Introduction to Using the TSS

Ari Singer

NTRU Cryptosystems

November 7, 2005

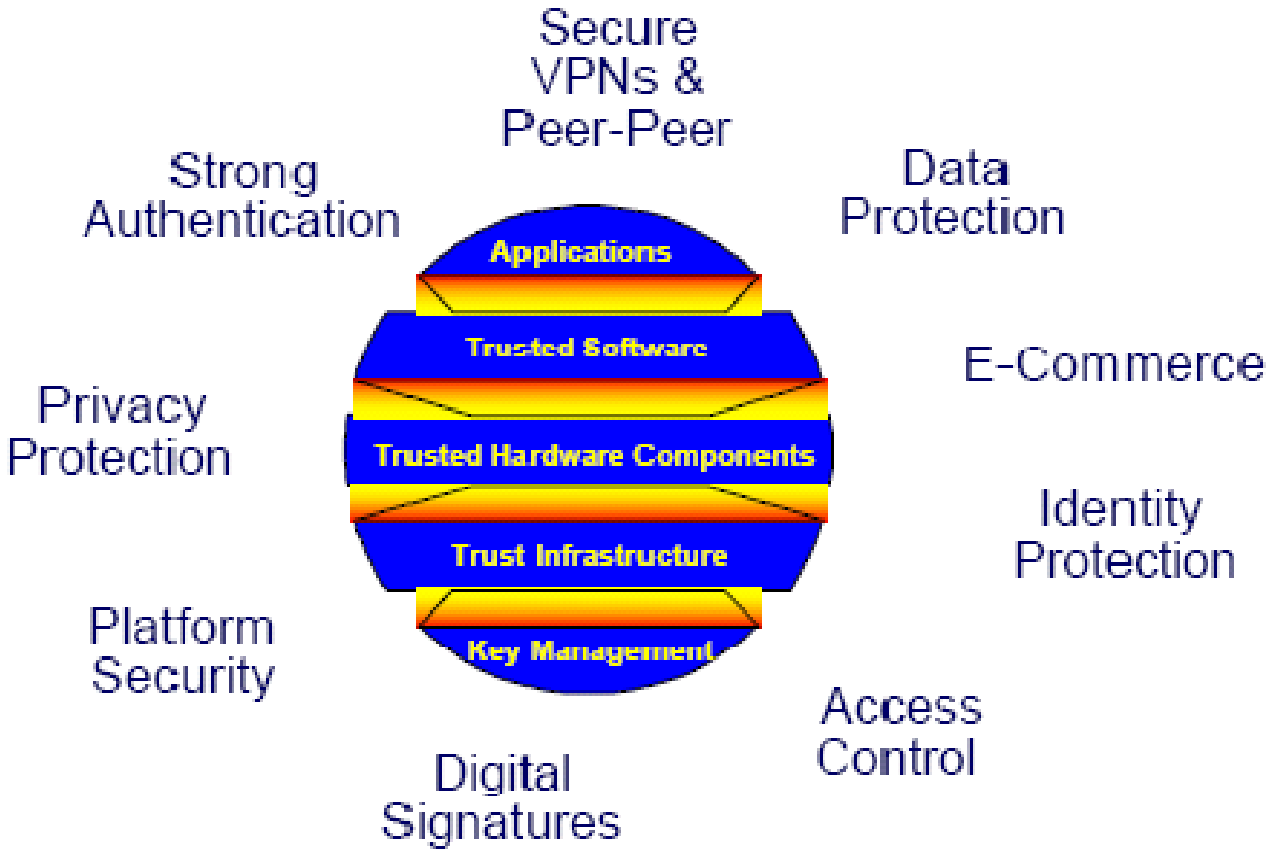
Outline

- Motivating Use Cases
- TPM overview
- Summary of TCG (PC) Architecture
- Accessing the TPM
- TSS overview
- Coding to the TSS
- Mapping to use cases
- Conclusions

TRUSTED COMPUTING GROUP™

Ntrū

Trusted Computing Applications



TRUSTED COMPUTING GROUP™

Ntrū

Some Motivating Use Cases

- How do I…
  - Store a key securely, so a user can access it with a password?
  - Back up a key securely, so IT can help the user out when he forgets his password?
  - Ensure that I am communicating with a particular user with access to a particular machine?
  - Make sure my software only runs on a specific machine?
  - Make sure my software runs only on machines in a specific state?
- We'll first do a lot of background, then come back to these use cases

TRUSTED COMPUTING GROUP™

Ntrū

Building On Existing Building Blocks

- There are many components that go into building a “trusted” system
- Great progress has been made building up an ecosystem of trusted computing products and services
- The task ahead of us is to build on that ecosystem

TRUSTED COMPUTING GROUP™

Ntrū

Understanding the TPM

- The TPM specification is rather complex
- It is important to have some understanding of the TPM when using the TSS
- So, here are some of the basics . . .

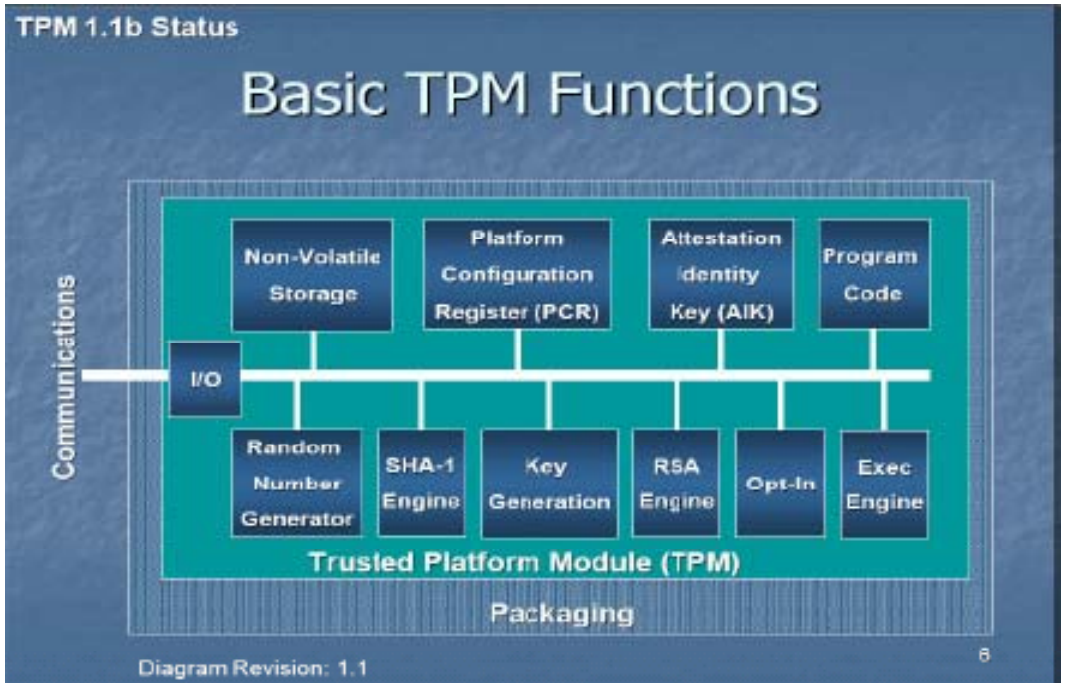
What Makes TPMs Special?

- Hardware-based state measuring (root of trust for measurement)
- Hardware-based attestation (root of trust for reporting)
- Fundamental part of the platform
- Wide variety of cryptographic/security functionality
- Robust management interface

TRUSTED COMPUTING GROUP™

Ntrū

TPM 1.1b Overview



TRUSTED COMPUTING GROUP™

Ntrū

TPM Management

- The “Owner” manages access control to various TPM resources
  - Protected by use of an authorization secret: essentially a password, stored in TPM tamperproof memory
  - Controls access to root keys on the TPM (EK and SRK)
  - Controls attestation capabilities of TPM
  - Controls migration capabilities TPM keys
- “Physical presence” helps in TPM management
  - Used to restrict access to the TPM usually until ownership is taken
  - Used to recover TPM functionality when owner password is lost
- In 1.2, owner can perform “delegation”
  - Associate an alternate password for specified operations
  - The person who knows the extra password can perform approved actions, but no others.
  - Allows enterprises to have IT as owner, user as privileged operator

TPM Keys

- Endorsement key for root of TPM trust
- Storage root key for top of key hierarchy
- Storage keys for key hierarchy and sealing
- Identity keys for certifiable signatures
- Binding keys for binding
- Signing keys for signing arbitrary data
- Legacy keys that can both sign and encrypt.

TRUSTED COMPUTING GROUP™

Ntrū

Platform Configuration Registers (PCR)

- Designed to correspond to hardware/software configuration
- Functionally stores hashes (“measurements”)
- Cannot be overwritten, only “extended”
- Used for access control (“seal” data to a particular configuration)
- Used for attestation (“quote” a particular configuration)
- Some PCRs are resettable and controlled by specific “localities” (ports on the TPM)

TRUSTED COMPUTING GROUP™

Ntrū

Key Migration

- Keys are either migratable or non-migratable
- Owner authorizes where keys can be migrated to
- Migrating a parent also migrates the entire branch of the key hierarchy below it
- In 1.2, you can create certifiable migration keys (CMK), which have finer access control

TPM 1.1b Functions

- Seal/Unseal – Encrypting for local platform only
- Bind/Unbind – Encrypting between platforms
- Quote – Signing with attestation

TRUSTED COMPUTING GROUP™

Ntrū

Additional TPM 1.2 Functions

- Transport sessions – SSL-like functionality
- Direct Anonymous Attestation (DAA) – Attestation without revealing the identity of the TPM
- Tick count – Used to provide time stamping functionality
- Monotonic counter – Used to prevent replay
- Non-volatile (NV) storage – Hardware-protected access controlled NV memory

TRUSTED COMPUTING GROUP™

Ntrū

Trusted PC Building Blocks

- A trusted personal computer (PC) may have many components
  - TCG core roots of trust
    - RTS and RTR inside TPM
    - RTM part of motherboard
  - TCG-enabled BIOS
  - TCG-aware OS
  - TCG Software Stack (TSS)
  - TCG-enabled CSP(s)
  - TCG development tools
  - TCG-enabled applications

TRUSTED COMPUTING GROUP™

Ntrū

Accessing the TPM

- There are several APIs and mechanisms for accessing the TPM
- These APIs require differing levels of understanding of the TPM
- Some mechanisms abstract away more TPM complexity than others

TCG-enabled BIOS

- When the PC first boots, the trust in the code is valid as long as the code that is running has been measured
- The BIOS measures the initial code and performs certain “physical” actions on the TPM

TCG-aware Operating System

- If the BIOS measures the operating system (OS) properly, the OS can take advantage of the TPM state
- Microsoft™ has announced support for some TPM functionality in its new Vista™ OS

TCG Software Stack (TSS)

- The TSS is a software stack that exposes the functionality of the TPM and provides a common interface to access TPM functionality.
- Built on top of the OS
- We'll talk about this in a minute

TCG-enabled CSP

- Cryptographic Service Providers (CSP) provide standard interfaces to applications for the use of keys (e.g. CAPI, PKCS #11)
- TCG-enabled CSPs abstract away the TPM and TSS
- May or may not limit access to TPM functionality

TRUSTED COMPUTING GROUP™

Ntrū

TCG Development Tools

- Software companies may make development tools available to simplify the writing of TCG applications
- This may make it even easier for application writers to write TCG-enabled applications

TCG-enabled Applications

- Applications that ultimately build on the security and trust provided by all of the layers below

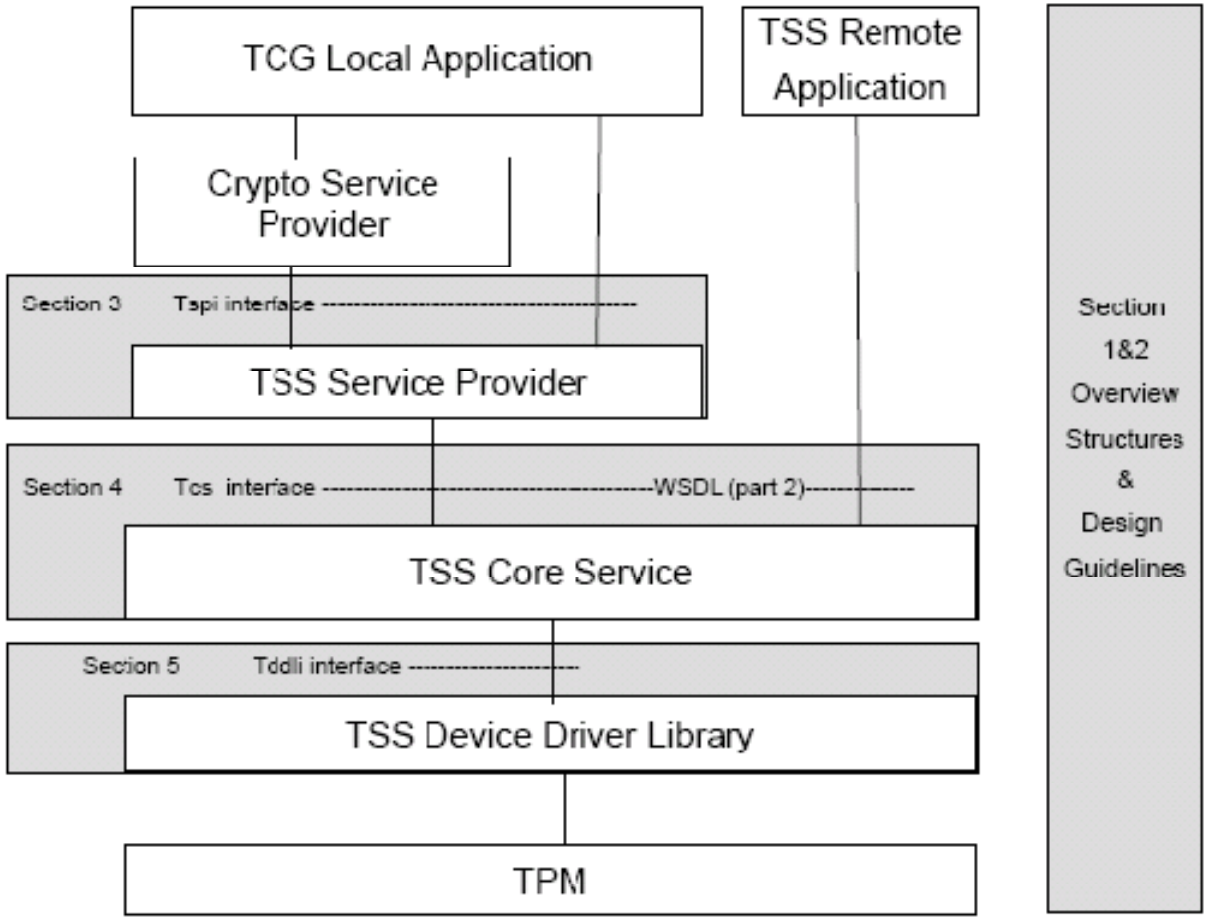
Understanding the TSS

- The TSS abstracts some of the TPM complexities away
- If you learn the TPM basics and the TSS API, you can create secure applications
- The main goals of the TSS are:
  - Supply one entry point for applications to the TPM functionality
  - Provide synchronized access to the TPM
  - Hide building command streams with appropriate byte ordering and alignment from the applications
  - Manage TPM resources
  - Release TPM resources when appropriate
  - Manage application use of secrets and keys

TRUSTED COMPUTING GROUP™

Ntrū

TSS 1.2 Architecture



TCG Device Driver Library (TDDL)

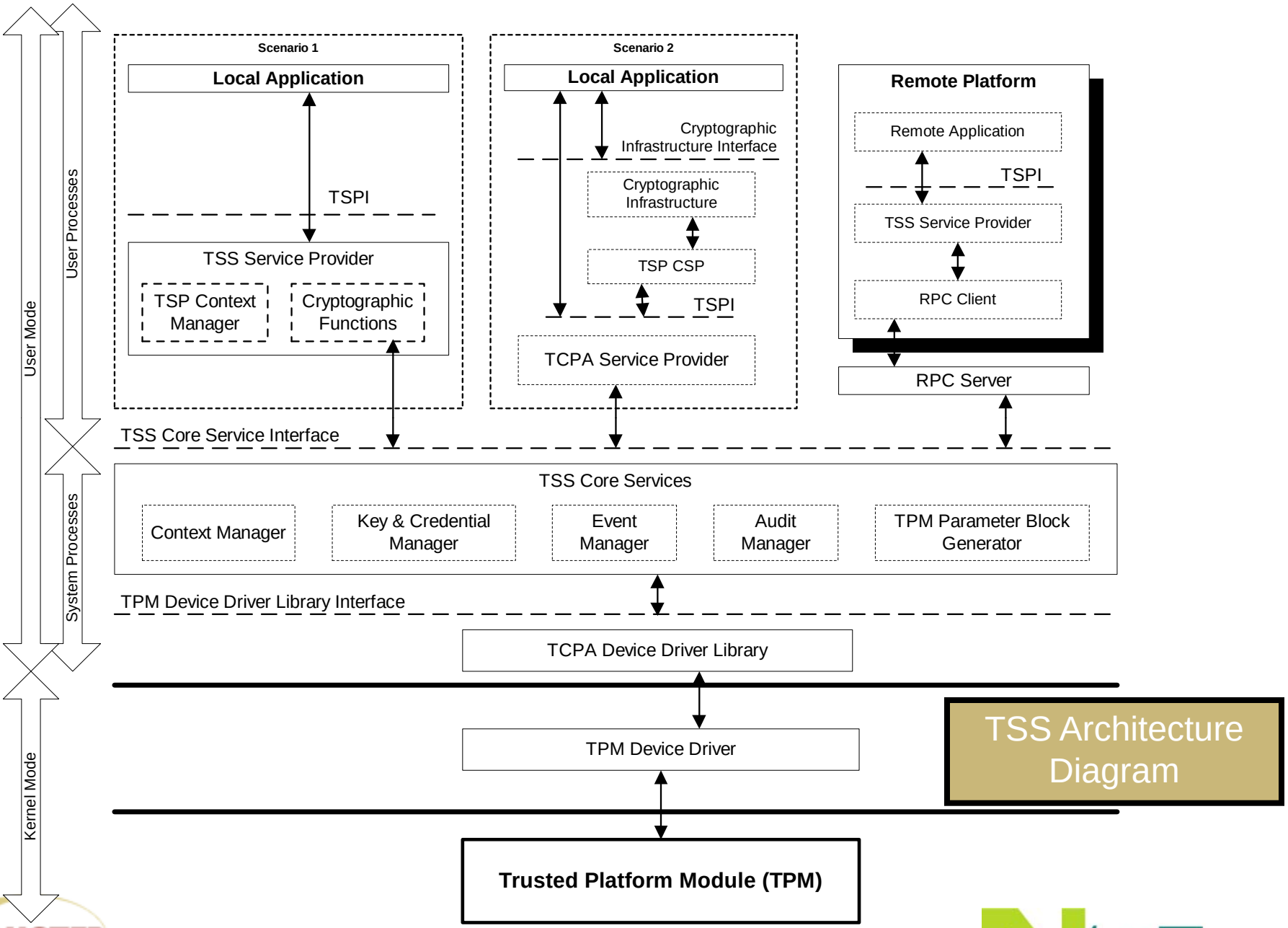
- Creates an abstraction layer hiding OS-specific device driver interfaces from the TCS
- Single point of compatibility for TSS developers
- Allows the TPM vendor to get/set device driver capabilities

TCG Core Services (TCS)

- Parameter Block Generator (PBG)
  - Converts ‘C’ style parameters into TPM format.
- Key and Credential Manager (KCM)
  - Allows the user to alias and persistently store a TPM key.
  - Dynamically swaps keys into and out of the TPM
- Context Manager
  - Allows multiple TSP modules to access TCS simultaneously
  - Performs memory management on a per context basis
- Event Manager
  - Generates, manages and exports “PCR Events”
- Audit Manager
  - Intended to leave records of TPM activity

TCG Service Provider (TSP)

- Exposes TSPI
  - User Friendly API that incorporates object oriented principles
  - Abstracts the underlying protocols and data structures
- TSP Context Manager
  - Allows multiple instances of TSP layer
  - Performs memory management at the TSP Layer
- Public-key cryptography and hashing/HMAC
  - Not all cryptography requires the TPM
  - Performs public-key, hashing and HMAC algorithms to enhance cryptographic security and authorization for the TPM


User Mode
User Processes
System Processes
Kernel Mode
Scenario 1
Local Application
TSPI
TSS Service Provider
TSP Context Manager
Cryptographic Functions
Scenario 2
Local Application
Cryptographic Infrastructure Interface
Cryptographic Infrastructure
TSP CSP
TSPI
TCPA Service Provider
Remote Platform
Remote Application
TSPI
TSS Service Provider
RPC Client
RPC Server
TSS Core Service Interface
TSS Core Services
Context Manager
Key & Credential Manager
Event Manager
Audit Manager
TPM Parameter Block Generator
TPM Device Driver Library Interface
TCPA Device Driver Library
TPM Device Driver
Trusted Platform Module (TPM)
TSS Architecture Diagram

TSS Highlights

- Virtualizes resources used inside the TPM
  - Multiple applications can run simultaneously, each using different keys
  - Applications do not have to manage key load/unload themselves
- Actions such as Seal are authorized using an authorization secret
  - TSS provides means to enter, cache, and expire the secret
- TPM commands are all formed inside the TSS – they are not exposed directly to the applications

Code Samples

- Now, here are some details about how to actually use this stuff . . .

Seal to PCR Code

```
int
main(void)
{
    TSS_HCONTEXT    hContext;
    TSS_HTPM        hTPM;
    TSS_HPOLICY     hPolicy;
    TSS_HKEY        hSRK, hSealKey;
    TSS_HENCDATA    hSealData;
    TSS_HPCRS       hPCRs;
    BYTE            wellKnownSecret[] = TSS_WELL_KNOWN_SECRET;
    BYTE            rawData[64];
    UINT32          unsealedDataLength, pcrLength;
    BYTE           *unsealedData, *pcrValue;
    int             i;

    for (i = 0; i < 64; i++)
      rawData[i] = (BYTE) i;

/* create context and connect to TPM */

    Tspi_Context_Create(&hContext);
    Tspi_Context_Connect(hContext, NULL);
```

Seal to PCR Code (2)

```
/* create empty keys and data object */

      Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_RSAKEY, TSS_KEY_TSP_SRK,
                                        &hSRK);
      Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_RSAKEY,
                                        TSS_KEY_TYPE_STORAGE |
                                         TSS_KEY_SIZE_2048 |
                                         TSS_KEY_NO_AUTHORIZATION |
                                         TSS_KEY_NOT_MIGRATABLE,
                                        &hSealKey);

      Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_ENCDATA,
                                        TSS_ENCDATA_SEAL, &hSealData);

/* get TPM object */

      Tspi_Context_GetTpmObject(hContext, &hTPM);

/* set up the default policy - this will apply to all objects */

      Tspi_Context_GetDefaultPolicy(hContext, &hPolicy);
      Tspi_Policy_SetSecret(hPolicy, TSS_SECRET_MODE_SHA1, 20, wellKnownSecret);
```

Seal to PCR Code (3)

```
/* create and load the sealing key */

      Tspi_Key_CreateKey(hSealKey, hSRK, 0);
      Tspi_Key_LoadKey(hSealKey, hSRK);

/* seal to PCR values */

    /* set the PCR values to the current values in the TPM */

      Tspi_TPM_PcrRead(hTPM, 5, &pcrLength, &pcrValue);
      Tspi_PcrComposite_SetPcrValue(hPCRs, 5, pcrLength, pcrValue);
      Tspi_TPM_PcrRead(hTPM, 7, &pcrLength, &pcrValue);
      Tspi_PcrComposite_SetPcrValue(hPCRs, 7, pcrLength, pcrValue);

    /* perform the seal operation */

      Tspi_Data_Seal(hSealData, hSealKey, 64, rawData, hPCRs);

    /* unseal the blob */

    unsealedData = NULL;
      Tspi_Data_Unseal(hSealData, hSealKey, &unsealedDataLength, &unsealedData);
```

Seal to PCR Code (4)

```
        /* free memory */

        Tspi_Context_FreeMemory(hContext, unsealedData);

/* clean up */

        Tspi_Key_UnloadKey(hSealKey);

        Tspi_Context_CloseObject(hContext, hPCRs);
        Tspi_Context_CloseObject(hContext, hSealKey);
        Tspi_Context_CloseObject(hContext, hSealData);

/* close context */

        Tspi_Context_Close(hContext);

     return 0;
}
```

Sign/verify Code

```
int
main(void)
{
    TSS_HCONTEXT    hContext;
    TSS_HHASH       hHash;
    TSS_HKEY        hSigningKey, hSRK;
    TSS_HPOLICY     hPolicy;
    TSS_UUID        srkUUID = TSS_UUID_SRK;
    BYTE            secret[] = TSS_WELL_KNOWN_SECRET;
    UINT32          sigLen;
    BYTE            *sig;
    BYTE            hash[] =
      {0x32, 0xd1, 0x0c, 0x7b, 0x8c, 0xf9, 0x65, 0x70, 0xca, 0x04,
       0xce, 0x37, 0xf2, 0xa1, 0x9d, 0x84, 0x24, 0x0d, 0x3a, 0x89};

/* create context and connect */

       Tspi_Context_Create(&hContext);
       Tspi_Context_Connect(hContext, remote-pc);
```

Sign/verify Code (2)

```
/* create a signing key under the SRK */

      Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_POLICY,
                                        TSS_POLICY_USAGE, &hPolicy);
      Tspi_Policy_SetSecret(hPolicy, TSS_SECRET_MODE_SHA1, 20, secret);
      Tspi_Context_GetKeyByUUID(hContext, TSS_PS_TYPE_SYSTEM, srkUUID, &hSRK);
      Tspi_Policy_AssignToObject(hPolicy, hSRK);
      Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_RSAKEY,
                              TSS_KEY_TYPE_SIGNING |
                               TSS_KEY_SIZE_2048 |
                               TSS_KEY_AUTHORIZATION |
                               TSS_KEY_NOT_MIGRATABLE,
                              &hSigningKey);
      Tspi_Policy_AssignToObject(hPolicy, hSigningKey);
      Tspi_Key_CreateKey(hSigningKey, hSRK, 0);
      Tspi_Key_LoadKey(hSigningKey, hSRK);

/* open valid hash object */

      Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_HASH,
                                                              TSS_HASH_SHA1,
      &hHash);
```

Sign/verify Code (3)

```
/* set hash value and get valid signature */

       Tspi_Hash_SetHashValue(hHash, sizeof(hash), hash);
       Tspi_Hash_Sign(hHash, hSigningKey, &sigLen, &sig);

/* verify signature */

       Tspi_Hash_VerifySignature(hHash, hSigningKey, sigLen, sig);

/* free sig memory, close signing key object and context */

       Tspi_Context_FreeMemory(hContext, sig);
       Tspi_Context_CloseObject(hContext, hSigningKey);

/* close context */

       Tspi_Context_Close(hContext);

// we forgot to unload the signing key, but the TSS did it for us
// when we closed the context

    return 0;
}
```

Using This knowledge to Write Trusted Applications

- Given a TPM and TSS, choose correct ways to use functionality to meet security objectives
- Requires some security architecting, but the building blocks are all there
- So, let's look at our original motivating use cases . . .

Secure Key Storage

- How do I store a key securely, so a user can access it with a password?
  - Select a starting password
    - Set a policy with that password: Tspi_Policy_SetSecret
  - Get the key onto the platform
    - Create a key object with chosen parameters: Tspi_Context_CreateObject
    - Create a new one: Tspi_Key_CreateKey
    - . . . or import a known one: Tspi_Key_WrapKey
  - Optionally let the user change the password
    - Use Tspi_ChangeAuth
  - Store the key blob somewhere
    - Register it in the TSS Key Store: Tspi_Context_RegisterKey
    - Store the key blob somewhere else and load it when needed

Key Backup

- How do I back up a key securely, so that IT can help the user out with a forgotten password?
  - Can use remote connection to the TPM . . .
    - Tspi_Context_Connect(hContext, userMachineName);
  - . . . then import key using previous method, keeping a copy of the key
  - Or, use migration capabilities
    - Set up a trusted key to migrate to: Tspi_TPM_AuthorizeMigrationTicket
    - Begin migration process: Tspi_Key_CreateMigrationBlob
    - Complete migration process: Tspi_Key_ConvertMigrationBlob
  - That's it!

User and Machine Authentication

- How do I ensure that I am communicating with a particular user with access to a particular machine?
  - Choose TPM authentication method
    - Decryption of a challenge: Tspi_Data_Unbind
    - Signature of a challenge with selected configuration: Tspi_TPM_Quote
  - Set up a key for the user on a particular TPM
    - Use previous methods
    - Perhaps ensure that the key is non-migratable: TSS_KEY_NOT_MIGRATABLE
  - Require that the user use the key as a means to authenticate
    - Could leverage existing authentication mechanisms
      - Smart cards
      - VPN
      - Digital signature
    - Could use a new protocol if desired

TRUSTED COMPUTING GROUP™

Ntrū

Binding to a Specific Platform

- How do I make sure my software only runs on a specific machine?
  - Use the previous authentication mechanisms, but embed them in the application
    - Require a signature from a key tied to the platform before continuing: Tspi_Hash_Sign, Tspi_Hash_VerifySignature
  - Encrypt key data to the platform
    - Seal a symmetric key that the application needs to function: Tspi_Data_Seal, Tspi_Data_Unseal

Binding to a Platform State

- How do I make sure my software only runs on machines in a specific state?
  - Leverage secure use of PCRs
    - Could involve secure boot
    - Could involve use of locality
  - Leverage platform credentials
    - Assume that each EK has a certificate
    - Create identity keys for your application: Tspi_Key_CreateKey, Tspi_TPM_CollateIdentityRequest, Tspi_TPM_ActivateIdentity
    - Certify other keys (e.g. an encryption key) if necessary: Tspi_Key_CertifyKey
  - Require platform authentication of state before allowing software to be run
    - Encrypt software and require certified key to decrypt when in correct state using previous methods
    - Require signature of state before continuing application: Tspi_TPM_Quote

TRUSTED COMPUTING GROUP™

Ntrū

Conclusions

- TCG technologies provide a very rich set of functionality to implement security features
- Early application writers will need to understand a few of the specifics of TCG technologies
- The TSS makes it easier to properly use the TPM
- The increasing deployment trend of TCG technologies will allow for more and more trusted applications to come into existence

Questions?

Contact Info:

Ari Singer, NTRU Cryptosystems

asinger@ntru.com