An Introduction to programming the TPM

TSS / Trousers basics

David Challener
Johns Hopkins University Applied Physics Laboratory

Table of Contents

Getting your machine set up

Assumption: You are using Fedora 12 Linux or Ubuntu Linux with gcc

- Main install (Fedora 12 Linux w/ gcc)
  - yum install trousers
  - yum install tpm-tools
  - yum install trousers-devel
  - yum install gcc

Ubuntu Linux w/ gcc

```
sudo apt-get install trousers
sudo apt-get install tpm-tools
sudo apt-get install libtspi-dev
sudo apt-get install gcc
```

- Turn on the TPM
  - Go to BIOS and make sure the TPM is on
    - (if it is and you don't know owner auth, you may want to clear it and start over).
    - The procedures differ from PC to PC unfortunately

- Start up tcsd (`sudo tcsd start`)
  - Make sure you can run the TPM tools (use `tpm_getpubek`)

- Take ownership using `tpm_takeownership –z`
  - (The –z sets the SRK password to all zeros, the default "well known secret")
  - Use 123 for the owner_auth for this class

Note: If your machine doesn't have the TPM listed in its ACPI table, you can still get the device driver to use it

- In that case you must use:
  - `sudo modprobe tpm_tis force=1 interrupts=0`
  - `sudo tcsd start`

APL

Comment: Sample code

- **The Trousers test suite exercises each command at least once.**
- **As a result, sample code using each command is available**
- **http://sourceforge.net/projects/trousers/files/**
  - Download TSS API Test Suite

Includes

```
//Basic includes look like this:

#include <stdio.h>
#include <string.h>

#include <tss/tss_error.h>
#include <tss/platform.h>
#include <tss/tss_defines.h>
#include <tss/tss_typedef.h>
#include <tss/tss_structs.h>
#include <tss/tspi.h>
#include <trousers/trousers.h>
```

Error Reporting

If a trousers api fails, you need to translate the error code it gives you into English

Fortunately, that is already coded into trousers

include <trousers/trousers.h> in your includes

Use a debugging statement like:

```
#define DEBUG 0

#define  DBG(message,tResult) if(DEBUG) {fprintf("(Line %d, %s) %s returned 0x%08x. %s.\n", __LINE__, __func__, message, tResult,
      trspi_Error_String(tResult));}
```

Example use: DBG("Created my signing key", result);

Preamble (in virtually every program)

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

{

   TSS_HCONTEXT        hContext=0;
  TSS_HTPM             hTPM = 0;
  TSS_RESULT           result;
   TSS_HKEY            hSRK = 0;
   TSS_HPOLICY         hSRKPolicy=0;
   TSS_UUID            SRK_UUID = TSS_UUID_SRK;
   BYTE                wks[20];  // Place to put the well known secret
  memset(wks,0,20);    // Set wks to the well known secret of 20 bytes of all zeros

// Pick the TPM you are talking to  in this case the system TPM  (which you connect to with "NULL")

       result =Tspi_Context_Create(&hContext);                                  DBG(" Create a Context\n",result);
  result=Tspi_Context_Connect(hContext, NULL);                                  DBG(" Connect to TPM\n", result);

// Get the TPM handle

  result=Tspi_Context_GetTpmObject(hContext, &hTPM);                            DBG(" GetTPM Handle\n",result);  //

Get the SRK handle

  result=Tspi_Context_LoadKeyByUUID(hContext, TSS_PS_TYPE_SYSTEM, SRK_UUID, &hSRK);   DBG(" Tspi_Context_Connect\n",result);

//Get the SRK policy

  result=Tspi_GetPolicyObject(hSRK,  TSS_POLICY_USAGE, &hSRKPolicy);            DBG(" Get TPM Policy\n" ,result; );


// Then we set the SRK policy to be the well known secret
  result=Tspi_Policy_SetSecret(hSRKPolicy,TSS_SECRET_MODE_SHA1,20, wks);    // Note: TSS_SECRET_MODE_SHA1 says "Don't hash this. Just use the 20 bytes
  as is.
                                                                                DBG(" Tspi_Policy_Set_Secret\n" ,result);
```

Cleanup (at end of every program)

```
/* Clean up */
     Tspi_Context_Close (h objects you have created);
     Tspi_Context_FreeMemory(hContext, NULL);
      // this frees memory that was automatically allocated for you
     Tspi_Context_Close(hContext);
 return 0;
}
gcc file –o file.exe -ltspi -Wall
```

Comments on Memory handling

- **If a function calls for a BYTE **, chances are good that the TCS is going to allocate memory for you.**
  - The spec should tell you this if you look for it.

- **This means you need to**
- **Define the variable as**

  BYTE **variable;*

- ***USE the variable as &variable;***
  - **This way the TCS can allocate memory to an unassigned pointer**

- **Note: if you do something stupid, like**

  BYTE *variable[256];*

  Pass &variable;
  - **It will do unpredictable things!!**

Example

Prototype:

```
TSS_RESULT Tspi_Hash_Sign
(
    TSS_HHASH hHash,                // in
    TSS_HKEY  hKey,                 // in
    UINT32*   pulSignatureLength,   // out
    BYTE**    prgbSignature         // out
);
```

Code:

```
UINT32      SignatureLength;
BYTE        *rgbSignature   ;
TSS_RESULT  result;
TSS_HHASH   hHash;
TSS_HKEY    hKey;
…

    result=Tspi_Hash_Sign(hHash, hKey, &SignatureLength, &rbgSignature);
```

What to do in the middle

- **Create objects**
  - Play with their attributes (GetAttrib, SetAttrib)
    - Attributes you can play with are listed in the spec in the section that has functions for that object
  - Create a Policy object to associate with the object
  - Associate the Policy object with another object
  - Instantiate object in Silicon (Key_CreateKey, etc.)
  - Register a key
  - Use an object
    - Sign/Seal/Bind/UnBind/verifySignature/quote with keys
    - Read / Write NVRAM
    - Read/Extend/Reset PCRs

Authorization

- **The TPM requires using a key's authorization every time you use it.**
- **If a user had to enter a password every time he used a key – he wouldn't use the key**
- **TSS Solution:**
  - Tell the TCS context the password for an object – once
  - Every time thereafter (in that program), it will remember and use it
  - Otherwise, create objects which don't require authorization

The middle - authorization

To let the TSS know the authorization for a particular object, like a key or the TPM :

- **Define a Policy object handle**
  - TSS_HPOLICY myPolicyHandle;

- **Associate the Policy handle with a Policy Object**
  - Get an existing Policy (e.g. for the TPM)

  result=Tspi_GetPolicyObject(hTPM, TSS_POLICY_USAGE, &hTPMPolicy);
  - Create a policy (and later associate it with an object)

  result=Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_POLICY, TSS_POLICY_USAGE, &hNewPolicy);

- **Fill in the authorization value into the Policy object**

  Tspi_Policy_SetSecret(hNewPolicy, TSS_SECRET_MODE_PLAIN, lengthOfPassword, *newPassword);

- **Associate the Policy Object with the appropriate object (If you didn't Get an existing policy from an object to begin with)**

  Tspi_Policy_AssignToObject(hNewPolicy, hObject);

Example code

```
// Getting the TPM's policy object
TSS_HPOLICY     hTPMPolicy;
       result=Tspi_GetPolicyObject(hTPM,  TSS_POLICY_USAGE, &hTPMPolicy);
                                                    DBG(" Tspi_GetPolicyObject TPM Policy“, result);

/* Then we set the default Owner's Authorization as its secret */
       result=Tspi_Policy_SetSecret(hOldTPMPolicy,TSS_SECRET_MODE_PLAIN,3, “123”);  // Note: 3 = strlen(“123”)
                                                    DBG(" Tspi_Policy_Set_Secret“, result);

/* Create new Policy and put the new Password in it */
       result= Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_POLICY, TSS_POLICY_USAGE, &hNewPolicy);
                                                    DBG(" Tspi_Context_CreateObject Policy Object“, result);

       result= Tspi_Policy_SetSecret(hNewPolicy, TSS_SECRET_MODE_PLAIN, 20, *newPassword);
                                                    DBG(" Tspi_Policy_SetSecret“, result);


/*Change the password to the one in the new Policy
       result=  Tspi_ChangeAuth(hTPM,0x00000000,hNewPolicy);   //(0x00000000 is the parent of the TPM).
                                                     DBG(" ChangeAuth of TPM“, result);
```

Keys

- **Types of keys**
  - Storage (sealing)
    - Locks to password + PCRs and records PCR values at time of creation
  - Binding
    - Locks to password only (but the key can be locked to PCRs)
  - AIK
    - Are restricted in what they can do, currently only created with the Tspi_Key_CollateIdentityKey command
      - *Can only be 2048 RSA non-migratable keys*
  - Signing
    - Can do anything an AIK can do, plus more. Can be migratable or non-migratable. Can be 1024 or 2048 keys
      - *4sig schems possible. One of them , TSS_SS_RSASSAPKCS11V15_INFO only signs structures, so it is not spoofable*
  - Legacy
    - Can both bind and sign. Dangerous, but used for backwards compatibility
  - (In the future, defined by characteristics)

Create Key

Create Object (by Type)

Fill in what you know / want the key to look like

- Authorization + PCR locking
- Size of key
- Migratable / Non-migratable
- Handle of parent key

Load parent (if not SRK)

Ask TPM to fill in the blanks

- Tspi_Key_CreateKey (unless Identity Key)
- Tspi_TPM_CollateIdentityRequest (if it IS an identity key)

LOAD the key

- Register Key (by UUID)

- Extract the encrypted key blob and store it in a file
- Extract the public key and store it in a file

APL

Code example: Create Binding Key

```
#define BACKUP_KEY_UUID  {0, 0, 0, 0, 0, {0, 0, 0, 0, 0, 2, 10}}
TSS_HKEY                    hBackup_Key;
 TSS_UUID                   MY_UUID = BACKUP_KEY_UUID;
TSS_HPOLICY      hBackup_Policy;
TSS_FLAG         initFlags;
BYTE             *pubKey;
UINT32           pubKeySize;
FILE             *fout;
/* Create a policy for the new key.  I will set it's password to "123" */
        result=Tspi_Context_CreateObject(hContext,   TSS_OBJECT_TYPE_POLICY,   0,   &hBackup_Policy);
           DBG(" Tspi_Context_CreateObject  Policy\n",result);
        Tspi_Policy_SetSecret(hBackup_Policy, TSS_SECRET_MODE_PLAIN, 3, "123");    // SECRET_MODE_PLAIN means it needs to be hashed before use
           DBG(" Set Secret",result);
/* Instantiate a key object that is a 2048 bit RSA key of type "BIND", that requires authorization. */
        initFlags = TSS_KEY_TYPE_BIND | TSS_KEY_SIZE_2048 |  TSS_KEY_AUTHORIZATION | TSS_KEY_NOT_MIGRATABLE;  // Section 2.3.2.2 has choices
        result=Tspi_Context_CreateObject( hContext, TSS_OBJECT_TYPE_RSAKEY, initFlags, &hBackup_Key );   DBG(" Tspi_Context_CreateObject Key\n",result);
/*Assign the policy*/
        result=Tspi_Policy_AssignToObject( hBackup_Policy,hBackup_Key);   DBG(" Tspi_Policy_AssignToObject\n",result); // Can't assign the policy until you have the handle
/*Create and register it */
        result=Tspi_Key_CreateKey(hBackup_Key, hSRK, NULL);         DBG(" Tspi_Key_CreateKey\n",result);              //  Ask the TPM to fill in the blanks
        result=Tspi_Key_LoadKey(hBackup_Key, hSRK);
        result=Tspi_Context_RegisterKey(hContext,   hBackup_Key,  TSS_PS_TYPE_SYSTEM,  MY_UUID,  TSS_PS_TYPE_SYSTEM,   SRK_UUID);
           if(result!=TSS_SUCCESS)  { DBG(" Tspi_Context_RegisterKey\n",result);   return 1; }

/* Now that the key is registered, I also want to store the public portion of the key in a file for distribution*/
// This is done in two parts:       1) Get the public key and 2) stuff it into Backup.pub
        result=Tspi_Key_GetPubKey(hBackup_Key,&pubKeySize, &pubKey);
           if(result!=TSS_SUCCESS)  { DBG(" Tspi_Key_GetPubKey\n,result");   return 1; }
printf("error=%s",(char *)Trspi_Error_String(result));
//          2) Save it in a file.  The file name will be "Backup.pub"
        fout=fopen( "Backup.pub", "w");
           write(fileno(fout), pubKey,pubKeySize);
        fclose(fout);
```

Create a Signing Key, register it and get its public portion

```
#define TSS_UUID_SIGN {0, 0, 0, 0, 0, {0, 0, 0, 0, 2, 0}}// user Sign key 1
UINT32  pubKeyLength;
BYTE    *pubKey;
```

```
// We are going to create a Signing
// Here I determine the key will be a Signing key of 2048 bits, non-migratable, with no authorization.


        initFlags = TSS_KEY_TYPE_SIGNING | TSS_KEY_SIZE_2048 |  TSS_KEY_NO_AUTHORIZATION | TSS_KEY_NOT_MIGRATABLE;

// Create the key object
        result=Tspi_Context_CreateObject( hContext, TSS_OBJECT_TYPE_RSAKEY, initFlags, &hSigning_Key );
           DBG("Tspi_Context_CreateObject SigningKey",result);
```

```
// Now I finally create the key, with the SRK as its parent.
        printf("Creating key... this may take some time\n");
        result=Tspi_Key_CreateKey(hSigning_Key, hSRKey, 0);
        DBG("Create Key", result);

// Once created, I register the key blob so I can retrieve it later
        result=Tspi_Context_RegisterKey(hContext,hSigning_Key, TSS_PS_TYPE_SYSTEM, SIGNING_UUID, TSS_PS_TYPE_SYSTEM,  SRK_UUID);
        DBG("Register key",result);
```

```
/* Now that the key is registered, I also want to store the public portion of the key in a file for distribution*/
/* This is done in two parts: 1) Load the key and read out the public key and stuff it into pubKey*/

        result=Tspi_Key_LoadKey(hSigning_Key,hSRKey);
        DBG("LoadKey",result);
        result = Tspi_Key_GetPubKey(hSignking_Key, &pubKeyLength, &pubKey);
```

Create AIK

- **Requires Owner_auth**
  - Get TPM policy
  - Set Owner_auth secret
- **Get SRK handle (from preamble)**
- **CreateObject (Key of type AIK)**
- **Fill in what you know (key size, etc.)**
- **Tspi_CollateIdentityRequest**
  - implicitly uses TPM auth
  - Requires a CA pub key, EK pub key, etc. usually faked
- **Register it so you can find it later by UUID**

Sample Code

- **Go to:**

http://www.privacyca.com/code.html

For sample code of creating an AIK

Load Key by UUID

- **Get Key by UUID**
- **Load Key**

- **LoadKeyByUUID doesn’t work in TrouSerS, unless parent key is No_Auth. Note the “well known secret” used by the SRK is NOT a no_auth key.**
  - However that is the way you get the SRK handle

// Get the SRK handle

result=Tspi_Context_LoadKeyByUUID(hContext, TSS_PS_TYPE_SYSTEM, SRK_UUID, &hSRK);

if (result!=TSS_SUCCESS) { DBG(" Tspi_Context_Connect\n”,result); return 1; }

Sample Code

```
TSS_HKEY        hBind_Key=0;

Tspi_Context_GetKeyByUUID(hContext, TSS_PS_TYPE_SYSTEM, BACKUP_KEY_UUID, &hBind_Key);
Tspi_Key_LoadKey(hBind_Key, hSRK);

//Cleanup
Tspi_Context_CloseObject(hContext, hBind_Key);
```

Get a public key, given its handle

- Use GetAttributes (Section 4.3.4.18.4)
  - or-
- Use Tspi_Key_GetPubKey

- Save Public key to file

Sample Code

```
UINT32          pubKeySize;
BYTE            *pubKey;  //(Don't use pubKey[284];)
FILE        *fout;
// Get the Public key (can use this or GetPubKey)
      result=Tspi_GetAttribData(hSigning_Key,
                        TSS_TSPATTRIB_KEY_BLOB,
      TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY,
                        &pubKeySize,
                        &pubKey);
        DBG("Get Public key from key object", result);

// 2) Save the public key in a file.  The file name will be "Signing.pub"
      fout=fopen( "Signing.pub", "w");
          write(fileno(fout),pubKey,284);  // or write(fileno(fout),pubKey,pubKeySize);
      fclose(fout);
```

Binding data – the data object

Load a binding key (only the public key is necessary)

- Create a data object
- Fill in the clear text and “bind” (encrypt) data
- Read out encrypted data

Sample Code

```
UINT32    ulDataLength;
BYTE      *rbgBoundData;
// Retrieve the public key
        fin =fopen("Bind.pub", "r");
           read(fileno(fin),newPubKey,284);
        fclose(fin);
// Create a key object
        result=Tspi_Context_CreateObject( hContext, TSS_OBJECT_TYPE_RSAKEY, initFlags, &hBind_Key );      DBG("Tspi_Context_CreateObject BindKey",result);

// Feed the key object with the public key read from the file
        result=Tspi_SetAttribData(hBind_Key,TSS_TSPATTRIB_KEY_BLOB,TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY, 284, newPubKey);
                                                                                                    DBG("Set Public key into new key object", result);
// Read in the data to be encrypted
        fin=fopen("AES.key","r");
           read(fileno(fin),encData,7);
        fclose(fin);
// Create a data object , fill it with clear text and then bind it.
        result=Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_ENCDATA, TSS_ENCDATA_BIND, &hEncData);  DBG("Create Data object",result);
        result=Tspi_Data_Bind( hEncData, hBind_Key, 7,encData);                                          DBG("Bind data",result);

// Get the encrypted data out of the data object
        result=Tspi_GetAttribData( hEncData, TSS_TSPATTRIB_ENCDATA_BLOB,TSS_TSPATTRIB_ENCDATABLOB_BLOB, &ulDataLength,&rgbBoundData);
                                                                                                    DBG("Get encrypted data", result);
// Write the encrypted data out to a file called Bound.data
        fout=fopen("Bound.data", "w");
           write(fileno(fout),rgbBoundData,ulDataLength);
        fclose(fout);
```

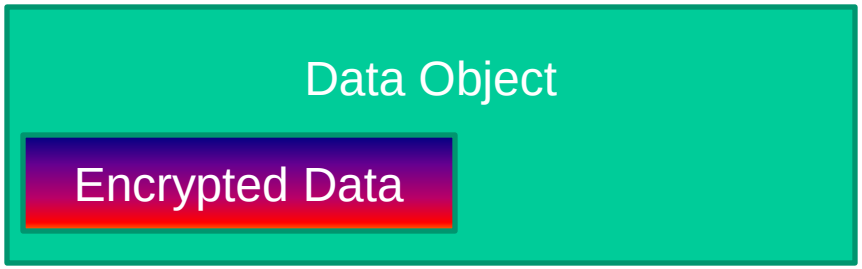
UnBinding data

Load the private key in the TPM

Create a binding data object

Read in encrypted data from file to the data object

Unbind data into variable



Example Code

```
TSS_HENCDATA    hData;
UINT32          encLen=256;
BYTE            encryptedData[256];
BYTE            *rgbDataUnBound;
UINT32          ulDataLength;
```

```
// Read the encrypted data from the file
        fin=fopen("Bound.data", "r");
           read(fileno(fin), encryptedData, ulDataLength);
        fclose(fin);


// Create a new data object
        result=Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_ENCDATA, TSS_ENCDATA_BIND, &hData);          DBG("Create Data object",result);
// Write the encrypted data into the new data object
        result=Tspi_SetAttribData( hData, TSS_TSPATTRIB_ENCDATA_BLOB,TSS_TSPATTRIB_ENCDATABLOB_BLOB, encLen, encryptedData);
                                                                                                                  DBG("Set encrypted data", result);
```

```
// Get the Unbinding private key handle from the standard UUID
        Tspi_Context_GetKeyByUUID(hContext,TSS_PS_TYPE_SYSTEM, BIND_UUID,&hUnBind_Key);                          DBG("Get Key by UUID",result);
// Load the private key into the TPM using its handle
        Tspi_Key_LoadKey(hRecovered_UnBind_Key,hSRKey);                                                  DBG("Load Key", result);
```

```
// Use the private key to decrypt the data into the variable rgbDataUnBound
        result=Tspi_Data_Unbind( hNewEncData, hRecovered_UnBind_Key, &ulDataLength,&rgbDataUnBound);
           DBG("Unbind", result);
```

Sealing data

- **Two ways:**
  - Create a binding key “sealed” to PCRs
  - Create data sealed to PCRs

- **Data sealed to PCRs:**
  - Create PCR object
  - File in PCR values needed for release
  - Create data object for SEAL
  - Write clear text to data object
  - Load storage key
  - Seal data
  - Read out encrypted data

Code example

```
char    TypePass[12]="My Password";


result=Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_PCRS, 0, &hPcrs);          DBG("Create PCR object", result);
result=Tspi_TPM_PcrRead(hTPM, 8, &ulPcrLen, &rgbPcrValue);      DBG("Read the PCR value of PCR 8",result);
result=Tspi_PcrComposite_SetPcrValue(hPcrs, 8, 20, rgbPcrValue );    DBG("Set the current value of PCR 8 for sealing", result);
result=Tspi_TPM_PcrRead(hTPM, 9, &ulPcrLen, &rgbPcrValue);      DBG("Read the PCR value of PCR 9",result);
result=Tspi_PcrComposite_SetPcrValue(hPcrs, 9, 20, rgbPcrValue );    DBG("Set the current value of PCR 9 for sealing", result);
// Create an encrypted data object.
// Data object is used for a seal operation.
result=Tspi_Context_CreateObject(hContext,  TSS_OBJECT_TYPE_ENCDATA,  TSS_ENCDATA_SEAL,  &hEncData );
                                                                    DBG("Create a data object to seal things with", result);
 result = Tspi_Policy_AssignToObject(hEncDataPolicy, hEncData);     DBG("Assign policy to data object", result);
// Seal the password using first data object
result = Tspi_Data_Seal(hEncData,hSRKey,strlen(TypePass),TypePass,hPcrs);          DBG("Sealing with data object", result);
```


APL

Unsealing data

- Create data object (seal type)
- Read in encrypted data
- Write encrypted data into data object
- Load key
- Unseal
- Read out plain text

Code example

```
UINT32      outlength;
BYTE        *outstring;
BYTE        EncryptedData[312];

memset(EncryptedData, 0, 312);
//  Read in the sealed data
      fin=fopen("owner_auth.pass","r");
        read(fileno(fin), EncryptedData,312);
      fclose(fin);


      result=Tspi_SetAttribData(hRetrieveData,
                                TSS_TSPATTRIB_ENCDATA_BLOB,
                                TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                                312,
                                EncryptedData);
            DBG("Set the data object's encrypted data to be that just read in", result);


result=Tspi_Data_Unseal(hRetrieveData, hSRKey, &outlength,&outstring);
            DBG("Unseal the data", result);
```

Signing with a Sign Key

- Load a signing key
- Create Hash object and populate
- Sign Hash object with signing key
- Extract signature and save to file

Sample Code

```
// Get the Signing key handle from the standard UUID
    result=Tspi_Context_GetKeyByUUID(hContext,TSS_PS_TYPE_SYSTEM,SIGNING_UUID,&hSigning_Key);  DBG("Get Key by UUID",result);
// Load the private key into the TPM using its handle
     result=Tspi_Key_LoadKey(hSigning_Key,hSRKey);                                    DBG("Load Key", result);

// Create a Hash Object so as to have something to sign so we  create a generic Hash object //
result=Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_HASH, TSS_HASH_SHA1, &hHashToSign);   DBG("Create Hash object", result);

// Read in a file to hash
        pubKeyLength=filelength("file.dat");
        fin=fopen("file.dat","r");
          read(fileno(fin),pPubKey,pubKeyLength);
        fclose(fin);

// Hash the data using SHA1//

        result=Tspi_Hash_UpdateHashValue(hHashToSign, pubKeyLength, pPubKey);          DBG("Hash in the public key", result);

// Sign the resultant hash object
        result=Tspi_Hash_Sign(hHashToSign,hSigning_Key,&ulSignatureLength,&rgbSignature);          DBG("Sign",result);

// Write the resultant signature to a file called Signature.dat

        fout=fopen( "Signature.dat", "w");
          write(fileno(fout),rgbSignature,ulSignatureLength);
        fclose(fout);
```

Verify Signature

- ***Re-create hash that signature is over***

- ***Load public key into a key object***

- ***Read in signature***
- ***Run VerifySignature***

Sample Code

```
// Create a Hash Object so as to have something to compare the signature to
        // Create a generic Hash object //
        result=Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_HASH, TSS_HASH_SHA1, &hHashToSign);     DBG("Create Hash
        object", result);


        pubKeyLength=filelength("file.dat");
        fin=fopen("file.dat","r");
           read(fileno(fin),pPubKey,pubKeyLength);
        fclose(fin);


        // Hash the data using SHA1//


        result=Tspi_Hash_UpdateHashValue(hHashToSign, pubKeyLength, pPubKey);                DBG("Hash in the public key", result);


 // We are going to create a Verify key
        fin = fopen("Sign.pub", "r");
           read(fileno(fin),pubVerifyKey,284);
        fclose(fin);


        initFlags = TSS_KEY_TYPE_SIGNING | TSS_KEY_SIZE_2048 | TSS_KEY_NO_AUTHORIZATION | TSS_KEY_NOT_MIGRATABLE;
        result=Tspi_Context_CreateObject( hContext, TSS_OBJECT_TYPE_RSAKEY, initFlags, &hVerify_Key );
        DBG("Tspi_Context_CreateObject Verify_Key",result);
        result=Tspi_SetAttribData(hVerify_Key,TSS_TSPATTRIB_KEY_BLOB,TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY,
        pubSignKeyLength,pubVerifyKey);
           DBG("SetPubKey in Verify_Key", result);


// Read in signature and verify it
        fin = fopen("Signature.dat", "r");
           read(fileno(fin), Signature, 256);
        fclose(fin);


        result=Tspi_Hash_VerifySignature(hHashToSign,hVerify_Key,256,Signature);              DBG("Verify", result);
```

NVRAM

- **Create space at specific index, specific size**

- **Requires TPM owner authorization to define or destroy**
  - Get TPM policy
  - Fill in TPM owner_auth in TPM policy

- **Create NVRAM object**
  - Set specific data (size, index, authorizations)
  - DefineSpace at a specified index

Comments: There is index overhead (about 93 bytes per index), so you typically can't make an infinite number of indices. You can however put multiple things in a particular index, using offsets to get to them.

Example Code (only run once!)

```
TSS_HNVSTORE       hNVStore;
/* Create a NVRAM object */
        result = Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_NV, 0, &hNVStore);
        if (result!=TSS_SUCCESS)        { DBG(" Tspi_Context_CreateObject: %x\n",result);            return 1;  }

/*Next its arbitrary index will be 0x00011101 (00-FF are taken, along with 00011600). */
        result = Tspi_SetAttribUint32(hNVStore, TSS_TSPATTRIB_NV_INDEX,0,0x00011101);
        if (result!=TSS_SUCCESS)        { DBG(" Tspi_SetAttribUint32 index %x\n",result);   return 1;  }

/* set its Attributes.  First it is only writeable by the owner */
        result = Tspi_SetAttribUint32(hNVStore,TSS_TSPATTRIB_NV_PERMISSIONS, 0, TPM_NV_PER_OWNERWRITE);
        if (result!=TSS_SUCCESS)        { DBG(" Tspi_SetAttribUint32 auth %x\n",result);   return 1;  }

/* next it holds 40 bytes of data */
        result = Tspi_SetAttribUint32(hNVStore, TSS_TSPATTRIB_NV_DATASIZE,0,40);
         if (result!=TSS_SUCCESS)       { DBG(" Tspi_SetAttribUint32 size%x\n",result);   return 1;  }

/* In order to either instantiate or write to the NVRAM location in NVRAM, owner_auth is required.  In the  case of NVRAM,
       owner_auth comes from the TPM's policy object.  We will put it in here. */
/* First we get a TPM policy object*/
        result = Tspi_GetPolicyObject(hTPM, TSS_POLICY_USAGE, &hTPMPolicy);
        if (result!=TSS_SUCCESS)        { DBG(" Tspi_GetPolicyObject: %x\n",result);   return 1;  }

/* Then we set the Owner's Authorization as its secret */
        result = Tspi_Policy_SetSecret(hTPMPolicy, TSS_SECRET_MODE_PLAIN, 3, “123");
        if (result!=TSS_SUCCESS)        { DBG(" Tspi_Policy_SetSecret: %x\n",result);   return 1;  }

/* Create the NVRAM space */
        result = Tspi_NV_DefineSpace(hNVStore,0,0);
         if (result!=TSS_SUCCESS)       { DBG(" Tspi_NV_DefineSpace: %x\n",result);   return 1;  }
```

NVRAM

- **Write to NVRAM**
  - Create NVRAM object
  - Set Policy secret
  - Set TPM Policy secret in the NVRAM object
  - Write data

  - ***Comment: Although the TPM knows that the NVRAM index's password is that of the TPM owner, TrouSerS has no way of knowing this. As a result, you must tell TrouSerS this by creating a policy secret, filling it with the TPM owner's authorization, and then associating it with the NVRAM object.***

Example Code (Write to NVRAM)

```
TSS_HNVSTORE    hNVStore;
TSS_HPOLICY     hNewPolicy;
char            dataToStore[19]="This is some data."
```

```
/* Create a NVRAM object */
        result = Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_NV, 0, &hNVStore);
        if (result!=TSS_SUCCESS)    { DBG(" Tspi_Context_CreateObject: %x\n",result);        return 1;  }

/*Next its arbitrary index will be 0x00011101 (00-FF are taken, along with 00011600). */
        result = Tspi_SetAttribUint32(hNVStore, TSS_TSPATTRIB_NV_INDEX,0,0x0x00011101);
        if (result!=TSS_SUCCESS)    { DBG(" Tspi_SetAttribUint32 index %x\n",result);   return 1;  }

/* set its Attributes.  First it is only writeable by the owner */
        result = Tspi_SetAttribUint32(hNVStore,TSS_TSPATTRIB_NV_PERMISSIONS, 0, TPM_NV_PER_OWNERWRITE);
        if (result!=TSS_SUCCESS)    { DBG(" Tspi_SetAttribUint32 auth %x\n",result);   return 1;  }

/* next it holds 40 bytes of data */
        result = Tspi_SetAttribUint32(hNVStore, TSS_TSPATTRIB_NV_DATASIZE,0,40);
         if (result!=TSS_SUCCESS)    { DBG(" Tspi_SetAttribUint32 size%x\n",result);   return 1;  }
```

```
/* Set Policy for the NVRAM object using the Owner Auth */
        result = Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_POLICY, TSS_POLICY_USAGE, &hNewPolicy);
        result = Tspi_Policy_SetSecret(hNewPolicy, TSS_SECRET_MODE_PLAIN,3,"123");
        result = Tspi_Policy_AssignToObject(hNewPolicy,hNVStore);
```

```
/* Write to the NVRAM space */
        result = Tspi_NV_WriteValue(hNVStore, 0,18, dataToStore);
        if (result!=TSS_SUCCESS)    { DBG(" Tspi_NV_WriteValue: %x\n",result); return 1;      }
```


APL

NVRAM

- **Read from NVRAM**
  - Create NVRAM object
  - Set Policy secret if needed
  - Read data
- **Faster than unseal, as it does not need a private key operation**
- **High overhead (around 93 bytes) limit number of NVRAM indices that can be used**
  - If the same authorization is used, the same index can be reused.

Example Code (Read fromNVRAM)

```
TSS_HNVSTORE      hNVStore;
char              dataToStore[19]={0};
/* Create a NVRAM object */
        result = Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_NV, 0, &hNVStore);
        if (result!=TSS_SUCCESS)     { DBG(" Tspi_Context_CreateObject: %x\n",result);  return 1;  }

/*Next its arbitrary index will be 0x00011101 (00-FF are taken, along with 00011600). */
        result = Tspi_SetAttribUint32(hNVStore, TSS_TSPATTRIB_NV_INDEX,0,0x0x00011101);
        if (result!=TSS_SUCCESS)     { DBG(" Tspi_SetAttribUint32 index %x\n",result);   return 1;  }

/* set its Attributes.  First it is only writeable by the owner */
        result = Tspi_SetAttribUint32(hNVStore,TSS_TSPATTRIB_NV_PERMISSIONS, 0, TPM_NV_PER_OWNERWRITE);
        if (result!=TSS_SUCCESS)     { DBG(" Tspi_SetAttribUint32 auth %x\n",result);   return 1;  }

/* next it holds 40 bytes of data */
        result = Tspi_SetAttribUint32(hNVStore, TSS_TSPATTRIB_NV_DATASIZE,0,40);
         if (result!=TSS_SUCCESS)    { DBG(" Tspi_SetAttribUint32 size%x\n",result);   return 1;  }

/* No authorization needed to read from this NVRAM the way it was created. /
/* Read from the NVRAM space */
         result = Tspi_NV_ReadValue(hNVStore,0, 18, &datatoStore[0]);
        if (result!=TSS_SUCCESS)     { DBG(" Tspi_NV_ReadValue: %x\n",result); return 1;    }
```


APL

PCR objects

- **Manipulate PCRs**
  - Read
  - Change (Extend)
  - Reset (only PCR 16 and 23)
- **Assign PCRs for authorization**
  - Keys
  - Data (sealing)
- **Quote PCRs (Attestation)**
- **Check attestation**

Create PCR object, read PCRs

```
/*Create a PcrComposite that has the current PCR values 17 and 18 in it. */
/* Create the PCR composite object.  I use TSS_PCR_INFO_SHORT, because my PCR > 15 */

BYTE          *digestValue17, *digestValue18;

result=Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_PCRS, TSS_PCR_STRUCT_INFO_SHORT, &hPcrs);
                                                                    DBG("CreateObject PCRs",result);

/* Read PCR indices 17, and 18 and set their values in the object */

result=Tspi_TPM_PcrRead(hTPM,17,&PCRlength,     &digestValue17);    DBG("PcrRead 17", result);
result = Tspi_PcrComposite_SetPcrValue( hPcrs, 17, PCRlength, digestValue17 );    DBG("SetPcrValue 17", result);
result=Tspi_TPM_PcrRead(hTPM,18,&PCRlength,     &digestValue18);    DBG("PcrRead 18", result);
result = Tspi_PcrComposite_SetPcrValue( hPcrs, 18, PCRlength, digestValue18 );    DBG("SetPcrValue 18", result);
```

Extend a PCR value

```
BYTE        myinput="Hello world"
BYTE        *Final_PCR_Value;
//Extend the value

result=Tspi_TPM_PcrExtend(hTPM,16,sizeof(myinput),(BYTE *)myinput, NULL, PCR_result_length, &Final_PCR_Value);

printf("Afterwards, PCR number 16 has current value %s\n",  Final_PCR_Value);
```

Attestation

- Decide what PCRs you want attested to
- Decide what AIK key you want to use
- Load the AIK
- Create a PCR object
- Put the correct PCR indexes in object
- Set the random number into the validation structure
- Quote

Sample Code for both quote and verify Quote

- **Go to:**

http://www.privacyca.com/code.html

For sample code both quoting and verifying a quote

Reading the log file (Note: use latest Trousers)

```
UINT32                  ulpcrIndex = 9;
UINT32                  ulStartNumber=0;
UINT32                  ulEventNumber=15;
TSS_PCR_EVENT           *prgbPcrEvents;
char                    eventBlank[256];
int                     i;

Tspi_TPM_GetEvents(hTPM,pcrIndex,
                        ulStartNumber,
                        (UINT32 *)&pcrNumber,
                        &prgbPcrEvents);

for(i=0; i\textless pcrnumber;++i)
{
    memset(eventBlank,0,256);
    memcpy(eventBlank,
              prgbPcrEvents[i].rgbEvent,
              prgbPcrEvent[i].ulEventlog
          );
    printf("Event %d, is %s \n “,I,eventBlank);
}
```

RNG

- Get the TPM handle (from the preamble)
- Ask it for some random bytes
- Store and print the random bytes

Sample Code

```
char            *randomBytes;
FILE            *fout;
TSS_RESULT      result;

int numRandomBytesOut =atoi(argv[1]);
if (( randomBytes = (char *) malloc( numRandomBytesOut ) ) == NULL)

/* Ask the TPM for a 20 byte Random Number, and stuff it in the randomBytes variable */
      Tspi_TPM_GetRandom(hTPM,numRandomBytesOut,random);
// Print it out for the user to see
/*    for (i=0;i<numRandom;++i)
      {
        fprint("%c02h”,random[i]);
      }
      fprint("\n");
```

Hashing data

- First you create a Hash object
- Then you getthe data to be hashed
- Then you use HashExtend
- Then you read out the hashed data

Example code: Hashing a string

```
TSS_HHASH       hHashOfKey;
BYTE            initialHash[20];
 BYTE           dataToHash[82]=“Four score and seven years ago, our forefathers brought forth upon this continent”
UINT32          digestLen;
BYTE            *digest;

memset(initialHash,0,20);


// Create a generic Hash object //
result=Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_HASH, TSS_HASH_SHA1, &hHashOfKey);
                                                                    DBG("Create Hash object", result);


// Hash the data using SHA1//
result=Tspi_Hash_UpdateHashValue(hHashOfKey, 82, dataToHash);       DBG("Hash in the data", result);
result=Tspi_Hash_GetHashValue(hHashOfKey,&digestLen, &digest);      DBG("Get the hashed result", result);
```

Get a public key, given its handle

- Use GetAttributes (Section 4.3.4.18.4)
  - or-
- Use Tspi_Key_GetPubKey

- Save Public key to file

Sample Code

```
UINT32        pubKeySize;
BYTE          *pubKey;//(Don't use pubKey[284];)
FILE          *fout;
// Get the Public key (can use this or GetPubKey)
    result=Tspi_GetAttribData(hSigning_Key,
                              TSS_TSPATTRIB_KEY_BLOB,
                              TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY,
                              &pubKeySize,
                              &pubKey);
     DBG("Get Public key from key object", result);
```

```
// 2) Save the public key in a file.  The file name will be "Signing.pub"
    fout=fopen( "Signing.pub", "w");
        write(fileno(fout),pubKey,284);  // or write(fileno(fout),pubKey,pubKeySize);
    fclose(fout);
```

OwnerEvictKey

- Load Key
- Set Owner auth (get TPM policy, set secret)
- Set key as owner_evict key

Sample Code

```
// Make the key an Owner_Evict key

// Set TPM Owner auth, so that you have permission to make the key “owner evict”

result=Tspi_GetPolicyObject(hTPM, TSS_POLICY_USAGE, &hTPM_Policy);          DBG("Getting TPM Policy object", result);

result = Tspi_Policy_SetSecret(hTPM_Policy, TSS_SECRET_MODE_PLAIN, 3, “123");   DBG("Set TPM policy object's secret to 123", result);

// Load the key into the TPM
result=Tspi_Key_LoadKey(hMB_AIK_Key, hSRKey);          DBG("Load the key into the TPM", result);

// Tell the TPM to not allow anyone but the owner to evict it
result=Tspi_TPM_KeyControlOwner(hTPM, hMB_AIK_Key, TSS_TSPATTRIB_KEYCONTROL_OWNEREVICT, TRUE,
&pUuidData);
                                                    DBG("Make key an owner evict key", result);

// In order to not fill up the TPM with repeated tests, change it back
result=Tspi_TPM_KeyControlOwner(hTPM,hMB_AIK_Key, TSS_TSPATTRIB_KEYCONTROL_OWNEREVICT, FALSE,
&pUuidData);
                                                    DBG("Unmake the key an owner evict key", result);
```

Migration: making a ticket

```
// Set TPM Owner auth, so that you have permission to make the ticket
        result=Tspi_GetPolicyObject(hTPM, TSS_POLICY_USAGE, &hTPM_Policy);
                DBG("Getting TPM Policy object", result);
        result = Tspi_Policy_SetSecret(hTPM_Policy, TSS_SECRET_MODE_PLAIN, 3, “123");
                DBG("Set TPM policy object's secret to 123", result);
// Read in the public key you want to bless
 fin=fopen( "Storage.pub", "rb");
        read(fileno(fout),pubKey,284);
 fclose(fin);
// Create the key object
 result=Tspi_Context_CreateObject( hContext, TSS_OBJECT_TYPE_RSAKEY,   initFlags,  &hStorage_Key );
          DBG("Tspi_Context_CreateObject StorageKey",result);
// Put the public key into the key object
result=Tspi_SetAttribData(hStorage_Key,TSS_TSPATTRIB_KEY_BLOB, TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY,
                                &pubKeySize,  &pubKey);
        DBG("Set Public key into key object", result);
// Create ticket
result=Tspi_TPM_AuthorizeMigrationTicket(hTPM,hStorage_Key, TSS_MS_REWRAP, &TicketLength, &rgbMigTicket);
        DBG("Make Ticket",result);
//Save ticket
 fout=fopen( "Ticket", "w");
    write(fileno(fout),rgbMigTicket,TicketLength);
 fclose(fout);
```

Migrating a key

```
initFlags = TSS_KEY_TYPE_SIGNING | TSS_KEY_SIZE_2048 |  TSS_KEY_NO_AUTHORIZATION | TSS_KEY_MIGRATABLE;

// Create the key object
result=Tspi_Context_CreateObject( hContext, TSS_OBJECT_TYPE_RSAKEY, initFlags, &hMigrateStorageKey );
            DBG("Tspi_Context_CreateObject SigningKey",result);


// I have to assign a migration policy to the key I am creating – hMigrateStorage\Key
// Create migration policy
result = Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_POLICY,
                                                TSS_POLICY_MIGRATION, &hMigPolicy);

//Set MIGRATION Secret
result = Tspi_Policy_SetSecret(hMigPolicy,   TSS_SECRET_MODE_PLAIN,   7,(BYTE *)"Migrate");

//Assign migration policy
result = Tspi_Policy_AssignToObject(hMigPolicy, hMigrateStorage_Key);

// Now I finally create the key, with the SRK as its parent.
result=Tspi_Key_CreateKey(hMigrateStorage_Key, hSRKey, 0);
       DBG("Create Key", result);

fin=fopen("ticket.dat","rb");
      read(fileno(fin),ticket,ticketLength);
fclose(fin);

result=Tspi_Key_CreateMigrationBlob(hMigrateStorage_Key, hSRKey, ticketLength, ticket,
                                                                &rnLength, &rn, &migBlobLength, &migBlob);

//(Note rn and rnLength are not used here, as they are for double encryption, not specified when the ticket was created)
       DBG("Create ReWrapped Key",result);

       fout=fopen( "Migrated.blob", "wb");
           write(fileno(fout), migrated_blob, blob_length);
        fclose(fout);
```

Loading a migrated key

```
TSS_HKEY  hMigratedKey;
FILE      *fin;
char      migrated_blob[1024];
int       blob_length=MIGRATED_KEY_BLOB_SIZE;

memset(migrated_blob,0,1024);

fin=fopen( "Migrated.blob", “rb");
      read(fileno(fin), migrated_blob, blob_length);
fclose(fin);

result=Tspi_Context_LoadKeyByBlob(hContext,hSRKey, blob_length,
                                  rgbBlobData, hMigratedKey) ;
     DBG(“Load the migrated blob",result);
```

More on authorization

- **TPM owner authorization is required for…**
  - Changing attributes of the TPM
  - Creating/destroying an NVRAM space
  - Creating an AIK key
  - Creating an AIK certificate with “Activate Identity”
  - Delegation of owner auth
  - Clearing the TPM (without physical presence)
  - Making a key an owner evict key
  - Etc…..
- **Sometime doing something with an object requires that owner auth be give to the context earlier!**

Problem Scenario

- Maintain the integrity of a public key
- Scenario: Suppose you want to have an enterprise public/private key pair. The private key is used to decrypt things in case of emergency, and it is tightly controlled. The public key is given to employees.
  - It is assumed that employees will have information (perhaps of order data) that is confidential and must be encrypted whenever it is at rest.
  - If the employee should die, that data needs to be available to the enterprise.
  - Software is used to encrypt the data with an AES key, with the key being non-migrateably bound to the platform.
  - The AES key is also encrypted with the enterprise public key, in case the employee dies or his platform / motherboard / TPM dies.

Your mission:

- Create a public / private Binding key, called EnterpriseBackup
- Register the key
- Create a file with the public portion (EnterpriseBackup.pub)
- Create an NVRAM space with 20 bytes, generally readable, but only writeable with the owner authorization at index 0x00011101
- Use the owner authorization to write a hash of the pub key into the NVRAM space
- Write a program that has input of 32 random hex bytes
  - Compares the hash value of EnterpriseBackup.pub with the value stored in the NVRAM location: 0x00011101
  - If the comparison matches, encrypts (binds) the 32 hex bytes with the EnterpriseBackup.pub key and stores the encrypted data in the file Encrypted.dat
- Write another program using the registered key to decrypt the file Encrypted.dat