```
#include <openssl/evp.h>
#include <limits.h>
#include "tpm_tspi.h"
#include "tpm_utils.h"
#include "tpm_seal.h"

static void help(const char *aCmd)
{
    logCmdHelp(aCmd);
    logCmdOption("-i, --infile FILE",
                 _
                 ("Filename containing key to seal. Default is STDIN."));
    logCmdOption("-o, --outfile FILE",
                 _
                 ("Filename to write sealed key to.    Default is STDOUT."));
    logCmdOption("-p, --pcr NUMBER",
                 _
                 ("PCR to seal data to.    Default is none.    This option can be specified
multiple times to choose more than one PCR."));
    logCmdOption("-z, --well-known", _("Use TSS_WELL_KNOWN_SECRET as
the SRK secret."));
    logCmdOption("-u, --unicode", _("Use TSS UNICODE encoding for the SRK
password to comply with applications using TSS popup boxes"));

}

static char in_filename[PATH_MAX] = "", out_filename[PATH_MAX] = "";
static TSS_HPCRS hPcrs = NULL_HPCRS;
static TSS_HTPM hTpm;
static UINT32 selectedPcrs[24];
static UINT32 selectedPcrsLen = 0;
static BOOL passUnicode = FALSE;
static BOOL isWellKnown = FALSE;
TSS_HCONTEXT hContext = 0;

static int parse(const int aOpt, const char *aArg)
{
    int rc = -1;

    switch (aOpt) {
    case 'i':
        if (aArg) {
            strncpy(in_filename, aArg, PATH_MAX);
            rc = 0;
```

```
            }
            break;
        case 'o':
            if (aArg) {
                strncpy(out_filename, aArg, PATH_MAX);
                rc = 0;
            }
            break;
        case 'p':
            if (aArg) {
                selectedPcrs[selectedPcrsLen++] = atoi(aArg);
                rc = 0;
            }
            break;
        case 'u':
            passUnicode = TRUE;
            rc = 0;
            break;
        case 'z':
            isWellKnown = TRUE;
            rc = 0;
            break;
        default:
            break;
        }
        return rc;

}

int main(int argc, char **argv)
{

        TSS_HKEY hSrk, hKey;                    //秘钥对象句柄
        TSS_HENCDATA hEncdata;                  //加密数据对象句柄
        TSS_HPOLICY hPolicy;                    //策略对象句柄
        int iRc = -1;
        struct option opts[] =
              { {"infile", required_argument, NULL, 'i'},
        {"outfile", required_argument, NULL, 'o'},
        {"pcr", required_argument, NULL, 'p'},
        {"unicode", no_argument, NULL, 'u'},
        {"well-known", no_argument, NULL, 'z'}
        };
        unsigned char line[EVP_CIPHER_block_size(EVP_aes_256_cbc()) * 16];
```

```
    int lineLen;
    unsigned char encData[sizeof(line) + EVP_CIPHER_block_size(EVP_aes_256_cbc())];
    int encDataLen;
    UINT32 encLen, i;
    BYTE *encKey;                    //无符号字符
    BYTE *randKey = NULL;
    UINT32 sealKeyLen;
    BYTE *sealKey;
    TSS_FLAG keyFlags = TSS_KEY_TYPE_STORAGE | TSS_KEY_SIZE_2048 |
//对象属性：用以包装其他秘钥的秘钥；2048 位；秘钥是易变的，必须在启动时卸载；
                        TSS_KEY_VOLATILE | TSS_KEY_AUTHORIZATION |
//秘钥需要授权；秘钥不可迁移
            TSS_KEY_NOT_MIGRATABLE;
    TSS_HPOLICY hSrkPolicy;
    char *passwd = NULL;
    int pswd_len;
    BYTE wellKnown[TCPA_SHA1_160_HASH_LEN] = TSS_WELL_KNOWN_SECRET;

    BIO *bin = NULL, *bdata=NULL, *b64=NULL;

    initIntlSys();

    if (genericOptHandler(argc, argv, "i:o:p:uz", opts,          //通用选项处理函数
                          sizeof(opts) / sizeof(struct option), parse,
                          help) != 0)
        goto out;

    if (contextCreate(&hContext) != TSS_SUCCESS)
        goto out;

    if (contextConnect(hContext) != TSS_SUCCESS)
        goto out_close;

    if (contextGetTpm(hContext, &hTpm) != TSS_SUCCESS)
        goto out_close;

    /* Create a BIO for the input file */
    if ((bin = BIO_new(BIO_s_file())) == NULL) {
        logError(_("Unable to open input BIO\n"));
        goto out_close;
```

```
    }

    /* Assign the input file to the BIO */
    if (strlen(in_filename) == 0)
        BIO_set_fp(bin, stdin, BIO_NOCLOSE);
    else if (!BIO_read_filename(bin, in_filename)) {
        logError(_("Unable to open input file: %s\n"),
              in_filename);
        goto out_close;
    }

    /* Create the PCRs object. If any PCRs above 15 are selected, this will need to be
      * a 1.2 TSS/TPM */
    if (selectedPcrsLen) {
        TSS_FLAG initFlag = 0;
        UINT32 pcrSize;
        BYTE *pcrValue;

        for (i = 0; i < selectedPcrsLen; i++) {
            if (selectedPcrs[i] > 15) {
#ifdef TSS_LIB_IS_12
                initFlag |= TSS_PCRS_STRUCT_INFO_LONG;
#else
                logError(_("This version of %s was compiled for a v1.1 TSS, which
"
                      "can only seal\n data to PCRs 0-15. PCR %u is out of range"
                      "\n"), argv[0], selectedPcrs[i]);
                goto out_close;
#endif
            }
        }

        if (contextCreateObject(hContext, TSS_OBJECT_TYPE_PCRS, initFlag,
                    &hPcrs) != TSS_SUCCESS)
            goto out_close;

        for (i = 0; i < selectedPcrsLen; i++) {
            if (tpmPcrRead(hTpm, selectedPcrs[i], &pcrSize, &pcrValue) !=
TSS_SUCCESS)
                goto out_close;

            if (pcrcompositeSetPcrValue(hPcrs, selectedPcrs[i], pcrSize, pcrValue)
                    != TSS_SUCCESS)
                goto out_close;
```

```
        }
#ifdef TSS_LIB_IS_12
        if (initFlag) {
            UINT32 localityValue =
                TPM_LOC_ZERO | TPM_LOC_ONE | TPM_LOC_TWO |
TPM_LOC_THREE |
                TPM_LOC_FOUR;

            if (pcrcompositeSetPcrLocality(hPcrs, localityValue) !=
TSS_SUCCESS)
                goto out_close;
        }
#endif
    }

    /* Retrieve random data to be used as the symmetric key
       (this key will encrypt the input file contents) */
    if (tpmGetRandom(hTpm, EVP_CIPHER_key_length(EVP_aes_256_cbc()),
            &randKey) != TSS_SUCCESS)
        goto out_close;

    /* Load the SRK and set the SRK policy (no password) */
    if (keyLoadKeyByUUID(hContext, TSS_PS_TYPE_SYSTEM, SRK_UUID,
&hSrk)
         != TSS_SUCCESS)
        goto out_close;

    /* Use the context's default policy for the SRK secret */
    if (policyGet(hSrk, &hSrkPolicy) != TSS_SUCCESS)
        goto out_close;

    /* Prompt for SRK password */
    if (!isWellKnown) {
        passwd = _GETPASSWD(_("Enter SRK password: "), (int *)&pswd_len,
FALSE,
                            passUnicode);
        if (!passwd) {
            logError(_("Failed to get SRK password\n"));
            goto out_close;
        }
    } else {
        passwd = (char *)wellKnown;
        pswd_len = sizeof(wellKnown);
    }
```

```
    if (policySetSecret(hSrkPolicy, (UINT32)pswd_len, (BYTE *)passwd) !=
TSS_SUCCESS)
        goto out_close;

    if (!isWellKnown)
        shredPasswd(passwd);
    passwd = NULL;

    /* Build an RSA key object that will be created by the TPM
       (this will encrypt and protect the symmetric key) */
    if (contextCreateObject
        (hContext, TSS_OBJECT_TYPE_RSAKEY, keyFlags,
         &hKey) != TSS_SUCCESS)
        goto out_close;

    if (contextCreateObject
        (hContext, TSS_OBJECT_TYPE_POLICY, TSS_POLICY_USAGE,
         &hPolicy) != TSS_SUCCESS)
        goto out_close;

    if (policySetSecret(hPolicy, strlen(TPMSEAL_SECRET), (BYTE
*)TPMSEAL_SECRET)
        != TSS_SUCCESS)
        goto out_close;

    if (policyAssign(hPolicy, hKey) != TSS_SUCCESS)
        goto out_close;

    /* Create the RSA key (under the SRK) */
    if (keyCreateKey(hKey, hSrk, NULL_HPCRS) != TSS_SUCCESS)
        goto out_close;

    /* Load the newly created RSA key */
    if (keyLoadKey(hKey, hSrk) != TSS_SUCCESS)
        goto out_close;

    /* Build an encrypted data object that will hold the encrypted
       version of the symmetric key */
    if (contextCreateObject
        (hContext, TSS_OBJECT_TYPE_ENCDATA, TSS_ENCDATA_SEAL,
         &hEncdata) != TSS_SUCCESS)
        goto out_close;
```

```
    if (contextCreateObject
          (hContext, TSS_OBJECT_TYPE_POLICY, TSS_POLICY_USAGE,
           &hPolicy) != TSS_SUCCESS)
          goto out_close;

    if (policySetSecret(hPolicy, strlen(TPMSEAL_SECRET), (BYTE
*)TPMSEAL_SECRET)
          != TSS_SUCCESS)
          goto out_close;

    if (policyAssign(hPolicy, hEncdata) != TSS_SUCCESS)
          goto out_close;

    /* Encrypt and seal the symmetric key */
    if (dataSeal
          (hEncdata, hKey, EVP_CIPHER_key_length(EVP_aes_256_cbc()),
           randKey, hPcrs) != TSS_SUCCESS)
          goto out_close;

    if (getAttribData(hEncdata, TSS_TSPATTRIB_ENCDATA_BLOB,
                TSS_TSPATTRIB_ENCDATABLOB_BLOB, &encLen,
                &encKey) != TSS_SUCCESS)
          goto out_close;

    if (getAttribData
                                    (hKey, TSS_TSPATTRIB_KEY_BLOB,
TSS_TSPATTRIB_KEYBLOB_BLOB,
          &sealKeyLen, &sealKey) != TSS_SUCCESS)
          goto out_close;

    /* Create a BIO to perform base64 encoding */
    if ((b64 = BIO_new(BIO_f_base64())) == NULL) {
          logError(_("Unable to open base64 BIO\n"));
          goto out_close;
    }

    /* Create a BIO for the output file */
    if ((bdata = BIO_new(BIO_s_file())) == NULL) {
          logError(_("Unable to open output BIO\n"));
          goto out_close;
    }

    /* Assign the output file to the BIO */
    if (strlen(out_filename) == 0)
```

```
        BIO_set_fp(bdata, stdout, BIO_NOCLOSE);
    else if (BIO_write_filename(bdata, out_filename) <= 0) {
        logError(_("Unable to open output file: %s\n"),
                out_filename);
        goto out_close;
    }

    /* Output the sealed data header string */
    BIO_puts(bdata, TPMSEAL_HDR_STRING);

    /* Sealing key used on the TPM */
    BIO_puts(bdata, TPMSEAL_TSS_STRING);
    bdata = BIO_push(b64, bdata);
    BIO_write(bdata, sealKey, sealKeyLen);
    if (BIO_flush(bdata) != 1) {
        logError(_("Unable to flush output\n"));
        goto out_close;
    }
    bdata = BIO_pop(b64);

    /* Sealed EVP Symmetric Key */
    BIO_puts(bdata, TPMSEAL_EVP_STRING);
    BIO_puts(bdata, TPMSEAL_KEYTYPE_SYM);
    BIO_puts(bdata, TPMSEAL_CIPHER_AES256CBC);
    bdata = BIO_push(b64, bdata);
    BIO_write(bdata, encKey, encLen);
    if (BIO_flush(bdata) != 1) {
        logError(_("Unable to flush output\n"));
        goto out_close;
    }
    bdata = BIO_pop(b64);

    /* Encrypted Data */
    BIO_puts(bdata, TPMSEAL_ENC_STRING);
    bdata = BIO_push(b64, bdata);

    EVP_CIPHER_CTX ctx;
    EVP_EncryptInit(&ctx, EVP_aes_256_cbc(), randKey, (unsigned char
*)TPMSEAL_IV);

    while ((lineLen = BIO_read(bin, line, sizeof(line))) > 0) {
        EVP_EncryptUpdate(&ctx, encData, &encDataLen,
                line, lineLen);
        BIO_write(bdata, encData, encDataLen);
```

```
    }

    EVP_EncryptFinal(&ctx, encData, &encDataLen);
    BIO_write(bdata, encData, encDataLen);
    if (BIO_flush(bdata) != 1) {
        logError(_("Unable to flush output\n"));
        goto out_close;
    }
    bdata = BIO_pop(b64);

    BIO_puts( bdata, TPMSEAL_FTR_STRING);

    iRc = 0;
    logSuccess(argv[0]);

out_close:
    contextClose(hContext);

out:
    if (bin)
        BIO_free(bin);
    if (bdata)
        BIO_free(bdata);
    if (b64)
        BIO_free(b64);
    return iRc;
}
```