From TPM 1.2 to TPM 2.0

Liqun Chen
(Graeme Proudler)
HP Labs, UK
TPM 1.2 – Brief Historical Perspective

- **TPM 1.2 specification released in Oct, 2003 (version 62)**
- **Limited crypto**
  - One hash algorithm – SHA1 + HMAC
  - One asymmetric algorithm – RSA (ENC, SIG, DAA); no block cipher
- **Targeted at PC**
  - Protection from malware from the network
  - Protection for “stolen laptop”
- **Accepted as an ISO standard in 2009**
  - ISO/IEC 11889
  - Based on version 103 of the TPM specification
- **100’s of millions have been shipped**
Changes Have Become Necessary (I)

SHA1 is showing signs of weakness and is being deprecated
NIST and ISO have made action to respond it

Different geographies are selecting different alternatives
Nobody trusts anybody else’s algorithms

Shift from RSA to ECC for asymmetric cryptography
World’s infrastructures still use a lot of RSA and will for some time
A change is happening
ECC allows parameterization, have seen geographic differentiation
Reaction: Algorithm Agility

TPM 1.2 supports
- RSA encryption
- RSA signature
- RSA-DAA
- SHA-1
- HMAC
- One-time-pad with XOR
- AES (optional)

TPM 2.0 supports
- RSA encryption and signature
- ECC encryption and signature
- ECC-DAA
- ECDH
- SHA-1, SHA-256
- HMAC
- AES and one-time-pad with XOR

Manufacturer can add any algorithms with TCG IDs
- e.g., SM2, SM3, SM4
Changes Have Become Necessary (II)

Finding that TPMs aren’t just for PCs
Anything that attaches to the network has similar problems and sharing a solution would be more economical than fragmented solutions
Broader application base brings additional requirements

Finding that TPMs aren’t just used by the users
Platforms need to use TPMs

People will not protect their own security if you make it too hard or expensive
Unfortunately the TPMv1.2 controls tend to make TPMv1.2 difficult to use once it is fitted
Reaction: Separation of Functions

- In TPM 1.2, everything is under the control of the “Owner”
  - If the TPM is not enabled, activated, and owned; there isn’t much that can be done with it
  - If you are the Owner, you control both the security and privacy functions

- In TPM 2.0, there are three separate domains
  - Security – functions that protect the security of the user
  - Privacy – functions that expose the identity of the platform/user
  - Platform – functions that protect the integrity of the platform/firmware services

- Each domain has its own resources and controls
  - Security – ownerAuth, storage hierarchy, hierarchy enable (shEnable)
  - Privacy – endorsementAuth, endorsement hierarchy, ehEnable
  - Platform – platformAuth, platform hierarchy, phEnable
Three Hierarchies Instead of One

TPM 1.2
A single hierarchy –
• Storage hierarchy
  - for platform user

TPM 2.0
Three hierarchies –
• Platform hierarchy
  - for platform firmware
• Storage hierarchy
  - for platform user
• Endorsement hierarchy
  - for platform administrator
Applying Lessons Learned

- The major problem is not many people having used TPM 1.2?!

- TPM 2.0 Specification Needs to Address
  - Controllability
  - Cryptographic algorithm variability
  - TPM management and privacy
  - Remove redundant/unused/costly features
  - Additional features for new devices and applications

- Do not want to increase the cost from TPM 1.2 !!!
  - TPM 1.2 and TPM 2.0 are not compatible to each other
Control and Management
Control States – Security

- The security domain is used for those functions that, in TPM 1.2, are controlled by ownerAuth.

- ownerAuth can be used to:
  - Allocate NonVolatile memory resources
  - Initialize the TPM – TPM2_Clear()
  - Can be used to turn itself off – shEnable
  - Change ownerAuth and set an ownerPolicy

- ownerAuth cannot control the endorsement hierarchy.

- When shEnable is off, use of things in the storage hierarchy is disabled
  - Keys and NonVolatile memory

- shEnable is turned on each time the system is booted
  - State of shEnable is not changed by TPM Resume
Control States – Privacy

- **Endorsement Keys**
  - TPM 2.0 generates certain key pairs from a seed
  - TPM vendor may generate certificates for Endorsement keys (EK) that would be generated by the TPM and ship the TPM without any EK installed on the TPM
  - The privacy administrator, not the security administrator, decides if EK are created

- **Attestations**
  - The term “attestation” applies to the use of a TPM key to sign values contained in the TPM
  - Quote PCR, Certify other keys and NonVolatile memory indexes, Sign audit logs, Timestamp
  - Firmware revision
  - Reset and restart counters
Control States – Platform

- When the platform boots, the platform hierarchy is enabled and **platformAuth** is set to a new value
  - Allows use of the TPM to ensure the integrity of the firmware
  - This is not a capability that should be under control of the user, so it isn’t

- **platformAuth** can be used to:
  - Allocate NonVolatile memory resources
  - Initialize the TPM
  - Control the enables of the other hierarchies
    - Doesn’t give platform ability to use other hierarchies, just the ability to deny use of those hierarchies to others

- Before platform firmware turns control of system to OS, **phEnable** can be turned off or **platformAuth** can be randomized
  - **platformAuth** would be placed in secure location (SMM) so that only platform firmware would be able to access it
Management Strategies

- There is no single strategy for managing the TPM for every platform in every segment so the TPM 2.0 specification does not mandate one.

- The controls are such that the platform decides which functions are available to the OS.
  - Controls are determined at boot by platform selection of the settings of $shEnable$ and $ehEnable$.

- Platform-specific TPM specifications might have mandates for the default settings.
  - A Personal Computer may select that $shEnable$ is on by default but $ehEnable$ is off.
  - Platform may provide a UI to allow owner to change these defaults.
  - A Mobile Phone may select that both $shEnable$ and $ehEnable$ are on by default.
  - May be no way to change these defaults without permission from service provider.
No OFF State

- Even if all enables are off, the TPM continues to function
  - This enables control access to associated resources but do not prevent commands from running

- With all enables off, the function of the TPM is indistinguishable from software
  - Acts just like a software service loaded by the OS

- Some reasons for leaving TPM on:
  - In some instances, the TPM is built around a crypto accelerator and access to the accelerator is through the TPM API
  - Want to get random numbers from the TPM
  - The TPM may implement algorithms that require a license and using the TPM may avoid additional licenses
Platform Configuration Registers
Recording measurements in a TPM

Platform Configuration Register → Input → TPM2_Extend → Hash algorithm (SHA-1/SHA-2/… → Append
Normal Use for PCR

- The most common use of PCR is to record the measurements/hashes of the code that was run during system initialization.

- If the same measurements are made each time the system is booted, the PCR will have familiar values:
  - This does not just mean that the same code was run but that the same code was run in the same order.

- Microsoft uses the PCR when unsealing a BitLocker key protector:
  - If the PCR do not have a familiar value, disk encryption key will not be unsealed.
    - User may enter their recovery key if they think that the system is in a trustworthy state.
  - This process is “brittle” – any change in boot requires user intervention.
PCR Banks

- The TPM 2.0 specification allows a TPM to have multiple banks of PCRs
  - Within a bank, all PCR are extended using the same hash algorithm

- The banks do not have to have the same number of PCR
  - Example: one bank might have PCR 1, 2, 4, 5, and 8 while a different bank might have PCR 1, 2, 3, 4, 6, and 7

- Can extend PCR individually
  - Example: Could extend PCR0 in SHA1 bank with SHA1 hash but leave PCR0 of SHA256 bank alone

- TPM will extend the identified PCR in each bank with the digest for that bank
  - SHA1 digest to SHA1 PCR, SHA256 digest to SHA256 PCR

\[
\text{PCR.digest[pcrNum][alg]_{new}} := H_{\text{alg}}(\text{PCR.digest[pcrNum][alg]_{old} || digest})
\]
Authorisation via policies
Policy Logic

- A policy can be written an equation using AND (&) and OR (|) for each element:
  \[(A \& B \& C) \mid (D \& E \& F)\]

- Roughly speaking, the left side is evaluated as:
  \[\text{digest}_{\text{left}} := H( H( H(0 || A) || B) || C)\]

- And the right side as:
  \[\text{digest}_{\text{right}} := H( H( H(0 || D) || E) || F)\]

Note: \(H( H( H(0 || C) || B) || A)\) would give a completely different result for \(\text{digest}_{\text{left}}\)
TPM2_PolicyAuthorize() – Flexible Policies

- As we learned with TPM 1.2, basing policy on PCR makes the policy “brittle”
  - Any change to PCR will break the policy even if the change is benign
- TPM2_PolicyAuthorize() allow a policy to have an “authority” determine if some policy is OK rather than have the policy hardwired in
- Example:
  - The policyHash representing a set of good PCR is known to the OEM
  - The OEM signs a digest that represents the policyHash representing the good PCR and distributes it along with their BIOS update
  - The user can create a policy that says, “if the OEM approves the PCR settings they are OK with me”
  - Use TPM2_PolicyPCR() to set the policyHash to the current value of the PCR and then use TPM2_PolicyAuthorize() to apply the OEM’s stamp of approval to those PCR
TPM 2.0 Objects Defined by:

- the way that their public and sensitive areas are cryptographically bound
  - RSA object – public key evenly divisible by the sensitive prime
  - ECC object – public key is the product of the private key times the generator
  - Symmetric object – public identity is hash of the secret data and a secret masking value

- their cryptographic uses
  - Sign
  - Encrypt
  - None

- and their constraints
  - Restricted
**Restricted Attribute**

- The TPM will only use the key on TPM-generated data or on data that is formatted specifically for TPM-consumption
  - *Sign* – operate on TPM-generated data
  - *Decrypt* – operate on TPM-formatted data
- The TPM will only use the default schemes defined for the key
### Combining Cryptographic Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Nominal Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>External data that is protected (via bind or seal) by the hierarchy</td>
</tr>
<tr>
<td>0 0 1</td>
<td>A key for protecting data</td>
</tr>
<tr>
<td>0 1 0</td>
<td>A key for signing data</td>
</tr>
<tr>
<td>0 1 1</td>
<td>A key for protecting and signing external data</td>
</tr>
<tr>
<td>1 0 1</td>
<td>A storage key, for constructing the hierarchy</td>
</tr>
<tr>
<td>1 1 0</td>
<td>A key for signing TPM data (certificates, quotes) (an “AK”)</td>
</tr>
<tr>
<td>1 0 0</td>
<td>Forbidden combination (don’t know what it means)</td>
</tr>
<tr>
<td>1 1 1</td>
<td>Forbidden combination (it has no useful purpose and is incompatible with the USA’s FIPS specifications)</td>
</tr>
</tbody>
</table>
Hierarchies of TPM 2.0 objects
Build a Hierarchy from a Primary Seed

**The Primary Seed**

This line indicates from which PS the key was derived.

The key is derived from the PS and the parameters of the created key.

TPM

The KDF is a one-way function that will ‘whiten’ the PS based on input parameters, one of which is the Primary Seed.

A primary object does not leave the confines of the TPM except as a saved context blob.
Any number or types of keys may be created from the Primary Seed.

These are all reproducible “aliases” for the Primary Seed and are called Primary Keys or Primary Objects.
Extended Hierarchy

Now have hierarchy with a depth of two

The outer shell indicates that the sensitive data of the object is protected, using values derived from the seed, when the object is stored off chip.

The double-headed arrow represents a connection that can be changed.
Storage Keys Build the Hierarchy

Because the protection values for an object are derived from a seed value in a ‘parent’ storage key, the ‘child’ can’t be decrypted and loaded into the TPM unless the parent is in the TPM. This is what establishes the hierarchical relationship.
Duplication from one TPM to Another

An object that can be imported can also be duplicated and moved to a different TPM.

After duplication, it exists in two places.

If the object is a Storage Key, all of its child keys are moved at the same time.
Endorsement and Attestation
EK Provisioning

- Need an EK for identification of the TPM
- Easy when everyone is happy using the same algorithms
- When we need different algorithms, the problem for the TPM manufacturer gets much harder
  - Each key-pair takes up NV memory
- What key pair does the TPM vendor install?
EK Provisioning Solution – Primary Seed

- Instead of a set of asymmetric keys, just have one master seed value from which we can generate any number of asymmetric or symmetric keys.

- Allow the process of creating an asymmetric key to be reproducible – when called with the same parameters, the same key is generated.

- The asymmetric keys are created by using the Primary Seed and set of caller-provided parameters as inputs to a standard key derivation function (KDF).

- The asymmetric keys become ‘aliases’ of the Primary Seed.
User Endorsement Provisioning

- TPM manufacturer EK for various markets and certifies them
  - Manufacturer is allowed to inject a primary seed
- TPM shipped anywhere without EK installed
- User creates an EK using the same template as the TPM vendor
  - Same parameters yields the same key
- User could tell the TPM to make the key persistent
  - Store in NV
- User will have an EK that the TPM vendor could have certified
- This process is enabled, but NOT mandated, by the spec
When the TPM creates an attestation blob, it will prepend the “magic” value to indicate that it is a TPM-created and then digest the block.

0xFF, ‘TCG’ =

The TPM then signs the digest with the specified key.
Digital Signatures in TPM 2.0
Digital Signatures in TPM 2.0

• Conventional signature
• DAA
• U-Prove

Can we implement them with a single TPM signature operation?

Can we do it very efficiently – use restricted TPM resources?
Two Observations

1. In a complex signature scheme, like DAA and U-Prove, the computation directly using a private key is only a small part of the computation of the signature.

2. The computation using the private key is a self-contained signature operation that can be used by different pieces of software to create different/multiple complicated signatures.
## TPM 2.0 Signature Primitive – tpm.sign

<table>
<thead>
<tr>
<th>Key generation</th>
<th>Signing</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Private key</strong></td>
<td>$m \in {0, 1}^<em>$, $str \in {0, 1}^</em>$, $P_1 \in G$.</td>
<td>$(P_1, P_2, R_1, R_2, K_2, c, s)$, $m, K_1$.</td>
</tr>
<tr>
<td>$x \leftarrow Z_p^*$</td>
<td><strong>Commit oracle:</strong>&lt;br&gt; If $str = 0$, set $P_2 = 1$,</td>
<td>If $P_1 = P_2 = 1$,</td>
</tr>
<tr>
<td><strong>Public key</strong></td>
<td>else $P_2 = H_G(str)$.&lt;br&gt;$r \leftarrow Z_p^*$, $R_1 = P_1^r$, $R_2 = P_2^r$, $K_2 = P_2^x$.</td>
<td>$H(R_1, R_2, m) \neq c$,</td>
</tr>
<tr>
<td>$y = g^x \in G$</td>
<td><strong>Sign oracle:</strong>&lt;br&gt;$c = H(R_1, R_2, m)$.&lt;br&gt;$s = r + cx \mod p$ and erase $r$.</td>
<td>$R_1 \neq P_1^s K_1^{-c}$, or</td>
</tr>
<tr>
<td></td>
<td>Signature = $(P_1, P_2, R_1, R_2, K_2, c, s)$.</td>
<td>$R_2 \neq P_2^s K_2^{-c}$,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>return invalid,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Else,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>return valid.</td>
</tr>
</tbody>
</table>
Security Analysis of tpm.sign

The tpm.sign scheme is existentially unforgeable under a chosen-message attack.

Security proof under
• Random oracle model
• The static DH assumption
Implementation of tpm.sign

The tpm.sign scheme is implemented by using

TPM2_Create() – create a key
TPM2_ActivateCredential() – certify a key
TPM2_Load() – load a key into the TPM
TPM2_Commit() – commit a “new” version of the key
TPM2_Sign() – sign a message with the committed key

The committed new version of a key provides the properties of anonymity, linkability, pseudonym credential, etc.
Application 1: DAA in TPM 2.0

General description of DAA
• $ik = (isk, ipk)$
  – issuer’s secret and public key
• $tk = (tsk, tpk)$
  – TPM’s secret and public key
• $cre$
  – DAA issuer’s signature on $(tsk, tpk)$
• $cre'$
  – randomized $cre$
• $\sigma_t$
  – TPM’s signature ($tpm.sign$)
• $SPK\{(tsk, cre): cre' \land \sigma_t\}(msg, bsn, n_v)$

Two ECDAA schemes are supported by TPM 2.0
• LRSW-DAA
• SDH-DAA

Privacy requirement
$cre'$ or $\sigma_t$ does not reveal $cre$ or $tpk$
Application 2: U-Prove with TPM as a Protected Device

U-Prove is a Microsoft pseudonym system

A U-Prove token is \((h, \sigma)\):

\[
h = \left( g_0 g_1^{x_1} \cdots g_n^{x_n} g_t^{x_t} g_d^{x_d} \right)^{\alpha}
\]

\(\sigma\) is a blind signature on \(h\).

TPM takes the same operations as in DAA – tpm.sign.

Here \(x_d\) is TPM’s key, and is proved by TPM.
The TPM 2.0 specification
In TPM 1.2, the detailed actions for a command (what the TPM was supposed to do) was written in English, in a sort of pseudo code.

Tried to do this with TPM 2.0 and found that the pseudo code was hiding important implementation implications.

The complexity of adding algorithm variability made it uncertain if it would ever be “right”.

A stylized version of C was chosen to describe the behavior of the TPM.

Auxiliary functions written to allow a full reference implementation in software.
Specification Structure

- **Part 1 is the informative description of a TPM and its methods**
- **Part 2 contains the normative definition of the interface elements**
  - Tables used to define the structures
  - Table annotations allow automated tools to extract the necessary C-code structure definitions and generate the marshaling and unmarshaling code
- **Part 3 is the normative definition of the TPM commands**
  - Narrative description of each command
  - Tables defining the interface parameters (commands and response)
  - Detailed actions written in C-code
- **Part 4 is an informative section that is almost all C-code**
  - Contains major subsystems that are implementation dependent (e.g., NV memory)
Thank you