT, $AAD, $Key, 04)
```

	if $tem_xx != $CT $Tag
	  ?
	  pause
	endif
	
	xx = aes128_decode_gcm($iv, $tem_xx, $aad, $key, $tag)
	x = datalen($xx)
	x = sub($x, $mac_length)
	yy = hmid($xx, 00, $x)
	if $yy != $pt
	  ?
	  pause
	endif
	
	返回数据分为2个部分
	● 第一部分密文，与明文长度一致
	● mac，长度由外面指定

aes_128_decode_gcm	y = aes128_decode_gcm(iv, data, aad, key, mac)

asn1_reset

asn1_length_begin

asn1_data

asn1_pushtlv

asn1_length_end

asn1_final

用法	x = asn1_reset()
	x = asn1_data(31)
	inc_indent
	  x = asn1_length_begin()
	  x = asn1_data(30)
	  inc_indent
	  x = asn1_length_begin()
	  x = asn1_data(06)
	  inc_indent
	  x = asn1_length_begin()
	  x = asn1_data(04 00 7F 00 07 02 02 02)
	  x = asn1_length_end()

```
        dec_indent
        x    = asnl_data( 02 )
        inc_indent
            x    = asnl_length_begin()
            x    = asnl_data( 02 )
            x    = asnl_length_end()
        dec_indent
        x    = asnl_length_end()
    dec_indent

    x    = asnl_pushtlv( 02, 9999 )
    x    = asnl_pushtlv( 02, 88888888 )
    x    = asnl_pushtlv( 02, 77777777777777777777 )
    x    = asnl_length_end()
dec_indent

x    = asnl_final()
```

```
//x = 3126300D060804007F00070202020201020202999902048888888020B77777777777777777777
(T:31, 0:0000, L:0026) SET
    (T:30, 0:0002, L:000D) SEQUENCE
        (T:06, 0:0004, L:0008) OBJECT IDENTIFIER : bsiTA (
        (T:02, 0:000E, L:0001) INTEGER : 02
    (T:02, 0:0011, L:0002) INTEGER : 9999
    (T:02, 0:0015, L:0004) INTEGER : 88888888
    (T:02, 0:001B, L:000B) INTEGER : 77777777777777777777
```

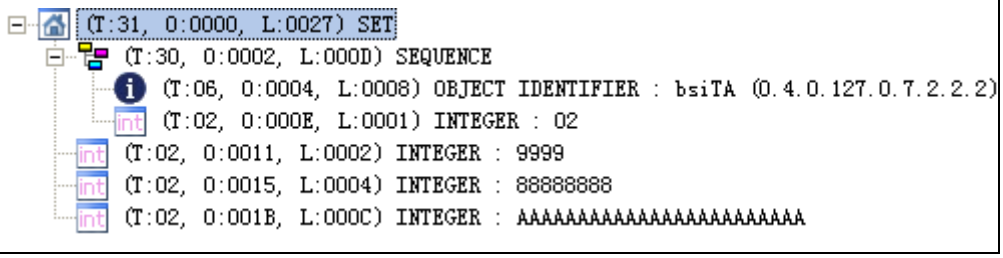
settlv

用法	`z = settlv( $x, 31 30 02, aaaaaaaaaaaaaaaaaaaaaaa )`

```
//z = 31313018060804007F00070202020201CAAAAAAAAAAAAAAAAAAAAAAAA020299990204888888888020B77777777777777777777
(T:31, 0:0000, L:0031) SET
    (T:30, 0:0002, L:0018) SEQUENCE
        (T:06, 0:0004, L:0008) OBJECT IDENTIFIER : bsiTA (0.4.0.127.0.7.2.2.2)
        (T:02, 0:000E, L:000C) INTEGER : AAAAAAAAAAAAAAAAAAAAAAAA
    (T:02, 0:001C, L:0002) INTEGER : 9999
    (T:02, 0:0020, L:0004) INTEGER : 88888888
    (T:02, 0:0026, L:000B) INTEGER : 77777777777777777777
```

用法	`z = settlv( $x, 31 02, aaaaaaaaaaaaaaaaaaaaaaa, 02 )`

```
//z =
3127300D060804007F0007020202020102020299990204888888888020CAAAAAAAAAAAA
AAAAAAAAAAAA
```



gettlv_bypath

settlv_bypath

用法

```
der =
3037A0030201020202A34A20E85000000002CD6300D06092A864886F70D010105050030
1531133011060A0992268993F22C6401191603636F6D

path = "/0/3/0/0/0"
path = "/0/0/0"

value = gettlv_bypath( $der, $path )

new_der = settlv_bypath( $der, $path, 888888 )

compare $der, $new_der
```

```
//------------------- Compare -------------------
//      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ----  00 01 02 03 04 05 06 07 08 09 0A 0B
//0000  30 37 A0 03 02 01 02 02 0A 34 A2 0E 85 00 00 00  ----  30 39 A0 05 02 03 88 88 88 02 0A 34
//0010  00 2C D6 30 0D 06 09 2A 86 48 86 F7 0D 01 01 05  ----  00 00 00 2C D6 30 0D 06 09 2A 86 48
//0020  05 00 30 15 31 13 30 11 06 0A 09 92 26 89 93 F2  ----  01 05 05 00 30 15 31 13 30 11 06 0A
//0030  2C 64 01 19 16 03 63 6F 6D -- --                 ----  93 F2 2C 64 01 19 16 03 63 6F 6D
//
```

passport_calcdigit

用法

```
x = passport_calcdigit( "11223344" )

//x = 38
```

ext_euclid

用法

根据扩展欧几里得算法，根据输入a，b的值，得到x，y与sign标志

满足a * x - b * y = 1，或b * y - a * x = 1

```
// 扩展欧几里得算法，普通小数计算
a           = 80
b           = 5b
```

```
xy              = ext_euclid( $a, $b )
x               = mid( $xy, 0, 4 )
y               = mid( $xy, 4, 4 )
gcd             = mid( $xy, 8, 4 )
sign            = mid( $xy, 12, 1 )

x_a             = big_mul( $a, $x )
y_b             = big_mul( $b, $y )

res             = big_sub( $x_a, $y_b )

//a = 80
//b = 5B
//xy = 000000200000002DFF
//x = 00000020
//y = 0000002D
//gcd = 00000001
//sign = FF
//x_a = 1000
//y_b = 0FFF
//res = 0001
```

```
a               = new_sm2_generate_keypair()
self_pri        = left( $a, 32 )
self_pub        = right( $a, 64 )

a               = new_sm2_generate_keypair()
self_tmp_pri    = left( $a, 32 )
self_tmp_pub    = right( $a, 64 )

a               = new_sm2_generate_keypair()
other_pri       = left( $a, 32 )
other_pub       = right( $a, 64 )

a               = new_sm2_generate_keypair()
other_tmp_pri   = left( $a, 32 )
other_tmp_pub   = right( $a, 64 )

self_id         = dup( 16, 11 )
other_id        = dup( 16, 22 )

keyab           = new_sm2_keyexchange( 00, $self_pri, $self_tmp_pri, $other_pub, $other_tmp_pub,
$self_id, $other_id )
```

```
keyba          = new_sm2_keyexchange( 01, $other_pri, $other_tmp_pri, $self_pub, $self_tmp_pub,
$other_id, $self_id )

if $keyab != $keyba
    ?
    pause
endif

self_pri       = 7D511F0A173101AD4B1E204956946F6C6E1B0A087C9F5AFC405751CD74C87429
self_pub       =
13F17A6A022AFFE04523A1BABE9A65C3BB3B90EDB57404F1D4CC80C048E57BB507EF820388F0771D2C5AA92DAE53
B75732C8A774F838E1095FC2E4E836F7FB29

self_tmp_pri   = 16332BBB07B929915707348063EF6B922A7578C7203D405514CB2B5A6A1F6D93
self_tmp_pub   =
D15AA0A9EB024EE12AF054D6BF25A8887CF6C5CC9910DD535C6E8D2DAD0DD2707A378F38876B37675B985591E403
028FA8E10BD0020F7F27608CA4FDF2A95745

other_pri      = 581C3B64332F12386F874C02278249A0531906B8715F44B16A1F3155230C76CD
other_pub      =
1D9E4517A0483378B59CCD8CD65D5A6F180FA85850102E0CFCE0FD6E27B38C4D7434456860CF9292B0E29505F7CE
8A23031F6997EFBEF90C18FAA1EB596A2F94

other_tmp_pri  = 2B3B097C799A6F9333BA3D785CC26B37031456B663EB605F0C5233C718E337D8
other_tmp_pub  =
5FFE69C7756B2B14B3CC131A10AB0E0DAD9C5778F89E52CD5C94E354D8F80592ACD3B97849C18FBFAA38E3456C60
C5E7CC5581691681DE57FEDFE1E720165CB2

keyab          = new_sm2_keyexchange( 00, $self_pri, $self_tmp_pri, $other_pub, $other_tmp_pub,
$self_id, $other_id )

keyba          = new_sm2_keyexchange( 01, $other_pri, $other_tmp_pri, $self_pub, $self_tmp_pub,
$other_id, $self_id )

if $keyab != CF41314ECF7C0FEDB67CC92D62C5434C0E33C07264120F85990E5B98EC88B4F5
    ?
    pause
endif

if $keyba != CF41314ECF7C0FEDB67CC92D62C5434C0E33C07264120F85990E5B98EC88B4F5
    ?
    pause
```

```
endif
```

big_mod_u32

用法	

	big_a = random(13)
	small_b = 5b
	m = big_div_u32($big_a, $small_b)
	r = big_mod_u32($big_a, $small_b)
	xx = big_mul($m, $small_b)
	xx = big_add($xx, $r)
	if $xx != $big_a
	    ?
	    pause
	endif
	
	//big_a = 8A696809FB9895936B5B547E6D
	//small_b = 5B
	//m = 000000018560CAA8BE29073F11E1FBC3
	//r = 0000001C
	//xx = 8A696809FB9895936B5B547E51
	//xx = 8A696809FB9895936B5B547E6D

0.0.6.2 新增加部分

new_sm9_gen_sign_master_key

new_sm9_get_sign_master_key

new_sm9_gen_sign_pri_key

new_sm9_sign

new_sm9_verify

用法	

	// 生成主公私钥
	x = new_sm9_gen_sign_master_key()
	pri = mid($x, 0, 32)
	pub = mid($x, 32)
	
	// 生成签名私钥
	dek = new_sm9_gen_sign_pri_key(ansi_string("Alice"), 01, $pri)

```
// 采用规范中的标准数据
// 签名主密钥和用户签名密钥产生过程中的相关值：
// 签名主私钥ks：0130E7 8459D785 45CB54C5 87E02CF4 80CE0B66 340F319F
348A1D5B 1F2DC5F4
// 签名主公钥Ppub-s = [ks]P2 = (xPpub-s , yPpub-s)：
// 坐标xPpub-s：(9F64080B 3084F733 E48AFF4B 41B56501 1CE0711C 5E392CFB
0AB1B679 1B94C408,
//              29DBA116 152D1F78 6CE843ED 24A3B573 414D2177 386A92DD
8F14D656 96EA5E32)
// 坐标yPpub-s：(69850938 ABEA0112 B57329F4 47E3A0CB AD3E2FDB 1A77F335
E89E1408 D0EF1C25,
//              41E00A53 DDA532DA 1A7CE027 B7A46F74 1006E85F 5CDFF073
0E75C05F B4E3216D)

ks  = 000130E7 8459D785 45CB54C5 87E02CF4 80CE0B66 340F319F 348A1D5B
1F2DC5F4
pub = new_sm9_get_sign_master_key( $ks )
if $pub !=
9F64080B3084F733E48AFF4B41B565011CE0711C5E392CFB0AB1B6791B94C40829DBA1
16152D1F786CE843ED24A3B573414D2177386A92DD8F14D65696EA5E3269850938ABEA
0112B57329F447E3A0CBAD3E2FDB1A77F335E89E1408D0EF1C2541E00A53DDA532DA1A
7CE027B7A46F741006E85F5CDFF0730E75C05FB4E3216D
    ?
    pause
endif

ida = ansi_string( "Alice" )
hid = 01

// 签名私钥dsA = [t2]P1 = (xdsA, ydsA)：
// 坐标xdsA：A5702F05 CF131530 5E2D6EB6 4B0DEB92 3DB1A0BC F0CAFF90
523AC875 4AA69820
// 坐标ydsA：78559A84 4411F982 5C109F5E E3F52D72 0DD01785 392A727B
B1556952 B2B013D3

dsa = new_sm9_gen_sign_pri_key( $ida, $hid, $ks )
if $dsa !=
A5702F05CF1315305E2D6EB64B0DEB923DB1A0BCF0CAFF90523AC8754AA6982078559A
844411F9825C109F5EE3F52D720DD01785392A727BB1556952B2B013D3
    ?
    pause
endif
```

```
// 签名步骤中的相关值：
// 待签名消息M：Chinese IBS standard
// M 的16 进制表示：4368696E 65736520 49425320 7374616E 64617264

m   = ansi_string( "Chinese IBS standard" )

r_s = new_sm9_sign( $m, $pub, $dsa )
r   = mid( $r_s, 0, 32 )
s   = mid( $r_s, 32 )

res = new_sm9_verify( $m, $ida, $hid, $pub, $r_s )
if $res != 00
    ?
    pause
endif

? "success"
```

new_sm9_get_saved_random

用法	取得sm9运算中用到的随机数

new_sm9_gen_encrypt_master_key

new_sm9_get_encrypt_master_key

new_sm9_gen_encrypt_pri_key

new_sm9_encrypt_block

new_sm9_encrypt_stream

new_sm9_decrypt_block

new_sm9_decrypt_stream

用法

```
x       = new_sm9_gen_encrypt_master_key()
pri     = mid( $x, 0, 32 )
pub     = mid( $x, 32 )

enc_pri = new_sm9_gen_encrypt_pri_key( ansi_string( "Bob" ), 03, $pri )

// 加密主私钥ke：01EDEE 3778F441 F8DEA3D9 FA0ACC4E 07EE36C9 3F9A0861
8AF4AD85 CEDE1C22
```

```
// 加密主公钥Ppub-e = [ke]P1= (xPpub-e , yPpub-e):
// 坐标xPpub-e: 787ED7B8 A51F3AB8 4E0A6600 3F32DA5C 720B17EC A7137D39
ABC66E3C 80A892FF
// 坐标yPpub-e: 769DE617 91E5ADC4 B9FF85A3 1354900B 20287127 9A8C49DC
3F220F64 4C57A7B1
// 加密私钥生成函数识别符hid: 0x03
// 实体B 的标识IDB: Bob
// IDB 的16 进制表示: 426F62

pri     = 0001EDEE 3778F441 F8DEA3D9 FA0ACC4E 07EE36C9 3F9A0861 8AF4AD85
CEDE1C22
pub     = new_sm9_get_encrypt_master_key( $pri )
if $pub !=
787ED7B8A51F3AB84E0A66003F32DA5C720B17ECA7137D39ABC66E3C80A892FF769DE6
1791E5ADC4B9FF85A31354900B202871279A8C49DC3F220F644C57A7B1
    ?
    pause
endif

idb     = ansi_string( "Bob" )
hid     = 03

// 计算deB=[t2]P2=(xdeB, ydeB):
// 21
// 坐标xdeB: (94736ACD 2C8C8796 CC4785E9 38301A13 9A059D35 37B64141
40B2D31E ECF41683,
//           115BAE85 F5D8BC6C 3DBD9E53 42979ACC CF3C2F4F 28420B1C
B4F8C0B5 9A19B158)
// 坐标ydeB: (7AA5E475 70DA7600 CD760A0C F7BEAF71 C447F384 4753FE74
FA7BA92C A7D3B55F,
//           27538A62 E7F7BFB5 1DCE0870 4796D94C 9D56734F 119EA447
32B50E31 CDEB75C1)

deb     = new_sm9_gen_encrypt_pri_key( $idb, $hid, $pri )
if $deb !=
94736ACD2C8C8796CC4785E938301A139A059D3537B6414140B2D31EECF41683115BAE
85F5D8BC6C3DBD9E5342979ACCCF3C2F4F28420B1CB4F8C0B59A19B1587AA5E47570DA
7600CD760A0CF7BEAF71C447F3844753FE74FA7BA92CA7D3B55F27538A62E7F7BFB51D
CE08704796D94C9D56734F119EA44732B50E31CDEB75C1
    ?
    pause
endif
```

```
m       = ansi_string( "Chinese IBE standard" )

cipher  = new_sm9_encrypt_block( $m, $idb, $hid, $pub )
plain   = new_sm9_decrypt_block( $cipher, $idb, $hid, $deb )

if $plain != $m
    ?
    pause
endif

cipher  = new_sm9_encrypt_stream( $m, $idb, $hid, $pub )
plain   = new_sm9_decrypt_stream( $cipher, $idb, $hid, $deb )

if $plain != $m
    ?
    pause
endif

? "success"
```

new_sm9_key_pack

new_sm9_key_unpack

用法

```
// 加密主私钥ke：01EDEE 3778F441 F8DEA3D9 FA0ACC4E 07EE36C9 3F9A0861
8AF4AD85 CEDE1C22
// 加密主公钥Ppub-e = [ke]P1= (xPpub-e , yPpub-e)：
// 坐标xPpub-e：787ED7B8 A51F3AB8 4E0A6600 3F32DA5C 720B17EC A7137D39
ABC66E3C 80A892FF
// 坐标yPpub-e：769DE617 91E5ADC4 B9FF85A3 1354900B 20287127 9A8C49DC
3F220F64 4C57A7B1
// 加密私钥生成函数识别符hid：0x03
// 实体B 的标识IDB: Bob
// IDB 的16 进制表示：426F62

pri = 0001EDEE 3778F441 F8DEA3D9 FA0ACC4E 07EE36C9 3F9A0861 8AF4AD85
CEDE1C22
pub = new_sm9_get_encrypt_master_key( $pri )
if $pub !=
787ED7B8A51F3AB84E0A66003F32DA5C720B17ECA7137D39ABC66E3C80A892FF769DE6
1791E5ADC4B9FF85A31354900B202871279A8C49DC3F220F644C57A7B1
    ?
```

```
    pause
endif

idb = ansi_string( "Bob" )
hid = 03

// 计算deB=[t2]P2=(xdeB, ydeB):
// 21
// 坐标xdeB: (94736ACD 2C8C8796 CC4785E9 38301A13 9A059D35 37B64141
40B2D31E ECF41683,
//          115BAE85 F5D8BC6C 3DBD9E53 42979ACC CF3C2F4F 28420B1C
B4F8C0B5 9A19B158)
// 坐标ydeB: (7AA5E475 70DA7600 CD760A0C F7BEAF71 C447F384 4753FE74
FA7BA92C A7D3B55F,
//          27538A62 E7F7BFB5 1DCE0870 4796D94C 9D56734F 119EA447
32B50E31 CDEB75C1)

deb = new_sm9_gen_encrypt_pri_key( $idb, $hid, $pri )
if $deb !=
94736ACD2C8C8796CC4785E938301A139A059D3537B6414140B2D31EECF41683115BAE
85F5D8BC6C3DBD9E5342979ACCCF3C2F4F28420B1CB4F8C0B59A19B1587AA5E47570DA
7600CD760A0CF7BEAF71C447F3844753FE74FA7BA92CA7D3B55F27538A62E7F7BFB51D
CE08704796D94C9D56734F119EA44732B50E31CDEB75C1
    ?
    pause
endif

len = 10
x   = new_sm9_key_pack( $idb, $hid, $pub, $len )
k   = hmid( $x, 00, $len )
c   = hmid( $x, $len )

y   = new_sm9_key_unpack( $c, $idb, $deb, $len )
if $y != $k
    ?
    pause
endif

// 换一个长度
len = 30
x   = new_sm9_key_pack( $idb, $hid, $pub, $len )
k   = hmid( $x, 00, $len )
c   = hmid( $x, $len )
```

```
y    = new_sm9_key_unpack( $c, $idb, $deb, $len )
if $y != $k
    ?
    pause
endif


? "success"
```

new_sm9_h1

用法

```
x = new_sm9_h1( 112233,
B640000002A3A6F1D603AB4FF58EC74449F2934B18EA8BEEE56EE19CD69ECF25 )
```

new_sm9_key_exchange

用法

```
// 加密主密钥和用户加密密钥产生过程中的相关值：
// 加密主私钥ke：02E65B 0762D042 F51F0D23 542B13ED 8CFA2E9A 0E720636
1E013A28 3905E31F
// 加密主公钥Ppub-e = [ke]P1 = (xPpub-e , yPpub-e)：
// 坐标xPpub-e：91745426 68E8F14A B273C094 5C3690C6 6E5DD096 78B86F73
4C435056 7ED06283
// 坐标yPpub-e：54E598C6 BF749A3D ACC9FFFE DD9DB686 6C50457C FC7AA2A4
AD65C316 8FF74210
// 加密私钥生成函数识别符hid：0x02
// 实体A 的标识IDA：Alice

pri     = 0002E65B 0762D042 F51F0D23 542B13ED 8CFA2E9A 0E720636 1E013A28
3905E31F
pub     = new_sm9_get_encrypt_master_key( $pri )
if $pub !=
9174542668E8F14AB273C0945C3690C66E5DD09678B86F734C4350567ED0628354E598
C6BF749A3DACC9FFFEDD9DB6866C50457CFC7AA2A4AD65C3168FF74210
    ?
    pause
endif

ida     = ansi_string( "Alice" )
idb     = ansi_string( "Bob" )
hid     = 02
```

```
dea      = new_sm9_gen_encrypt_pri_key( $ida, $hid, $pri )
if $dea !=
0FE8EAB395199B56BF1D75BD2CD610B6424F08D1092922C5882B52DCD6CA832A7DA57B
C50241F9E5BFDDC075DD9D32C7777100D736916CFC165D8D36E0634CD783A457DAF52C
AD464C903B26062CAF937BB40E37DADED9EDA401050E49C8AD0C6970876B9AAD1B7A50
BB4863A11E574AF1FE3C5975161D73DE4C3AF621FB1EFB
    ?
    pause
endif

deb      = new_sm9_gen_encrypt_pri_key( $idb, $hid, $pri )
if $deb !=
74CCC3AC9C383C60AF083972B96D05C75F12C8907D128A17ADAFBAB8C5A4ACF701092F
F4DE89362670C21711B6DBE52DCD5F8E40C6654B3DECE573C2AB3D29B244B0294AA042
90E1524FF3E3DA8CFD432BB64DE3A8040B5B88D1B5FC86A4EBC18CFC48FB4FF37F1E27
727464F3C34E2153861AD08E972D1625FC1A7BD18D5539
    ?
    pause
endif

// 取rA 为: 5879 DD1D51E1 75946F23 B1B41E93 BA31C584 AE59A426 EC1046A4
D03B06C8
// 计算RA=[rA]QB=( xRA , yRA):
// 坐标xRA:7CBA5B19 069EE66A A79D4904 13D11846 B9BA76DD 22567F80 9CF23B6D
964BB265
// 坐标yRA:A9760C99 CB6F7063 43FED056 37085864 958D6C90 902ABA7D 405FBEDF
7B781599

ra       = 00005879 DD1D51E1 75946F23 B1B41E93 BA31C584 AE59A426 EC1046A4
D03B06C8
rra      = 7CBA5B19 069EE66A A79D4904 13D11846 B9BA76DD 22567F80 9CF23B6D
964BB265 A9760C99 CB6F7063 43FED056 37085864 958D6C90 902ABA7D 405FBEDF
7B781599

// 取rB 为: 018B98 C44BEF9F 8537FB7D 071B2C92 8B3BC65B D3D69E1E EE213564
905634FE
// 计算RB=[rB]QA=( xRB , yRB):
// 坐标xRB:861E9148 5FB7623D 2794F495 031A3559 8B493BD4 5BE37813 ABC710FC
C1F34482
// 坐标yRB:32D906A4 69EBC121 6A802A70 52D5617C D430FB56 FBA729D4 1D9BD668
E9EB9600
rb       = 00018B98 C44BEF9F 8537FB7D 071B2C92 8B3BC65B D3D69E1E EE213564
```

	905634FE rrb = 861E9148 5FB7623D 2794F495 031A3559 8B493BD4 5BE37813 ABC710FC C1F34482 32D906A4 69EBC121 6A802A70 52D5617C D430FB56 FBA729D4 1D9BD668 E9EB9600 len = 10 keyb = new_sm9_key_exchange(00, $ida, $hid, $idb, $pub, $dea, $ra, $rra, $rrb, $len) if $keyb != C5C13A8F59A97CDEAE64F16A2272A9E7 ? pause endif keya = new_sm9_key_exchange(01, $idb, $hid, $ida, $pub, $deb, $rb, $rrb, $rra, $len) if $keya != C5C13A8F59A97CDEAE64F16A2272A9E7 ? pause endif ? "success"

无版本

show_wsq_image

用法	wsq = FFA0FFA8007B4E4953545F434F4D20390A5049585F57494454482035313 50A5049585F48454947485420343837.....FECC7FC4FC0F46E146D4A1FA9DA45F5BC5FC4162CF8DDA79FC85100D4E4667103C24323F61D47B0ED122C7B4276CED0B287B549DDEB63763AB4D34FA1ED3FA4FFFA1 x = show_wsq_image($wsq)

save_wsq_to_tmp

用法	wsq = FFA0FFA8007B4E4953545F434F4D20390A5049585F57494454482035313 50A5049585F48454947485420343837.....FECC7FC4FC0F46E146D4A1FA9DA45F5BC5FC4162CF8DDA79FC85100D4E4667103C24323F61D47B0ED122C7B4276CED0B287B549DDEB63763AB4D34FA1ED3FA4FFFA1 x = save_wsq_to_bmp($wsq, "finger.bmp")

big_div

big_mod

big_mod_div

```
x   = random( 80 )
y   = random( 60 )

q   = big_div( $x, $y )
r   = big_mod( $x, $y )

z   = big_mul( $q, $y )
z   = big_add( $z, $r )
if $z != $x
    ?
    pause
endif
```

getnextprime

getprevprime

```
x   = random( 80 )
y   = getnextprime( $x )
z   = getprevprime( $x )
compare $z, $x
compare $x, $y
compare $z, $y
```

```
//------------------- Compare -------------------
//      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ----  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
// 0000 1B 66 90 A0 B9 BD 36 EA 62 70 52 3A C9 77 C9 6D  ----  1B 66 90 A0 B9 BD 36 EA 62 70 52 3A C9 77 C9 6D
// 0010 A9 B0 D6 64 AB 2F 09 FD 8A D5 8F 42 75 24 B3 E1  ----  A9 B0 D6 64 AB 2F 09 FD 8A D5 8F 42 75 24 B3 E1
// 0020 B5 51 A5 DD 8F 9B 7A 82 6C DC 75 9E 37 5E 85 EE  ----  B5 51 A5 DD 8F 9B 7A 82 6C DC 75 9E 37 5E 85 EE
// 0030 07 EF 0E D7 FE 38 EC 56 71 4A B6 3A 48 7A 3F 91  ----  07 EF 0E D7 FE 38 EC 56 71 4A B6 3A 48 7A 3F 91
// 0040 A8 6F 63 60 D0 79 00 97 41 24 42 43 22 0E 23 DF  ----  A8 6F 63 60 D0 79 00 97 41 24 42 43 22 0E 24 06
//
//
//------------------- Compare -------------------
//      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ----  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
// 0000 1B 66 90 A0 B9 BD 36 EA 62 70 52 3A C9 77 C9 6D  ----  1B 66 90 A0 B9 BD 36 EA 62 70 52 3A C9 77 C9 6D
// 0010 A9 B0 D6 64 AB 2F 09 FD 8A D5 8F 42 75 24 B3 E1  ----  A9 B0 D6 64 AB 2F 09 FD 8A D5 8F 42 75 24 B3 E1
// 0020 B5 51 A5 DD 8F 9B 7A 82 6C DC 75 9E 37 5E 85 EE  ----  B5 51 A5 DD 8F 9B 7A 82 6C DC 75 9E 37 5E 85 EE
// 0030 07 EF 0E D7 FE 38 EC 56 71 4A B6 3A 48 7A 3F 91  ----  07 EF 0E D7 FE 38 EC 56 71 4A B6 3A 48 7A 3F 91
// 0040 A8 6F 63 60 D0 79 00 97 41 24 42 43 22 0E 24 06  ----  A8 6F 63 60 D0 79 00 97 41 24 42 43 22 0E 24 65
//
//
//------------------- Compare -------------------
//      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ----  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
// 0000 1B 66 90 A0 B9 BD 36 EA 62 70 52 3A C9 77 C9 6D  ----  1B 66 90 A0 B9 BD 36 EA 62 70 52 3A C9 77 C9 6D
// 0010 A9 B0 D6 64 AB 2F 09 FD 8A D5 8F 42 75 24 B3 E1  ----  A9 B0 D6 64 AB 2F 09 FD 8A D5 8F 42 75 24 B3 E1
// 0020 B5 51 A5 DD 8F 9B 7A 82 6C DC 75 9E 37 5E 85 EE  ----  B5 51 A5 DD 8F 9B 7A 82 6C DC 75 9E 37 5E 85 EE
// 0030 07 EF 0E D7 FE 38 EC 56 71 4A B6 3A 48 7A 3F 91  ----  07 EF 0E D7 FE 38 EC 56 71 4A B6 3A 48 7A 3F 91
// 0040 A8 6F 63 60 D0 79 00 97 41 24 42 43 22 0E 23 DF  ----  A8 6F 63 60 D0 79 00 97 41 24 42 43 22 0E 24 65
//
```

```
new_sm2_get_yflag( point )

new_sm2_computer_y( yflag, point_x )

new_sm2_check_point( point )

new_sm2_point_add( p, q )

new_sm2_point_double( p )

new_sm2_check_pubkey( pubkey )

new_sm2_ecdh_gm_map( k, point )

new_sm2_kp( k, p )

new_sm2_kp_add_lq( k, p, l, q )

new_ecc_get_yflag( point )
```

```
// sm2
P                = FF FF FF FE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00 FF FF
FF FF FF FF FF FF
A                = FF FF FF FE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00 FF FF
FF FF FF FF FF FC
B                = 28 E9 FA 9E 9D 9F 5E 34 4D 5A 9E 4B CF 65 09 A7 F3 97 89 F5 15 AB 8F 92 DD BC
BD 41 4D 94 0E 93
GX               = 32 C4 AE 2C 1F 19 81 19 5F 99 04 46 6A 39 C9 94 8F E3 0B BF F2 66 0B E1 71 5A
```

```
45 89 33 4C 74 C7
GY              = BC 37 36 A2 F4 F6 77 9C 59 BD CE E3 6B 69 21 53 D0 A9 87 7C C6 2A 47 40 02 DF
32 E5 21 39 F0 A0
N               = FF FF FF FE FF FF FF FF FF FF FF FF FF FF FF FF 72 03 DF 6B 21 C6 05 2B 53 BB
F4 09 39 D5 41 23

h               = 01
hex_len_in_byte = 20

r               = new_ecc_initialize( $p, $a, $b, $gx, $gy, $n, $h, $hex_len_in_byte )
if $r != 00
    ?
    pause
endif

keypair         = new_ecc_generate_keypair()
len             = datalen( $keypair )
prilen          = big_div( $len, 03 )
if $len > 00
    pri = hmid( $keypair, 00, $prilen )
    pub = hmid( $keypair, $prilen )
else
    // error
    pause
endif

r               = new_ecc_check_pubkey( $pub )
if $r != 00
    ?
    pause
endif

pri             = 127C3DEF3718230E28B0134A41B22F1312F5676A47C9073005A77DA51FCE4101
pub             =
D72A6958CC1540E7BE420F07D3C1E51F77B5E12EAF050F9F28E6E19ACED4E99BA272A6AB31EC61C7FD0A28A972E1
B02569ED6ADF98DD2F32D9987EF59CE974F2
x               = new_ecc_get_pubkey( $pri )
if $x != $pub
    ?
    pause
endif

r               = new_ecc_check_pubkey( $pub )
```

```
if $r != 00
    ?
    pause
endif

pri1            = hrandom( $prilen )
pri2            = hrandom( $prilen )
pub1            = new_ecc_get_pubkey( $pri1 )
pub2            = new_ecc_get_pubkey( $pri2 )

x               = new_ecc_ecdh_agreement( $pri1, $pub2 )
y               = new_ecc_ecdh_agreement( $pri2, $pub1 )
if $x != $y
    ?
    pause
endif

pri1            = 86F31E7A3A3291CD81DB5A6A028CEB0A66DAD60C5C45EB0CE16AA0B6BE7BB7DC
pri2            = 52ED1AC791700ACB4D24781EB50AB3EB1BB944A1FA52169137B5DBB8A7F5AAD9
pub1            =
4F6B8DF5ED018EFEE3E3EA65E71CE7F2C8A49FB1DD7E0BD97A673B4BCC4A00A45D1884822638037477A7BFE359ED
4173F344383984521D7F2802B43AEEC46E8C
pub2            =
9CB4DA1FE75CA340187D63ACFDFCAA226E217376FAA0DB7AB496E29A697E7A3ACA04262C5537304C16C03BBCC227
64A602AE41DEF043F4936B6EB96FBCD7DAB9
stx             =
EFF2E277542ABBF7CB371376B13B6FF9643ACE7E654464E72C6543312C901BC0A002201195DFAB4801BD5F69E66B
5FAFF2231EA49ECCB1B41701A6EB5887225D
x               = new_ecc_ecdh_agreement( $pri1, $pub2 )
y               = new_ecc_ecdh_agreement( $pri2, $pub1 )
if $x != $stx
    ?
    pause
endif
if $y != $stx
    ?
    pause
endif

hash            = hrandom( $prilen )
r_s             = new_ecc_sign( $pri1, $hash ) // or new_ecc_sign( $pri1, $hash, $random )

kk              = new_ecc_verify( $pub1, $hash, $r_s )
```

```
if $kk != 00
    ?
    pause
endif

pub1            =
4F6B8DF5ED018EFEE3E3EA65E71CE7F2C8A49FB1DD7E0BD97A673B4BCC4A00A45D1884822638037477A7BFE359ED
4173F344383984521D7F2802B43AEEC46E8C
hash            = 22E4FCC99451AF042CF1DBA03964ED662FE28F6AA82DCC98AAC47590285AC4B4
r_s             =
0D9A85758DCEB848C02AC9692989A31D9CEE4FBB92BC46FD2D8872A661045A1972A05C0A789DF5174614BA58DD03
55E178346F83C73388E08B604048AB50BFA2

kk              = new_ecc_verify( $pub1, $hash, $r_s )
if $kk != 00
    ?
    pause
endif

// 计算y值功能
keypair         = new_ecc_generate_keypair()
len             = datalen( $keypair )
prilen          = big_div( $len, 03 )
if $len > 00
    pri = hmid( $keypair, 00, $prilen )
    pub = hmid( $keypair, $prilen )
else
    // error
    pause
endif
yflag           = new_sm2_get_yflag( $pub )

pointy          = new_sm2_computer_y( $yflag, $pub )
pub_y           = hmid( $pub, $prilen )
if $pub_y != $pointy
    ?
    pause
endif

pri             = 0B0A7D0D759505E56F5C38DF5D1359901CB6238D31B77B5673C94B7420A64CB6
pub             =
DA2EDEA262AFC868C70AB76A057D24A3B0784C2FCC830E375544E30643007369E224F16425487C2CFA813DE4995A
9FC0F3C592B2E3ABA4E775D1927B69ABF6C1
```

```
yflag           = new_sm2_get_yflag( $pub )

pointy          = new_sm2_computer_y( $yflag, $pub )
pub_y           = hmid( $pub, $prilen )
if $pub_y != $pointy
    ?
    pause
endif

yflag           = new_ecc_get_yflag( $pub )

pointy          = new_ecc_computer_y( $yflag, $pub )
pub_y           = hmid( $pub, $prilen )
if $pub_y != $pointy
    ?
    pause
endif

SM2_P1          =
0555AF337D1994F08E44A738BDE8A37A0066C03726B84EAD7B83F9B2F5CE95F2A705FCCC0A6703A3C29DA01FC91F
EF9D7EF52AC4D98B256B01724C91E65B8E84
r               = new_sm2_check_point( $sm2_p1 )
if $r != 00
    ?
    pause
endif

SM2_P2          =
5E7DF5818844E122238A067284749144A3DE8D180D9AE58D014C77E302C6A2FA94DF18470F8563D87FEC74C16D0A
60E13CEEEADB1F3157AA58622AD5B6D272FC
r               = new_sm2_check_point( $sm2_p2 )
if $r != 00
    ?
    pause
endif


SM2_P1_ADD_P2   =
8C52A0DDF1294926E6AC640CD591C7E9E0728BFCE7F54027BB18B012AB6EADBB20B96FCDE41160B73C6E461F8E00
B1D6403C022CDE0F7BA16344250B46C88C09
r               = new_sm2_check_point( $SM2_P1_ADD_P2 )
if $r != 00
    ?
```

```
    pause
endif

a               = new_ecc_point_add( $sm2_p1, $sm2_p2 )
if $a != $sm2_p1_add_p2
    ?
    pause
endif

b               = new_sm2_point_add( $sm2_p1, $sm2_p2 )
if $b != $sm2_p1_add_p2
    ?
    pause
endif

sm2_p1_double   =
67B0230B2872D8130F3DD64D5EDC13366B55DA9FEB6F46666DA52DBF9659125B5E41306D3F11C232861FEF844F36
E8E25829F2F1A9D59B81D43BBA93E43F2339
a               = new_sm2_point_double( $sm2_p1 )
if $a != $sm2_p1_double
    ?
    pause
endif

x               = new_ecc_check_pubkey( $a )
if $x != 00
    ?
    pause
endif

x               = new_sm2_check_pubkey( $a )
if $x != 00
    ?
    pause
endif

a               = new_ecc_point_double( $pub1 )
if $a !=
42CDA3170F0413A7FF57D47C77AA31E8463987BB92FE2A4613210BC6564FEF32C8D58CE31F66F099F531BB1E3481
4316D221FE3D04773067B305DFE0A818EE95
    ?
    pause
endif
```

```
x               = new_ecc_check_pubkey( $a )
if $x != 00
    ?
    pause
endif


x               = hrandom( $prilen )
a               = new_ecc_kp( $x, $pub1 )
x               = new_ecc_check_pubkey( $a )
if $x != 00
    ?
    pause
endif

l               = hrandom( $prilen )
k               = hrandom( $prilen )
a               = new_ecc_kp_add_lq( $k, $pub1, $l, $pub2 )
a1              = new_ecc_kp( $k, $pub1 )
a2              = new_ecc_kp( $l, $pub2 )
a3              = new_ecc_point_add( $a1, $a2 )
if $a3 != $a
    ?
    pause
endif
x               = new_ecc_check_point( $a )
if $x != 00
    ?
    pause
endif


x               = new_sm2_check_pubkey( $a )
if $x != 00
    ?
    pause
endif


a               = new_sm2_point_double( $pub1 )
if $a !=
42CDA3170F0413A7FF57D47C77AA31E8463987BB92FE2A4613210BC6564FEF32C8D58CE31F66F099F531BB1E3481
```

```
4316D221FE3D04773067B305DFE0A818EE95
    ?
    pause
endif

x               = new_sm2_check_pubkey( $a )
if $x != 00
    ?
    pause
endif


x               = hrandom( $prilen )
a               = new_sm2_kp( $x, $pub1 )
x               = new_sm2_check_pubkey( $a )
if $x != 00
    ?
    pause
endif

l               = hrandom( $prilen )
k               = hrandom( $prilen )
a               = new_sm2_kp_add_lq( $k, $pub1, $l, $pub2 )
a1              = new_sm2_kp( $k, $pub1 )
a2              = new_sm2_kp( $l, $pub2 )
a3              = new_sm2_point_add( $a1, $a2 )
if $a3 != $a
    ?
    pause
endif
x               = new_ecc_check_point( $a )
if $x != 00
    ?
    pause
endif


k               = FFF3A31F73D09FAC889D4DA2EBE9521D15D29F5F15BF50219051D42706964F17
h               =
3334C5922D7F14F3C29C52BC0F88014F2E65A52B1BC8C73A38254D99CE1B7B4FE38B126B602E41B11357DC96758C
4B465CFF95CA25B2C25BDF26B7300A43AF7A
z               =
9F84D746A33BF6ACAEAA9157D95161332AFBD00C5847001A49E00B72534CECF9B0EB79B512474B0B638AFE5C96F5
```

```
4C8016799998l4DB0071B0AB9DF5FA988E14
t               = new_ecc_ecdh_gm_map( $k, $h )
if $z != $t
    ?
    pause
endif


k               = FFF3A31F73D09FAC889D4DA2EBE9521D15D29F5F15BF50219051D42706964F17
h               =
3334C5922D7F14F3C29C52BC0F88014F2E65A52B1BC8C73A38254D99CE1B7B4FE38B126B602E41B11357DC96758C
4B465CFF95CA25B2C25BDF26B7300A43AF7A
z               =
9F84D746A33BF6ACAEAA9157D95161332AFBD00C5847001A49E00B72534CECF9B0EB79B512474B0B638AFE5C96F5
4C8016799998l4DB0071B0AB9DF5FA988E14
t               = new_sm2_ecdh_gm_map( $k, $h )
if $z != $t
    ?
    pause
endif
```

file_read_linehex

get_line_count

get_linehex_from_mem

```
x               = file_read_linehex( "所有密钥.txt" )
count           = get_line_count()

line            = get_linehex_from_mem( 00 )
line            = get_linehex_from_mem( 01 )
```

big_sqrt

```
x   = random( 20 )
y   = big_mul( $x, $x )
z   = big_sqrt( $y )
compare $x, $z
```

rockey4_smart_getcrc

```
a = rockey4_smart_getcrc( 88 d7  85 48 c1 91  77 b1 aa 89  d7 eb 80 73  d6 f6 a0 4b  3a 7b c1
```

```
55  bd dc 7b a2  53 44 e2 cb 19 d9 cf c9  32 60 b6 05  61 b1 47 74  6b 59 7f 6a  1b f7 14 4a  89
ae bf 0d  5f 03 6c aa  9a cd 42 0b b5 bf a5 91  6a f6 )
```

ed25519_generate_keypair

ed25519_get_pubkey

ed25519_sign

ed25519_verify

x25519_generate_keypair

x25519_get_pubkey

x25519

```
clear

x           = ed25519_generate_keypair()
pri         = mid( $x, 0, 32 )
pub         = mid( $x, 32 )

test        = ed25519_get_pubkey( $pri )
if $test != $pub
    ?
    pause
endif

msg         = 11223344
rs          = ed25519_sign( $pri, $msg )

ver         = ed25519_verify( $pub, $msg, $rs )
if $ver != 00
    ?
    pause
endif

// Alice's private key, a:
aprikey = 77076d0a7318a57d3c16c17251b26645df4c2f87ebc0992ab177fba51db92c2a
// Alice's public key, X25519(a, 9):
apubkey = 8520f0098930a754748b7ddcb43ef75a0dbf3a0d26381af4eba4a98eaa9b4e6a
tmp     = x25519_get_pubkey( $aprikey )
if $tmp != $apubkey
    ?
```

```
    pause
endif

// Bob's private key, b:
bprikey = 5dab087e624a8a4b79e17f8b83800ee66f3bb1292618b6fd1c2f8b27ff88e0eb
// Bob's public key, X25519(b, 9):
bpubkey = de9edb7d7b7dc1b4d35b61c2ece435373f8343c85b78674dadfc7e146f882b4f
tmp     = x25519_get_pubkey( $bprikey )
if $tmp != $bpubkey
    ?
    pause
endif

// Their shared secret, K:
share   = 4a5d9d5ba4ce2de1728e3bf480350f25e07e21c947d19e3376f09b3c1e161742
tmp     = x25519( $aprikey, $bpubkey )
if $tmp != $share
    ?
    pause
endif

tmp     = x25519( $bprikey, $apubkey )
if $tmp != $share
    ?
    pause
endif
```

shake128_hash

shake128_hash_init

shake128_hash_update

shake128_hash_dofinal

shake256_hash

shake256_hash_init

shake256_hash_update

shake256_hash_dofinal

```
x = shake128_hash( "" )
```

```
if $x != 7F9C2BA4E88F827D616045507605853ED73B8093F6EFBC88EB1A6EACFA66EF26
    ?
    pause
endif

x = shake256_hash( "" )
if $x !=
46B9DD2B0BA88D13233B3FEB743EEB243FCD52EA62B81B82B50C27646ED5762FD75DC4DDD8C0F200CB05019D67B5
92F6FC821C49479AB48640292EACB3B7C4BE
    ?
    pause
endif

x = shake128_hash_init()
x = shake128_hash_update( "" )
x = shake128_hash_dofinal( 32 )

x = shake256_hash_init()
x = shake256_hash_update( "" )
x = shake256_hash_dofinal( 32 )
```

vpn_direct_onlysend

vpn_direct_sendrecv

```
x = vpn_direct_sendrecv( 48535345 0000 0084000008 )

x = vpn_direct_onlysend( 48535345 0800 0084000008 )
x = vpn_direct_sendrecv( 99999999999999999999999999 )
```

systemtime_sub

systemtime_add

systemtime_diff

```
// 时间 - 秒数 = 新时间
x       = systemtime_sub( 20220425120000, hex( 86399 ) )
// 时间 + 秒数 = 新时间
x       = systemtime_add( 20220425120000, hex( 86399 ) )
// 得到两个时间的秒数差
x       = systemtime_diff( 20200425120000, 20060101000000 )
// 由秒数计算分钟值
minute  = big_div( $x, hex( 60 ) )
```

tlcrc

```
x       = tlcrc( 11223344 )
```

sm4_encode_gcm

sm4_decode_gcm

```
iv          = 000012345678000000000ABCD
key         = 0123456789ABCDEFFEDCBA9876543210
plaintext   =
AAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCDDDDDDDDDDDDDDDDEEEEEEEEEEEEEEEEFFFFFFFFFFFF
FFFFEEEEEEEEEEEEEEEEAAAAAAAAAAAAAAAA
aad         = FEEDFACEDEADBEEFFEEDFACEDEADBEEFABADDAD2
//    CipherText:               17F399F08C67D5EE19D0DC9969C4BB7D
//                              5FD46FD3756489069157B282BB200735
//                              D82710CA5C22F0CCFA7CBF93D496AC15
//                              A56834CBCF98C397B4024A2691233B8D
//    Authentication Tag:       83DE3541E4C2B58177E065A9BF7B62EC

x           = sm4_encode_gcm( $iv, $plaintext, $aad, $key, 10 )

mac         = right( $x, 16 )
len         = strlen( $x )
len         = sub( $len, 10 )
encryptd    = hmid( $x, 00, $len )

y           = sm4_decode_gcm( $iv, $encryptd, $aad, $key, $mac )
```

scrypt

```
x = scrypt( "", "", 10, 01, 01, 40 )
if $x !=
77D6576238657B203B19CA42C18A0497F16B4844E3074AE8DFDFFA3FEDE21442FCD0069DED0948F8326A753A0FC8
1F17E8D3E0FB2E0D3628CF35E20C38D18906
    ?
    pause
endif

// 该函数为内存型函数
```

pbkdf1_md2

pbkdf1_md4

pbkdf1_md5

pbkdf1_ripemd128

pbkdf1_ripemd160

pbkdf1_sha1

pbkdf1_sha224

pbkdf1_sha256

pbkdf1_sha384

pbkdf1_sha512

pbkdf1_sha512_224

pbkdf1_sha512_256

pbkdf1_sha3_224

pbkdf1_sha3_256

pbkdf1_sha3_384

pbkdf1_sha3_512

pbkdf1_blake2b_160

pbkdf1_blake2b_256

pbkdf1_blake2b_384

pbkdf1_blake2b_512

pbkdf1_blake2s_128

pbkdf1_blake2s_160

pbkdf1_blake2s_224

pbkdf1_blake2s_256

pbkdf1_tiger

pbkdf1_whirlpool

pbkdf1_sm3_160

pbkdf1_sm3_192

pbkdf1_sm3_256

```
? "所有函数参数类似，示例如下"
p       = "password"
s       = "salt"
c       = hex( 4096 )
len     = hex( 64 )
x       = pbkdf1_sha512( $p, $s, $c, $len )
if $x !=
412A2DC289E35A975B8374F1644995F9FDBD025CA87AE363B7A019228B7411EAE09EC0DB9DEA35CD1F0F91F55F07
750A52121DA270C180ED6BC1CC5E98401324
    ?
    pause
endif
```

pbkdf2_md2

pbkdf2_md4

pbkdf2_md5

pbkdf2_ripemd128

pbkdf2_ripemd160

pbkdf2_sha1

pbkdf2_sha224

pbkdf2_sha256

pbkdf2_sha384

pbkdf2_sha512

pbkdf2_sha512_224

pbkdf2_sha512_256

pbkdf2_sha3_224

pbkdf2_sha3_256

pbkdf2_sha3_384

pbkdf2_sha3_512

pbkdf2_blake2b_160

pbkdf2_blake2b_256

pbkdf2_blake2b_384

pbkdf2_blake2b_512

pbkdf2_blake2s_128

pbkdf2_blake2s_160

pbkdf2_blake2s_224

pbkdf2_blake2s_256

pbkdf2_tiger

pbkdf2_whirlpool

pbkdf2_sm3_160

pbkdf2_sm3_192

pbkdf2_sm3_256

```
? "所有函数参数类似，示例如下"
p       = "passwordPASSWORDpassword"
s       = "saltSALTsaltSALTsaltSALTsaltSALTsalt"
c       = hex( 4096 )
len     = hex( 64 )
x       = pbkdf2_sha512( $p, $s, $c, $len )
if $x !=
8C0511F4C6E597C6AC6315D8F0362E225F3C501495BA23B868C005174DC4EE71115B59F9E60CD9532FA33E0F75AE
FE30225C583A186CD82BD4DAEA9724A3D3B8
    ?
    pause
endif
```

hkdf_md2

hkdf_md4

hkdf_md5

hkdf_ripemd128

hkdf_ripemd160

hkdf_sha1

hkdf_sha224

hkdf_sha256

hkdf_sha384

hkdf_sha512

hkdf_sha512_224

hkdf_sha512_256

hkdf_sha3_224

hkdf_sha3_256

hkdf_sha3_384

hkdf_sha3_512

hkdf_blake2b_160

hkdf_blake2b_256

hkdf_blake2b_384

hkdf_blake2b_512

hkdf_blake2s_128

hkdf_blake2s_160

hkdf_blake2s_224

hkdf_blake2s_256

hkdf_tiger

hkdf_whirlpool

hkdf_sm3_160

hkdf_sm3_192

hkdf_sm3_256

```
? "所有函数参数类似，示例如下"
IKM     = 0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b// (22 octets)
salt    = 000102030405060708090a0b0c// (13 octets)
info    = f0f1f2f3f4f5f6f7f8f9// (10 octets)
L       = hex( 42 )
okm     = hkdf_sha256( $ikm, $salt, $info, $l )
if $okm !=
3CB25F25FAACD57A90434F64D0362F2A2D2D0A90CF1A5A4C5DB02D56ECC4C5BF34007208D5B887185865
    ?
    pause
endif
```

hmac_md2

hmac_md4

hmac_md5

hmac_ripemd128

hmac_ripemd160

hmac_sha1

hmac_sha224

hmac_sha256

hmac_sha384

hmac_sha512

hmac_sha512_224

hmac_sha512_256

hmac_sha3_224

hmac_sha3_256

hmac_sha3_384

hmac_sha3_512

hmac_blake2b_160

hmac_blake2b_256

hmac_blake2b_384

hmac_blake2b_512

hmac_blake2s_128

hmac_blake2s_160

hmac_blake2s_224

hmac_blake2s_256

hmac_tiger

hmac_whirlpool

hmac_sm3_160

hmac_sm3_192

hmac_sm3_256

hmac_shake128

hmac_shake256

```
? "所有函数参数类似，示例如下"
Key     = 0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b  //                  (20 bytes)
Data    = 4869205468657265          //        ("Hi There")

x       = hmac_sha224( $data, $key )
if $x !=  896fb1128abbdf196832107cd49df33f47b4b1169912ba4f53684b22
    ?
    pause
endif
x       = hmac_sha256( $data, $key )
if $x != b0344c61d8db38535ca8afceaf0bf12b881dc200c9833da726e9376c2e32cff7
    ?
    pause
endif
```

a3a8_fun

comp128_1

comp128_2

comp128_3

```
? "所有函数参数类似，示例如下"
// 参数1-随机数 - 8byte
// 参数2-密钥 - 16byte
// 其中 a3a8_fun 与 comp128_1 功能一致
x = a3a8_fun( 0000000000000000, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA )
if $x != 0E9FF8FF 24119D2D4ED18C00
    ?
    pause
endif
```

```
y = comp128_1( 0000000000000003, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA )
if $y != $x
    ?
    pause
endif

z = comp128_2( 0000000000000003, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA )
if $z != 9299E7F994D3288FA2E38000
    ?
    pause
endif

e = comp128_3( 0000000000000003, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA )
if $e != 9299E7F994D3288FA2E38216
    ?
    pause
endif
```

milenage_f1

milenage_f2345

milenage_f1star

milenage_f5star

milenage_compute_opc

milenage_opc_f1

milenage_opc_f2345

milenage_opc_f1star

milenage_opc_f5star

milenage_gsm

milenage_opc_gsm

```
//milenage_f1( k_16B, rand_16B, sqn_6B, amf_2B, op_16B )
//milenage_f2345( k_16B, rand_16B, op_16B )
//milenage_f1star( k_16B, rand_16B, sqn_6B, amf_2B, op_16B )
//milenage_f5star( k_16B, rand_16B, op_16B )
```

```
//milenage_compute_opc( op_16B )
//milenage_opc_f1( k_16B, rand_16B, sqn_6B, amf_2B, opc_16B )
//milenage_opc_f2345( k_16B, rand_16B, opc_16B )
//milenage_opc_f1star( k_16B, rand_16B, sqn_6B, amf_2B, opc_16B )
//milenage_opc_f5star( k_16B, rand_16B, opc_16B )
//milenage_gsm( k_16B, rand_16B, op_16B )
//milenage_opc_gsm( k_16B, rand_16B, opc_16B )

k               = dup( 16, aa )
op              = dup( 16, aa )
randx           = dup( 16, 00 )
sqn             = dup( 6, 00 )
amf             = dup( 2, 00 )

f1_mac_a        = milenage_f1( $k, $randx, $sqn, $amf, $op )
if $f1_mac_a != AFDBE3276549ACBF
    ?
    pause
endif

tmp             = milenage_f2345( $k, $randx, $op )
f2_res          = mid( $tmp, 0, 8 )
f3_ck           = mid( $tmp, 8, 16 )
f4_ik           = mid( $tmp, 24, 16 )
f5_ak           = mid( $tmp, 40 )
if $f2_res != C42F454F5FDEC1A1
    ?
    pause
endif
if $f3_ck != 701FB9D446EE896D96B0E97685E35693
    ?
    pause
endif
if $f4_ik != 173438EF8FBD185DE382834F72209F7E
    ?
    pause
endif
if $f5_ak != 600D51C33584
    ?
    pause
endif

autn            = $f5_ak $amf $f1_mac_a
```

```
if $autn != 600D51C335840000AFDBE3276549ACBF
    ?
    pause
endif

f1star_mac_s    = milenage_f1star( $k, $randx, $sqn, $amf, $op )
if $f1star_mac_s != A53F5C8E60EEF2C4
    ?
    pause
endif

f5star          = milenage_f5star( $k, $randx, $op )
if $f5star != 1A1BC57B2D4E
    ?
    pause
endif

//---------- 使用opc模式
opc             = milenage_compute_opc( $op )
f1_mac_a        = milenage_opc_f1( $k, $randx, $sqn, $amf, $opc )
if $f1_mac_a != AFDBE3276549ACBF
    ?
    pause
endif

tmp             = milenage_opc_f2345( $k, $randx, $opc )
f2_res          = mid( $tmp, 0, 8 )
f3_ck           = mid( $tmp, 8, 16 )
f4_ik           = mid( $tmp, 24, 16 )
f5_ak           = mid( $tmp, 40 )
if $f2_res != C42F454F5FDEC1A1
    ?
    pause
endif
if $f3_ck != 701FB9D446EE896D96B0E97685E35693
    ?
    pause
endif
if $f4_ik != 173438EF8FBD185DE382834F72209F7E
    ?
    pause
endif
if $f5_ak != 600D51C33584
```

```
    ?
    pause
endif

autn            = $f5_ak $amf $f1_mac_a
if $autn != 600D51C335840000AFDBE3276549ACBF
    ?
    pause
endif

f1star_mac_s    = milenage_opc_f1star( $k, $randx, $sqn, $amf, $opc )
if $f1star_mac_s != A53F5C8E60EEF2C4
    ?
    pause
endif

f5star          = milenage_opc_f5star( $k, $randx, $opc )
if $f5star != 1A1BC57B2D4E
    ?
    pause
endif

tmp             = milenage_gsm( $k, $randx, $op )
sres            = mid( $tmp, 0, 4 )
kc              = mid( $tmp, 4 )
if $sres != 9BF184EE
    ?
    pause
endif
if $kc != 1219EB023E9058DD
    ?
    pause
endif

tmp             = milenage_opc_gsm( $k, $randx, $opc )
sres            = mid( $tmp, 0, 4 )
kc              = mid( $tmp, 4 )
if $sres != 9BF184EE
    ?
    pause
endif
if $kc != 1219EB023E9058DD
    ?
```

```
    pause
endif
```

sm4_encode_ccm

sm4_decode_ccm

sm4_encode_ccm	m = a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a0 a1 a2 a3 a4 a5 a6 a7 a8 a9
	
	m1 = 03 80 14 a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a0 a1 a2 a3 a4 a5 a6 a7 a8 a9
	
	key = ca 44 ef 8d f3 25 ab b3 8d ac 37 43 dd 32 43 df
	
	nonce = 72 43 52 3C 65 17 8B 96 68 37 A3 6F
	
	//SM4CCM对数据进行对称加密结果: fb 78 40 15 24 ca 9c 2d 68 3b c4 e9 5d dc 71 b8 28 07 77 81 aa 8e 3f e8 1c e4 de 21 38 76 49 19 59 ee 87 63 e2 21 55
	// x = sm4_encode_ccm($nonce, "", $m, $key, 10)
	x = sm4_encode_ccm($nonce, "", $m1, $key, 10)
	
	cipher = mid($x, 16)
	if $cipher != FB78401524CA9C2D683BC4E95DDC71B828077781AA8E3FE81CE4DE213876491959EE8763E22155
	?
	pause
	endif
	
	返回数据分为4个部分
	● 前0x10个字节为B0
	● 外部传入的adata，与输入数据长度相同
	● 密文，与明文长度相同
	● mac，长度由外面指定

sm4_decode_ccm	len = strlen($m1) y = sm4_decode_ccm($nonce, "", $cipher, $key, 10) y = hmid($y, 00, $len) if $y != $m1 ? pause

	endif

new_rsa_calculate_d2

new_rsa_calculate_d2	newd = new_rsa_calculate_d2($module_len, $e, $p, $q, $old_d)

new_ecc_prilen_random

new_ecc_prilen_random	random = new_ecc_prilen_random ()

new_ecc_prilen

new_ecc_prilen	len = new_ecc_prilen ()

new_ecc_publen

new_ecc_publen	len = new_ecc_publen ()

UrlDownloadToFile

UrlDownloadToFile	k = UrlDownloadToFile(“https:\\a.com\\b.txt”, “c:\\aa”) k = UrlDownloadToFile(“https:\\a.com\\b.txt”, “savepath”) 或 k = UrlDownloadToFile(“https:\\a.com\\b.txt”, “savepath”, 01, 08) 第三个参数表示是否要加入一个4字节序数，第四个参数是序数

Replace_in_file

Replace_in_file	k = Replace_in_file(“c:\\b.txt”, “old”, “new”, “txt\0html\0\0”) k = Replace_in_file(“c:”, “old”, “new”)

Delete_in_file

Delete_in_file	k = Delete_in_file(“C:\\b.txt”, “start”, “end” , “txt\0html\0\0”) k = Delete_in_file(“C:”, “start”, “end”)

crc16_x25

Crc_16_x25	crc16_x25(polynomial, icv_crc, xout, data) crc = crc16_x25(8408, ffff, ffff, $data)

pkcs7_pad

pkcs7_unpad

pkcs7_pad	pkcs7_pad(input, pack_len) pkcs7_unpad(input) a = dup(7, 88) x = pkcs7_pad($a) y = pkcs7_unpad($x) x = pkcs7_pad($a, 10) y = pkcs7_unpad($x)

	x = pkcs7_pad($a, 14) y = pkcs7_unpad($x) x = 88888888888880901090909090909090 y = pkcs7_unpad($x) x = 88888888888880109090909090909090 y = pkcs7_unpad($x) x = 8888888888888010D0D0D0D0D0D0D0D0D0D0D0D y = pkcs7_unpad($x) x = 88888888888888888888888888888888 y = pkcs7_unpad($x)

pkcs12_sha1_diver_by_password

pkcs12_sha1_diver_by_password	clear ansi_password = "1234" utf16_password = ansi_utf16_big($ansi_password) utf16_password = $utf16_password 0000 salt = D9A5B98E36B81236 id = 01 iteration = hex(2000) key_len = hex(24) x = pkcs12_sha1_diver_by_password($utf16_password, $salt, $id, $iteration, $key_len) if $x != 13B0F4E8EAA3F9E840F709C140A10F6F62659B786555FD99 ? pause endif

hotp_sha1_int

hotp_sha256_int

hotp_sha1_hex

hotp_sha256_hex

hotp_sha1_int hotp_sha256_int hotp_sha1_hex	seedkey = 0102030405060708010203040506070 8 counter = 00

hotp_sha256_hex	b = hotp_sha256_int($seedkey, $counter) b = hotp_sha1_int($seedkey, $counter) b = hotp_sha256_int($seedkey, $counter, 06) b = hotp_sha1_int($seedkey, $counter, 08) b = hotp_sha256_hex($seedkey, $counter) b = hotp_sha1_hex($seedkey, $counter)

totp_sha1_int

totp_sha256_int

totp_sha1_hex

totp_sha256_hex

totp_sha1_int totp_sha256_int totp_sha1_hex totp_sha256_hex	seedkey = 01020304050607080102030405060708 counter = 00 b = totp_sha256_int($seedkey, $counter) b = totp_sha1_int($seedkey, $counter) b = totp_sha256_int($seedkey, $counter, 06) b = totp_sha1_int($seedkey, $counter, 08) b = totp_sha256_hex($seedkey, $counter) b = totp_sha1_hex($seedkey, $counter)

get_reader_name

get_reader_name	x = get_reader_name(index)

utf16_little_ansi

utf16_little_ansi	a = 4A00410056004100430004F00530020005600690072007400750061006C00200043006F006E00740061006300740020005200650061006400650072002000300 0 b = show_utf16_little_string($a) c = utf16_little_ansi($a)

strstr

strstr	a = "123456" b = "34" c = strstr($a, $b)

0.0.6.6

LoadLibrary

GetProcAddress

FreeLibrary

```
dll_handle  = loadlibrary( "UserAlgoDll.dll" )
if $dll_handle != 00
    ? "load success"
else
    ? "load dll error"
endif

fun_add     = getprocaddress( $dll_handle, "Fun_Add" )
if $fun_add != 00
    t   = dllfun( $fun_add, 00000023, 00000022 )
    if $t != 00000045
        ?
        pause
    endif
endif
x           = freelibrary( $dll_handle )
```

dllFun

dllfun	t = dllfun($fun_add, 00000023, 00000022) 最多支持12个参数 dllfun(funaddr, p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12)

new_rsa_oaep_sha1_pub_encode

new_rsa_oaep_sha224_pub_encode

new_rsa_oaep_sha256_pub_encode

new_rsa_oaep_sha384_pub_encode

new_rsa_oaep_sha512_pub_encode

new_rsa_oaep_sha1_std_decode

new_rsa_oaep_sha224_std_decode

new_rsa_oaep_sha256_std_decode

new_rsa_oaep_sha384_std_decode

new_rsa_oaep_sha512_std_decode

new_rsa_oaep_sha1_crt_decode

new_rsa_oaep_sha224_crt_decode

new_rsa_oaep_sha256_crt_decode

new_rsa_oaep_sha384_crt_decode

new_rsa_oaep_sha512_crt_decode

new_rsa_oaep_sha1_pub_encode	x1 = new_rsa_oaep_sha512_pub_encode(0800, $e, $n, 12345678, 9999) y1 = new_rsa_oaep_sha512_std_decode(0800, $n, $d, $x1) if $y1 != 12345678 ? pause endif y2 = new_rsa_oaep_sha512_crt_decode(0800, $p, $q, $dp, $dq, $qinv, $x1) if $y2 != 12345678 ? pause endif

new_rsa_pss_sha1_pub_verify

new_rsa_pss_sha224_pub_verify

new_rsa_pss_sha256_pub_verify

new_rsa_pss_sha384_pub_verify

new_rsa_pss_sha512_pub_verify

new_rsa_pss_sha1_std_sign

new_rsa_pss_sha224_std_sign

new_rsa_pss_sha256_std_sign

new_rsa_pss_sha384_std_sign

new_rsa_pss_sha512_std_sign

new_rsa_pss_sha1_crt_sign

new_rsa_pss_sha224_crt_sign

new_rsa_pss_sha256_crt_sign

new_rsa_pss_sha384_crt_sign

new_rsa_pss_sha512_crt_sign

new_rsa_pss 示例	hash = sha256_hash($plain) a = new_rsa_pss_sha256_crt_sign(0800, $p, $q, $dp, $dq, $qinv, $hash, $saltlen) k = new_rsa_pss_sha256_pub_verify(0800, $e, $n, $a, $hash, $saltlen) if $k != 00 ? pause endif b = new_rsa_pss_sha256_std_sign(0800, $n, $d, $hash, $saltlen) k = new_rsa_pss_sha256_pub_verify(0800, $e, $n, $b, $hash, $saltlen) if $k != 00 ? pause endif

new_sm9_encrypt_block_2018

new_sm9_encrypt_stream_2018

new_sm9_decrypt_block_2018

new_sm9_decrypt_stream_2018

用法	与之前不带2018的版本一致

s2k_md5

s2k_sha1

s2k_sha224

s2k_sha256

s2k_sha384

s2k_sha512

用法

```
salt            = 3031323334353637

passphrase      = "123456"      // 用户口令
iteration_count = hex( 100000 )  // 迭代次数

len             = hex( 32 )

x               = s2k_sha256( $passphrase, $salt, $iteration_count,
$len )
if $x !=
773784A602B6C81E3F092F4D7D00E17CC822D88F7360FCF2D2EF2D9D901F44B6
    ?
    pause
endif

salt            = 41 42 43 44 45 46 47 48
passphrase      = "12345678"      // 用户口令
x               = s2k_sha256( $passphrase, $salt, $iteration_count,
$len )
if $x !=
2675D6164A0D4827D1D00C7EEA620D015C00030A1CAB38B4D0DD600B27DC9630
    ?
    pause
endif
```

crc_13239_lsb

crc_13239_msb

用法	xx = 6988698269856a81 yy = $credential_private_id $xx crc = crc_13239_lsb($yy)

var_exist

用法	xx = var_exist(“key1”) 存在返回01，不存在返回00

kdf_md5

kdf_sha1

kdf_sha224

kdf_sha256

kdf_sha384

kdf_sha512

kdf_sm3

用法	结果 = kdf_sha256(first, second, keylen) xx = kdf_sha256($k, $tmp1, 30) key = mid($xx, 0, 16) mackey = mid($xx, 16, 32)

crc16_buypass

crc16_umts

crc16_modbus

crc16_xmodem

用法	x = crc16_buypass(D26B20B067BE3974288FC708F5D1421136BFDDE4FE483CF32311D037AD4CBEB4) x = crc16_umts(D26B20B067BE3974288FC708F5D1421136BFDDE4FE483CF32311D037AD4CBEB4)

```
if $x != 17a5
    ?
    pause
endif

x = crc16_umts( 3BF824F3EAF35428B5598D56B4DA6FC9118D94818C8A3C49F8 )
if $x != dadb
    ?
    pause
endif
```

用户算法 D11 接口

```
typedef BOOL ( __stdcall *pUser_Algo)    (
    IN unsigned char *p1, IN unsigned long len1,
    [
        IN unsigned char *p2, IN unsigned long len2,
        ……
        IN unsigned char *p12, IN unsigned long len12,
    ]
    OUT unsigned char *outbuf, OUT unsigned long *outlen
);
```

用户设备 D11 接口

```
typedef DWORD ( __stdcall *pUser_ListDevs)          (   OUT char  *pszDrives,    IN OUT DWORD
*pulDrivesLen,  OUT DWORD *pulDriveNum );
typedef DWORD ( __stdcall *pUser_ConnectDev)        (   IN char *pszDrive,  OUT HANDLE
*phDevice);
typedef DWORD ( __stdcall *pUser_DisconnectDev)     (   IN HANDLE hDevice);
typedef DWORD ( __stdcall *pUser_resetCard)         (   IN HANDLE hDevice,  OUT BYTE *pbAtr,
    IN OUT DWORD *pulAtrLen);
typedef DWORD ( __stdcall *pUser_PPS)               (   IN HANDLE hDevice,  IN OUT BYTE *PPS,
INT OUT DWORD *ppsLen );
typedef DWORD ( __stdcall *pUser_Transmit)          (   IN HANDLE hDevice,  IN BYTE
```

*pbCommand, IN DWORD ulCommandLen, OUT BYTE *pbOutData, IN OUT DWORD *pulOutDataLen);

具体函数名为
User_ListDevs
User_ConnectDev
User_DisconnectDev
User_resetCard
User_PPS
User_Transmit

所以函数正确必须返回0，错误返回-1。

目前实现ansi格式接口。

部分设备类型

编号	设备类型	值
1.	pcsc	0000
2.	德卡 T6 dcic32.dll	000d-0010
3.	rf1201 读卡器 非接 rf1201.dll	0012
4.	rk501 非接	0013
5.	rk506 ct	0014
6.	rk506 cl	0015
7.	潘爷 ris	0016
8.	中孚	0017
9.	rk3	0018
10.	rk4 arm	001b
11.	jubite nordic	001c
12.	rk8 hid	001d
13.	rk8 scsi	001e
14.	rockey 1	0020
15.	rockey 4 smart	0021
16.	德卡 D6u dcic32_d6u_h.dll	0020 与 2 重复了
17.	讯闪 ic94 读卡器 hfreader.dll	0023
18.	软仿真 T11	0411
19.	eclipse	04F8
20.	用户自定义接口	0900

21.	用户自定义接口 2	0901
22.	空	FFFF
23.	IIS_SMART.dll	0999
24.	Feitian_W.dll	0998
25.	CUPSDUpi.dll	0997
26.	CUPSDUpi_2.dll	0995
27.	APDULibrary.dll	0996
28.	APDULibrary.dll	0994
29.	APDULibrary.dll	0993
30.	APDULibrary.dll	0992
31.	TFSC_W.DLL	0991
32.	Unipay_Dll.dll	0988
33.	Winscard.dll	无类型，握奇的一种虚拟读卡器，要配合 server 使用，放在工具路径下，被加载后，连接 server

dummy 函数列表

略

脚本示例

SCP02 认证

```
clear

reset

00 a4 04 00 00
if sw != 9000
    ?
    pause
endif
set resp
isd = gettlv( $resp, 6f 84 )
```

```
00 a4 04 00 ( $isd )
if sw != 9000
    ?
    pause
endif
set resp

call scp02_auth( 00 )

end

scp02_auth:
    tmplen                          = $maclength        // backup current maclength and set
maclength with 8
    maclength                       = 8

    auth_level                      = getpara

    static_key                      = 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f

    host_random                     = random( 8 )

    80 50 00 00 ( $host_random )
    if sw != 9000
        ?
        ? "initial update error"
        pause
    endif
    set resp
    Key_diversification_data        = mid( $resp, 0, 10 )       // Key diversification
data
    Key_information                 = mid( $resp, 10, 2 )       // Key information
    Sequence_counter                = mid( $resp, 12, 2 )       // Sequence counter
    Card_challenge                  = mid( $resp, 14, 6 )       // Card challenge
    Card_cryptogram                 = mid( $resp, 20, 8 )       // Card cryptogram


    ? "S-ENC = 2 byte constant + 2 byte sequence counter + 12 byte 00"
    SENC_P                          = 01 82 $Sequence_counter 00000000 00000000 00000000

    ? "C-MAC = 2 byte constant + 2 byte sequence counter + 12 byte 00"
    CMAC_P                          = 01 01 $Sequence_counter 00000000 00000000 00000000
```

```
    ? "C-DEK   = 2 byte constant + 2 byte sequence counter + 12 byte 00"
    CDEK_P                          = 01 81 $Sequence_counter 00000000 00000000 00000000

    SENC_SESSION                    = 3des_encode_cbc( 0000000000000000, $SENC_P,
$static_key )

    CMAC_SESSION                    = 3des_encode_cbc( 0000000000000000, $CMAC_P,
$static_key )

    SDEK_SESSION                    = 3des_encode_cbc( 0000000000000000, $CDEK_P,
$static_key )


    ? "Host Authenticate Cryptogram = sequence counter(2Byte) + card challenge(6) + host
challenge(8) +80000000 00000000"
    host_authenticate_cryptogram    = $Sequence_counter   $Card_challenge   $host_random
80000000 00000000

    host_authenticate_cryptogram_c  = 3des_encode_cbc( 00000000,
$host_authenticate_cryptogram, $SENC_SESSION )

    host_authenticate_cryptogram_MAC    = mid( $host_authenticate_cryptogram_c, 16, 8 )

    external_auth                   = 84 82 00 $auth_level <
$host_authenticate_cryptogram_MAC >
    pack_external_auth              = fixed80( $external_auth )

    external_auth_mac               = des_3des_mac( 0000000000000000, $pack_external_auth,
$CMAC_SESSION )

    $external_auth $external_auth_mac
    if sw != 9000
        ?
        ? "external error"
        pause
    endif

    maclength                       = $tmplen       // restore mac length
return
```

国密计算

有些国密算法是只能由硬件执行的，为此需要用带国密算法的芯片做一个小设备，借助设备，就可以实现国密计算了

```
clear

// reset

load_script "guomi_calc_fun.txt"

call open_calculator

ssf33_std_plain      = ACB331D20168304EE6AF06FC490737A9
ssf33_std_cipher     = B838F83CE18BD8B670E076185ED428AE
ssf33_key            = A593055A4827368CDB6D96417C03BE67

// ssf33_encode_ecb
// 参数1，数据
// 参数2，密钥
// 返回结果未设置由调用者自行设置

xxx                  = call ssf33_encode_ecb( $ssf33_std_plain, $ssf33_key )
if $xxx != $ssf33_std_cipher
    ?
    pause
endif

yyy                  = call ssf33_decode_ecb( $xxx, $ssf33_key )
if $yyy != $ssf33_std_plain
    ?
    pause
endif

xxx                  = call ssf33_encode_cbc( 00, $ssf33_std_plain, $ssf33_key )
if $xxx != $ssf33_std_cipher
    ?
    pause
endif

yyy                  = call ssf33_decode_cbc( 00, $xxx, $ssf33_key )
if $yyy != $ssf33_std_plain
    ?
```

```
    pause
endif

xxx                 = call ssf33_mac( 1122,
3344334433443344334433443344334433443344334433443344334433443344334433443344334433443344, 5566 )

// sm1
sm1_std_plain       = 112233445566778899aabbccddeeff00
sm1_std_cipher      = E43EA994EEF341F9A04DCDB8298E119B
sm1_key             = 0102030405060708090a0b0c0d0e0f10

xxx                 = call sm1_encode_ecb( $sm1_std_plain, $sm1_key )
if $xxx != $sm1_std_cipher
    ?
    pause
endif

yyy                 = call sm1_decode_ecb( $xxx, $sm1_key )
if $yyy != $sm1_std_plain
    ?
    pause
endif

xxx                 = call sm1_encode_cbc( 00, $sm1_std_plain, $sm1_key )
if $xxx != $sm1_std_cipher
    ?
    pause
endif

yyy                 = call sm1_decode_cbc( 00, $xxx, $sm1_key )
if $yyy != $sm1_std_plain
    ?
    pause
endif

xxx                 = call sm1_mac( 1122,
3344334433443344334433443344334433443344334433443344334433443344334433443344334433443344, 5566 )
```