## COS层

**说明：**见下面黄色标记部分

* SD\_CmdProc

b1为0x00或0x01时，代表此时数据为扩展模式

b1为0x00时，按正常数据格式处理，此时LC含b1字节共3字节长度，LE两字节长度

当b1为0x01时，标明此时LE长度大于256，占两个字节长度，LC大小小于256长度，上位机做了补256长度操作，cos计算LC需减去256.

* CMD\_ExtRSAPubKeyOPT、CMD\_ExtRSAPriKeyOPT、CMD\_ExtRSAEnDecrypt

函数内通过pInData以指针方式，复用全局变量地址

实际使用中会存在pInData指向地址非4字节对齐，导致数据传递运算出错

**修改**：指针改为局部变量数组形式

### void SD\_CmdProc(UINT32 nArg, UINT8 \*pBuf, UINT8 nFlgDir)

{

UINT16 nIndex = 0;

//写命令数据

if (nFlgDir == DIR\_WR)

{

UINT16 nAPDULen = 0;

gApduResp.SW1 = RT\_OK;

//检查数据包标识

if (pBuf[0] != 0xEF || pBuf[1] != 0x01)

{

DBG\_INFO("No Syno COS cmd flag! pBuf[0]=0x%x,pBuf[1]=0x%x", pBuf[0], pBuf[1]);

gApduResp.SW1 = RT\_FAI;

gApduResp.LEN = 0;

return;

}

//检查数据包类型

if (pBuf[2] != 0xE1)

{

DBG\_INFO("No Syno TF COS cmd flag! pBuf[2]=0x%x", pBuf[2]);

gApduResp.SW1 = RT\_COMMAND\_ERROR;

gApduResp.LEN = 0;

return;

}

//检查数据包类型

if (pBuf[3] != 0x00)

{

DBG\_INFO("No Syno TF COS cmd flag! pBuf[2]=0x%x", pBuf[2]);

gApduResp.SW1 = RT\_PARAM\_ERROR;

gApduResp.LEN = 0;

return;

}

//读取APDU数据长度

nIndex = 4;

nAPDULen = (pBuf[nIndex+1]<<8) | pBuf[nIndex]; //small endian mode

nIndex += 2;

DBG\_INFO("APDU total length LC=0x%x", nAPDULen);

//读取APDU头

memset(&gApduHeader, 0, sizeof(APDUHEADER\_S));

memcpy(&gApduHeader, pBuf + nIndex, sizeof(APDUHEADER\_S));

nIndex += sizeof(APDUHEADER\_S);

DBG\_INFO("gApduHeader.CLA = 0x%x, INS = 0x%x, P1=0x%x, P2=0x%x!", gApduHeader.CLA, gApduHeader.INS, gApduHeader.P1, gApduHeader.P2);

//根据长度判断APDU结构类型

//Case 1

if (nAPDULen == sizeof(APDUHEADER\_S))

{

gApduBody.LC = 0;

gApduBody.LE = 0;

DBG\_INFO("APDU is case 1!");

}

else

{

UINT8 b1 = 0;

UINT16 b2b3 = 0;

//B1

b1 = (BYTE)pBuf[nIndex];

nIndex += 1;

//Case 2S

if (nAPDULen == sizeof(APDUHEADER\_S) + 1)

{

gApduBody.LC = 0;

gApduBody.LE = b1;

if(gApduBody.LE == 0)

gApduBody.LE = 256;

DBG\_INFO("APDU is case 2S:LC=0x%x, LE=0x%x", gApduBody.LC, gApduBody.LE);

}

else if ((b1 != 0x00)&&(b1 != 0x01))

{

//Case 3S

if (nAPDULen == sizeof(APDUHEADER\_S) + 1 + b1)

{

gApduBody.LC = b1;

gApduBody.DATA = pBuf + nIndex;

nIndex += gApduBody.LC;

gApduBody.LE = 0;

DBG\_INFO("APDU is case 3S:LC=0x%x, LE=0x%x", gApduBody.LC, gApduBody.LE);

}

//Case 4S

else if (nAPDULen == sizeof(APDUHEADER\_S) + 2 + b1)

{

gApduBody.LC = b1;

gApduBody.DATA = pBuf + nIndex;

nIndex += gApduBody.LC;

gApduBody.LE = pBuf[nIndex]; //

if(gApduBody.LE == 0)

gApduBody.LE = 256;

nIndex += 1;

DBG\_INFO("APDU is case 4S:LC=0x%x, LE=0x%x", gApduBody.LC, gApduBody.LE);

}

else

{

DBG\_INFO("APDU is an unknown case!");

DBG\_INFO(pBuf, nAPDULen);

gApduResp.SW1 = RT\_COMMAND\_ERROR;

gApduResp.LEN = 0;

return;

}

}

else //B1=0

{

b2b3 = (pBuf[nIndex]<<8) | pBuf[nIndex+1];

nIndex += 2;

//Case 2E

if (nAPDULen == sizeof(APDUHEADER\_S) + 3) //[Not work] As the program word design, this situation not exist, when le length 3 the lc length is 3.

{

gApduBody.LC = b1;

gApduBody.LE = b2b3;

if(gApduBody.LE == 0)

gApduBody.LE = 65536;

nIndex += 2;

DBG\_INFO("APDU is case 2E:LC=0x%x, LE=0x%x", gApduBody.LC, gApduBody.LE);

}

//Case 3E

else if (nAPDULen == sizeof(APDUHEADER\_S) + 3 + b2b3)

{

gApduBody.LC = b2b3;

gApduBody.DATA = pBuf + nIndex;

nIndex += gApduBody.LC;

gApduBody.LE = 0;

DBG\_INFO("APDU is case 3E:LC=0x%x, LE=0x%x", gApduBody.LC, gApduBody.LE);

}

//Case 4E

else if (nAPDULen == sizeof(APDUHEADER\_S) + 5 + b2b3)

{

if(b1 == 0x01)

{

gApduBody.LC = b2b3 - 256;

}

else

gApduBody.LC = b2b3;

gApduBody.DATA = pBuf + nIndex;

nIndex += gApduBody.LC;

gApduBody.LE = (pBuf[nIndex]<<8) | pBuf[nIndex+1];

if(gApduBody.LE == 0)

gApduBody.LE = 65536;

nIndex += 2;

DBG\_INFO("APDU is case 4E:LC=0x%x, LE=0x%x", gApduBody.LC, gApduBody.LE);

}

else

{

DBG\_INFO("APDU is an unknown case!");

DBG\_INFO(pBuf, nAPDULen);

gApduResp.SW1 = RT\_COMMAND\_ERROR;

gApduResp.LEN = 0;

return;

}

}

}

//检查数据长度是否有效

if (gApduBody.LC > SD\_CMDBUF\_LEN)

{

DBG\_ERR("CMD data length gApduBody.LC is wrong, it's max value is 0x%x!", SD\_CMDBUF\_LEN);

gApduResp.SW1 = RT\_PARAM\_ERROR;

gApduResp.LEN = 0;

return;

}

//检查COS是否初始化完成

if (!gSystemInited)

{

DBG\_ERR("System has not initialized!");

gApduResp.SW1 = RT\_FAI;

gApduResp.LEN = 0;

return;

}

//处理命令

if ((gApduHeader.CLA&0xFF) == 0xFE) //

{

DoAS5xxCMDEx();

DBG\_INFO("DoAS5xxCMDEx(0x%02X) is returned", gApduHeader.INS);

}

else//COS CMD

{

DoAS5xxCMD();

DBG\_INFO("DoAS5xxCMD(0x%02X) is returned", gApduHeader.INS);

}

}

//读响应数据

else

{

//响应数据包头

pBuf[0] = 0xEF;

pBuf[1] = 0x01;

pBuf[2] = 0xE1;

pBuf[3] = gApduResp.SW1;

pBuf[4] = 0;

pBuf[5] = 0;

if(pBuf[3] == RT\_OK)

{

pBuf[4] = (gApduResp.LEN+2) & 0x00ff;

pBuf[5] = ((gApduResp.LEN+2) >> 8) & 0x00ff; //small endian mode

nIndex = 6;

//复制响应数据

if (gApduResp.LEN > 0)

{

memcpy(pBuf + nIndex, gApduResp.DATA, gApduResp.LEN);

nIndex += gApduResp.LEN;

}

//SW

pBuf[nIndex++] = (gApduResp.SW >> 8) & 0x00ff;

pBuf[nIndex++] = gApduResp.SW & 0x00ff;

}

DBG\_INFO("gApduResp.SW = 0x%x, len = 0x%x", gApduResp.SW, gApduResp.LEN);

}

}

### void CMD\_ExtRSAPubKeyOPT()

{

UINT16 nRet = SAR\_OK;

UINT32 uDataLen = 0;

UINT32 nByteLen = 0;

UINT32 tmpIndex = 0;

UINT32 rLen = 0;

// UINT8\* pInData = NULL;

UINT8 InData[256];

R\_RSA\_PUBLIC\_KEY RsaPubKey = {0};

//以下所有变量，必须按四字节对齐

\_\_align(4) UINT8 n[256], e[4];

DBG\_FUNC\_BEGIN;

RsaPubKey.bits = (gApduBody.DATA[3]<<24)|(gApduBody.DATA[2]<<16)|(gApduBody.DATA[1]<<8)|gApduBody.DATA[0];

RsaPubKey.modulus = n;

RsaPubKey.exponent = e;

tmpIndex += 4;

//外来公钥数据

nByteLen = RsaPubKey.bits/8;

memcpy((UINT8\*)RsaPubKey.modulus, gApduBody.DATA + tmpIndex, nByteLen);

tmpIndex += nByteLen;

memcpy((UINT8\*)RsaPubKey.exponent, gApduBody.DATA + tmpIndex, 4);

tmpIndex += 4;

//输入数据

uDataLen = (gApduBody.DATA[tmpIndex+3]<<24) | (gApduBody.DATA[tmpIndex+2]<<16) | (gApduBody.DATA[tmpIndex+1]<<8) | gApduBody.DATA[tmpIndex];

tmpIndex += 4;

// pInData = gApduBody.DATA + tmpIndex;

memcpy(InData, gApduBody.DATA + tmpIndex, uDataLen);

//使用公钥处理数据

nRet = RSA\_PubKeyOpt(gApduResp.DATA, &rLen, InData, uDataLen, &RsaPubKey);

if (nRet != RT\_OK)

{

DBG\_ERR("RSA\_PubKeyOpt() failed! nRet = 0x%x", nRet);

SetCmdResult(SAR\_SIGNVERYFIERR, 0);

return;

}

DBG\_FUNC\_END;

SetCmdResult(SAR\_OK, rLen);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Subroutine: CMD\_ExtRSAPriKeyOPT()

\* Function: Extern rsa pri key opt

\* Author: Syno Common

\* Date: 2015.06.20

\* Version: V1.0

\* ModifyRecord:

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### void CMD\_ExtRSAPriKeyOPT()

{

UINT16 nRet = SAR\_OK;

UINT32 uDataLen = 0;

UINT32 nByteLen = 0;

UINT32 nPrimeLen = 0;

UINT32 tmpIndex = 0;

UINT32 rLen = 0;

// UINT8\* pInData = NULL;

UINT8 InData[256];

R\_RSA\_PRIVATE\_KEY RsaPriKey = {0};

//以下所有变量，必须按四字节对齐

\_\_align(4) UINT8 n[256], e[4], d[256], p[128], q[128], dp[128], dq[128], qinv[128];

DBG\_FUNC\_BEGIN;

//读取私钥数据

RsaPriKey.bits = (gApduBody.DATA[3]<<24) | (gApduBody.DATA[2]<<16) | (gApduBody.DATA[1]<<8) | gApduBody.DATA[0];

RsaPriKey.modulus = n;

RsaPriKey.publicExponent = e;

RsaPriKey.exponent = d;

RsaPriKey.prime[0] = p;

RsaPriKey.prime[1] = q;

RsaPriKey.primeExponent[0] = dp;

RsaPriKey.primeExponent[1] = dq;

RsaPriKey.coefficient = qinv;

tmpIndex += 4;

nByteLen = RsaPriKey.bits / 8;

nPrimeLen = nByteLen / 2;

//change @ 0720 +MAX\_RSA\_MODULUS\_LEN-nByteLen

memcpy((UINT8\*)RsaPriKey.modulus, gApduBody.DATA + tmpIndex, nByteLen);

tmpIndex += nByteLen;

memcpy((UINT8\*)RsaPriKey.publicExponent, gApduBody.DATA + tmpIndex, 4);

tmpIndex += 4;

memcpy((UINT8\*)RsaPriKey.exponent, gApduBody.DATA + tmpIndex, nByteLen);

tmpIndex += nByteLen;

memcpy((UINT8\*)RsaPriKey.prime[0], gApduBody.DATA + tmpIndex, nPrimeLen);

tmpIndex += nPrimeLen;

memcpy((UINT8\*)RsaPriKey.prime[1], gApduBody.DATA + tmpIndex, nPrimeLen);

tmpIndex += nPrimeLen;

memcpy((UINT8\*)RsaPriKey.primeExponent[0], gApduBody.DATA + tmpIndex, nPrimeLen);

tmpIndex += nPrimeLen;

memcpy((UINT8\*)RsaPriKey.primeExponent[1], gApduBody.DATA + tmpIndex, nPrimeLen);

tmpIndex += nPrimeLen;

memcpy((UINT8\*)RsaPriKey.coefficient, gApduBody.DATA + tmpIndex, nPrimeLen);

tmpIndex += nPrimeLen;

//输入数据

uDataLen = (gApduBody.DATA[tmpIndex+3]<<24) | (gApduBody.DATA[tmpIndex+2]<<16) | (gApduBody.DATA[tmpIndex+1]<<8) | gApduBody.DATA[tmpIndex];

tmpIndex += 4;

// pInData = gApduBody.DATA + tmpIndex;

memcpy(InData, gApduBody.DATA + tmpIndex, uDataLen);

//私钥运算

nRet = RSA\_PriKeyOpt(gApduResp.DATA, &rLen, InData, uDataLen, &RsaPriKey);

if (nRet != RT\_OK)

{

DBG\_ERR("RSA\_PriKeyOpt() failed! nRet = 0x%x", nRet);

SetCmdResult(SAR\_ASYMM\_ENC\_FAIL, 0);

return;

}

DBG\_FUNC\_END;

SetCmdResult(SAR\_OK, rLen);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Subroutine: CMD\_ExtRSAEnDecrypt()

\* Function: Extern rsa key opt(encrypt or decrypt)

\* Author: Syno Common

\* Date: 2015.06.20

\* Version: V1.0

\* ModifyRecord:

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### void CMD\_ExtRSAEnDecrypt()

{

UINT16 nRet = SAR\_OK;

UINT16 nStatus = SAR\_OK;

UINT16 AlgID = 0;

UINT16 appid = 0;

UINT16 containerid = 0;

UINT16 index = 0;

UINT16 keybitlen = 0;

UINT16 nByteLen = 0;

UINT16 nPrimeLen = 0;

UINT32 datalen = 0;

UINT32 rlen = 0;

// UINT8 \*data = NULL;

UINT8 data[256];

UINT8 \*OutData = NULL;

UINT8\* RSAKeydata = NULL;

R\_RSA\_KEYPAIR RSAKey = {0};

APPSTATE\* AppState = NULL;

//以下所有变量，必须按四字节对齐

\_\_align(4) UINT8 n[256], e[4], d[256], p[128], q[128], dp[128], dq[128], qinv[128];

DBG\_FUNC\_BEGIN;

//检查数据长度

datalen = gApduBody.LC - 6;

if(datalen > 512)

{

DBG\_ERR("The data length is wrong! Data length = 0x%x, gApduBody.LC = 0x%x", datalen, gApduBody.LC);

SetCmdResult(SAR\_INDATALENERR, 0);

return;

}

// data = gApduBody.DATA;

memcpy(data, gApduBody.DATA, datalen);

//app id 和 container id 以及操作flag在数据之后！

appid = ((gApduBody.DATA[datalen + 1]<<8) | gApduBody.DATA[datalen]);

containerid = ((gApduBody.DATA[datalen + 3]<<8) | gApduBody.DATA[datalen + 2]);

AlgID = ((gApduBody.DATA[datalen + 5]<<8) | gApduBody.DATA[datalen + 4]);

DBG\_INFO("appid = 0x%x, containerid = 0x%x, AlgID = 0x%x", appid, containerid, AlgID);

//检查应用状态

AppState = GetAppStateInfoById(appid);

if (!AppState)

{

DBG\_ERR("This application ID is not exist! appid = 0x%x", appid);

SetCmdResult(SAR\_APPLICATION\_NOT\_EXISTS, 0);

return;

}

//检查安全状态

gCurSAStatus = AppState->AppState;

nStatus = IsHasControl(SA\_USERLOGIN, SA\_SOLOGIN, CON\_ALL);

if (nStatus != SAR\_OK)

{

DBG\_ERR("The user has not logined!");

SetCmdResult(nStatus, 0);

return;

}

//检查容器ID

if (SAR\_OK != IsContainerIdValid(appid, containerid))

{

DBG\_ERR("This container ID is not exist! containerid = 0x%x", containerid);

SetCmdResult(SAR\_CONTAINERID\_NOT\_EXISTS,0);

return;

}

//从文件读取密钥数据

nRet = FS\_SelectFileByCluster(containerid,containerid);

if(gApduHeader.P1)

{

DBG\_INFO("Sign key is selected!");

nRet = FS\_SelectFileByID(SIGNKEYID);

}

else

{

DBG\_INFO("Exch key is selected!");

nRet = FS\_SelectFileByID(EXKEYID);

}

if (nRet != RT\_OK)

{

DBG\_ERR("FS\_SelectFileByID() failed! nRet = 0x%x", nRet);

SetCmdResult(SAR\_FILE\_NOT\_EXIST, 0);

return;

}

RSAKeydata = gApduBody.DATA + datalen + (4 - datalen%4) + 8; //确保取址4字节对齐；

nRet = FS\_ReadFile(RSAKeydata, 0, 2);

keybitlen = (RSAKeydata[1]<<8) | RSAKeydata[0];

nByteLen = keybitlen / 8;

nPrimeLen = keybitlen / 16;

nRet |= FS\_ReadFile(RSAKeydata, 0, nByteLen\*2 + nPrimeLen\*5 + 6);

if (nRet != RT\_OK)

{

DBG\_ERR("FS\_ReadFile() failed! nRet = 0x%x", nRet);

SetCmdResult(SAR\_INDATAERR, 0);

return;

}

RSAKey.modulus = n;

RSAKey.publicExponent = e;

RSAKey.exponent = d;

RSAKey.prime[0] = p;

RSAKey.prime[1] = q;

RSAKey.primeExponent[0] = dp;

RSAKey.primeExponent[1] = dq;

RSAKey.coefficient = qinv;

index = 0;

memcpy((UINT8\*)&RSAKey.bits, RSAKeydata, 2);

index += 2;

memcpy(RSAKey.modulus, RSAKeydata + index, nByteLen);

index += nByteLen;

memcpy(RSAKey.publicExponent, RSAKeydata + index, 4);

index += 4;

memcpy(RSAKey.exponent, RSAKeydata + index, nByteLen);

index += nByteLen;

memcpy(RSAKey.prime[0], RSAKeydata + index, nPrimeLen);

index += nPrimeLen;

memcpy(RSAKey.prime[1], RSAKeydata + index, nPrimeLen);

index += nPrimeLen;

memcpy(RSAKey.primeExponent[0], RSAKeydata + index, nPrimeLen);

index += nPrimeLen;

memcpy(RSAKey.primeExponent[1], RSAKeydata + index, nPrimeLen);

index += nPrimeLen;

memcpy(RSAKey.coefficient, RSAKeydata + index, nPrimeLen);

index += nPrimeLen;

OutData = RSAKeydata;

if (AlgID) //AlgID=true 执行加密操作

{

if(gApduHeader.P2)

{

DBG\_INFO("Use public key to encrypt data!");

nRet = RSA\_PublicEncrypt(OutData, &rlen, data, datalen, (R\_RSA\_PUBLIC\_KEY\*)&RSAKey);

}

else

{

DBG\_INFO("Use private key to encrypt data!");

nRet = RSA\_PrivateEncrypt(OutData, &rlen, data, datalen, &RSAKey);

}

}

else //AlgID=false 执行解密操作

{

if(gApduHeader.P2)

{

DBG\_INFO("Use public key to decrypt data!");

nRet = RSA\_PublicDecrypt(OutData, &rlen, data, datalen, (R\_RSA\_PUBLIC\_KEY\*)&RSAKey);

}

else

{

DBG\_INFO("Use private key to decrypt data!");

nRet = RSA\_PrivateDecrypt(OutData, &rlen, data, datalen, &RSAKey);

}

}

if (nRet != RT\_OK)

{

DBG\_ERR("RSA operation failed! nRet = 0x%x", nRet);

SetCmdResult(SAR\_ASYMM\_DEC\_FAIL, 0);

return;

}

memcpy(gApduResp.DATA, OutData, rlen);

DBG\_FUNC\_END;

SetCmdResult(SAR\_OK, rlen);

}

## 上位机（PC）：

**CMD\_Transmit**和**CMD\_TransmitEx**函数，关于LC、LE长度处理调整部分

/\* 拷贝数据长度Lc和数据，如果有 \*/

usLc = (USHORT)ulInLen;

usLe = pulOutLen ? (USHORT)\*pulOutLen : 0;

if (usLc > 0)

{

if ((usLc < 256) && (usLe <= 256)) //change by comon @ 20151224

{

acApduBuf[ulApduLen] = usLc;

ulApduLen += 1;

}

else if ((usLc < 256) && (usLe > 256))

{

acApduBuf[ulApduLen] = 0x00;

usLc += 256; // when le > 256, lc must be 3 byte length (extension mode) @20151224

acApduBuf[ulApduLen + 1] = (usLc >> 8) & 0x00ff;

acApduBuf[ulApduLen + 2] = usLc & 0x00ff;

ulApduLen += 3;

}

else

{

acApduBuf[ulApduLen] = 0x00;

acApduBuf[ulApduLen + 1] = (usLc >> 8) & 0x00ff;

acApduBuf[ulApduLen + 2] = usLc & 0x00ff;

ulApduLen += 3;

}

memcpy(acApduBuf + ulApduLen, pucIn, usLc);

ulApduLen += usLc;

}

else if ((usLc == 0) && (usLe > 256))

{

acApduBuf[ulApduLen] = 0x00;

//usLc += 256; // when le > 256, lc must be 3 byte length (extension mode) @20151224

acApduBuf[ulApduLen + 1] = (usLc >> 8) & 0x00ff;

acApduBuf[ulApduLen + 2] = usLc & 0x00ff;

ulApduLen += 3;

}

/\* 拷贝Le，如果有 \*/

if (usLe > 0)

{

if (usLe < 256)

{

acApduBuf[ulApduLen] = usLe;

ulApduLen += 1;

}

else if (usLe == 256)

{

acApduBuf[ulApduLen] = 0x00;

ulApduLen += 1;

}

else

{

if (usLe == 65536)

{

usLe = 0;

}

acApduBuf[ulApduLen] = (usLe >> 8) & 0x00ff;

acApduBuf[ulApduLen + 1] = usLe & 0x00ff;

ulApduLen += 2;

}

//if (usLc == 0)

//{

// acApduBuf[ulApduLen] = 0x00;

// ulApduLen += 1;

//}

//acApduBuf[ulApduLen] = (usLe >> 8) & 0x00ff;

//acApduBuf[ulApduLen+1] = usLe & 0x00ff;

//ulApduLen += 2;

}

ulRet = APDUInterface(hHandle, acApduBuf, ulApduLen, acApduBuf, &ulRespLen);

/\* 检查APDU响应包中的SW \*/

## 上位机（Android）

static int sd\_interface\_do\_requst(int fd,TPCCmd cmd,BYTE \*pInBuf,UINT nInLEn,BYTE\*pOutBuf,UINT\*nOutLEn)

{

int err,value;

char \* pbuff = NULL;

char \* pdata = NULL;

char \*paligned = NULL;

char length\_l,length\_h;

char length\_mode = 0x00; //change by comon @ 20151225, when lc\_length need to add 256 by outside, the lengh\_mode = 0x01.

unsigned int status = 0;

unsigned int rxlen = 0;

unsigned short buff\_length = 0;

unsigned short apdu\_length = 0;

unsigned char lc\_length,le\_length;

unsigned int bytes,offset,packets,totalbytes,packetbytes;

static const struct timespec wait\_time = { .tv\_sec = 0, .tv\_nsec = 250000000L, };

LOGI("@$@ sd Interface 1\n");

if(fd <= 0){

LOGE("SD Interface:Hardware error,not opened!");

return SAR\_FILEERR;

}

if(cmd.LC > 0)

{

if(cmd.LC < 256)

{

lc\_length = 1;

}

else

lc\_length = 3;

}

else

lc\_length = 0;

if(cmd.LE > 0)

{

if(cmd.LE <= 256)

le\_length = 1;

else

le\_length = 3;

}

else

le\_length = 0;

if((le\_length == 3) && (lc\_length != 3))

{

cmd.LC += 256;

length\_mode = 0x01;

lc\_length = 3;

}

if(lc\_length == 3 && le\_length != 0) //if(lc\_length == 3 && le\_length == 3) change by comon @ 20151224, if lc\_length =3 and le\_length != 0, le\_length should be 2 byte length.

apdu\_length = 4 + lc\_length + le\_length - 1 + cmd.LC;

else

apdu\_length = 4 + lc\_length + le\_length + cmd.LC;

length\_l = (apdu\_length >> 0) & 0xff;

length\_h = (apdu\_length >> 8) & 0xff;

buff\_length = apdu\_length + TRANSTER\_HEAD\_LENGTH; //512 \* ((nInLen / 512) + 2); //

if(buff\_length > FILE\_FIFO\_LENGTH){

LOGE("SD Interface:Unsupported ,FIFO overflow! You will need to divided "

"the data into 4K per packets to send in upper layer ");

return SAR\_LENGTH\_ERR;

}

pbuff = (unsigned char \*)memalign(512,BLOCK\_ALIGNED(buff\_length));

if(!pbuff){

LOGE("SD Interface: malloc requst buff error.");

return SAR\_MEMORYERR;

}

bzero(pbuff,BLOCK\_ALIGNED(buff\_length));

pdata = pbuff;

/\*transfer head 6 Byte (2byte PID + 1 byte CMD + 1 byte PARAM + 2byte LC)\*/

\*pdata++ = 0xef; /\*2Byte PID\*/

\*pdata++ = 0x01;

\*pdata++ = 0xe1; /\*1Byte TF æ¶“æ’¶æ•¤é›æˆ’æŠ¤\*/

\*pdata++ = 0x00; /\*1Byte PARAM reseved\*/

\*pdata++ = length\_l;/\*2Byte LC æµ£åº¡ç“§é‘ºå‚šæ¹ªé“å¶‰ç®ç€›æ¥„å¦­éå¶…æ‚—\*/

\*pdata++ = length\_h;

/\*APDU packeg ......\*/

\*pdata++ = cmd.CLA;

\*pdata++ = cmd.INS;

\*pdata++ = cmd.P1;

\*pdata++ = cmd.P2;

LOGI("@$@ sd Interface 3\n");

if(cmd.LC){

if(pInBuf == NULL){

LOGE("NO INPUTE DATA!");

return SAR\_FAIL;

}

if(lc\_length == 1){

\*pdata++ = cmd.LC;

}else{

/\*APDU æ¥‚æ¨ºç“§é‘ºå‚šæ¹ªé“å¶„ç¶†ç€›æ¥„å¦­é¦ã„¥æ‚—\*/

\*pdata++ = length\_mode;

\*pdata++ = (cmd.LC >> 8) & 0xff;

\*pdata++ = (cmd.LC >> 0) & 0xff;

}

memcpy(pdata,pInBuf,cmd.LC);

pdata += cmd.LC;

}

if(cmd.LE){

if(pOutBuf == NULL){

LOGE("OUT BUFFER EMPTY!");

return SAR\_FAIL;

}

if(le\_length == 1)

{

if(cmd.LE == 256 ){

\*pdata++ = 0;

}else

\*pdata++ = cmd.LE;

}

else if(le\_length == 3)

{

if(cmd.LE == 65536)

{

\*pdata++ = 0;

\*pdata = 0;

}

else

{

\*pdata++ = (cmd.LE >> 8) & 0xff;

\*pdata = (cmd.LE >> 0) & 0xff;

}

}

}

/\*\*

\* OK,Now data in memory layout is as follows:

\* PID + CMD + PARAM + LC + APDU(CLA + INS + P1 + P2 + Lc + nDATA + Le)éŠ†ï¿½

\*/

LOGI("@$@ sd Interface 4\n");

lseek(fd,0,SEEK\_SET);

err = pwrite(fd,pbuff,BLOCK\_ALIGNED(buff\_length),0);

if (err < 0)

{

value = err;

LOGE("SD Interface: %s ",strerror(errno));

goto done;

}

if(!nOutLEn){

LOGE("SD interface : Input Param error!");

return SAR\_LENGTH\_ERR;

}

nanosleep(&wait\_time, NULL);

fsync(fd);

LOGI("@$@ sd Interface 5\n");

/\*make sure write done!\*/

/\*

\*1éŠ†ä½¸ç¶‹le æ¶“ï¿½0 éƒï¿½: ç’‡è¯²å½‡æ¶“ã‚„é‡œç€›æ¥„å¦­é¨å‰†w1sw2.

\*2éŠ†ä½¸ç¶‹le < 4K-head :

\*/

\*nOutLEn = 0;