TPM Keys
Creating, Certifying, and Using Them

Ariel Segall
ariels@alum.mit.edu

Day 1

Approved for Public Release: 12-2749.
Distribution unlimited

License

All materials are licensed under a Creative Commons "Share Alike" license.

- http://creativecommons.org/licenses/by-sa/3.0

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

to make commercial use of the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

What We'll Be Covering

- Creating TPM Keys
- Certifying TPM Keys
- Using TPM Keys

Creating TPM Keys

First question: What kind of key do you need?

Key creation is split into two:[1]:

- Identity key creation
- Everything else

[1] Not *quite* true, but certifiable migratable keys are beyond the scope of this class.

Quick Review: Types of TPM Keys

- Identity (AIK): sign data from the TPM, such as quotes or certificates
  - A TPM can have many identities!
- Signing: sign user data
- Storage: encrypt data, including other keys
- Binding: decrypt data (usually from remote platforms)
- Legacy: signing or encryption
  - Lower security for backwards compatibility; not recommended
  - Only usable in some commands
  - Not creatable in FIPS mode

Quick Review: Types of TPM Keys

- Identity (AIK): sign data from the TPM, such as quotes or certificates
  - A TPM can have many identities!
- Signing: sign user data
- Storage: encrypt data, including other keys
- Binding: decrypt data (usually from remote platforms)
- Legacy: signing or encryption
  - Lower security for backwards compatibility; not recommended
  - Only usable in some commands
  - Not creatable in FIPS mode

Wrap keys!

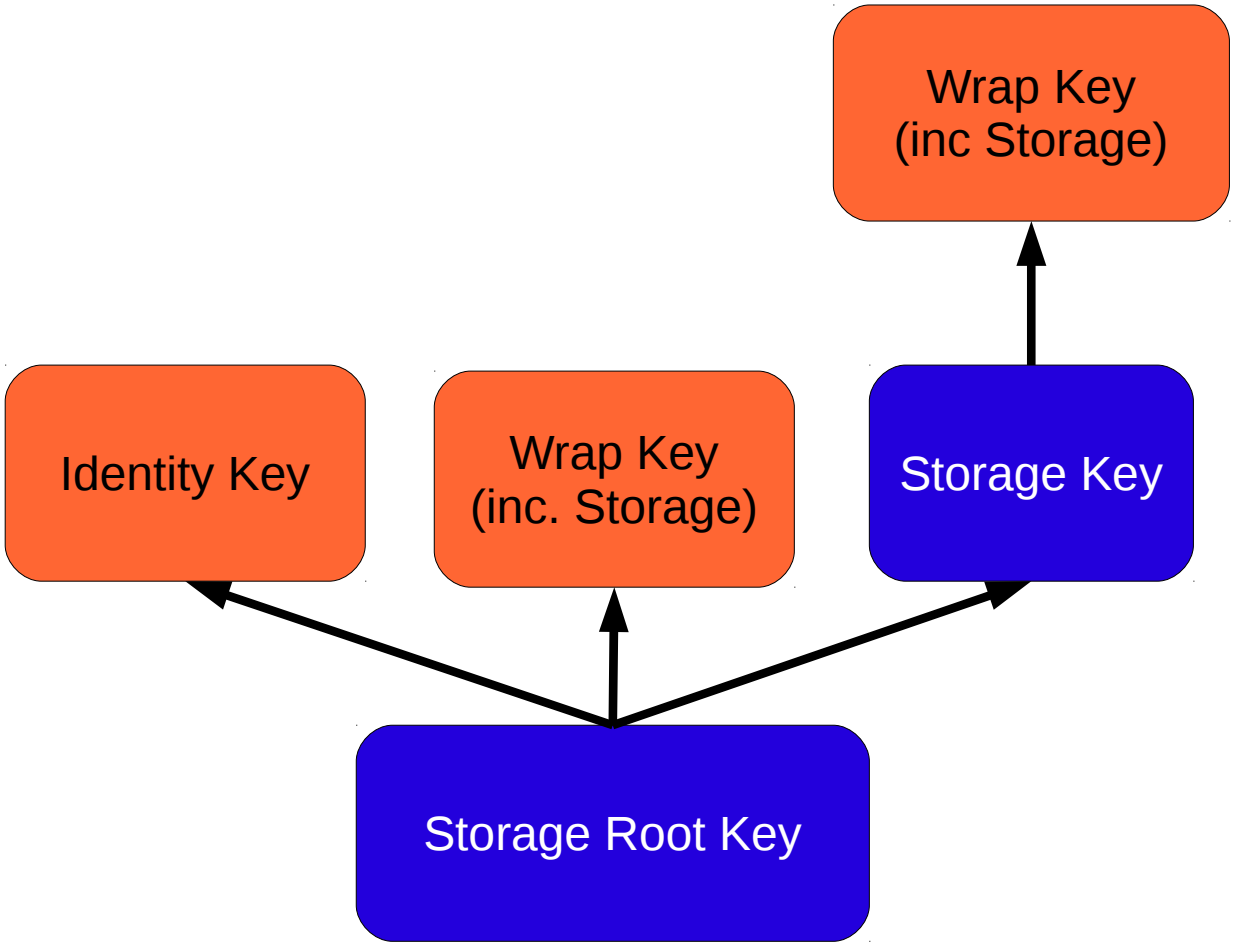
Creating Wrap Keys

- `TPM_CreateWrapKey`
- Offers many choices
  - Key type: Any except identity
  - Key size: 512-2048 bits
  - Authorization values to use, if any (key password)
  - PCR and/or locality constraints, if any
  - Migratable (can be exported) or non-migratable (this TPM only)
  - Some key-specific options
    - e.g., Signing keys have three signature scheme options
- Must provide a *parent* key, already loaded into TPM
  - Storage key that will be used to encrypt private portion
  - SRK is a storage key!
- Outputs a key blob, which user responsible for storing

Creating Identity Keys

- `TPM_MakeIdentity`
- TPM owner must authorize command!
- Only choices:
  - Authorization values to use (key password)
  - PCR and/or locality constraints, if any
  - Privacy CA to create signing request for
    - Privacy CAs in the next section
- All identity keys are created to meet minimum security requirements
  - Non-migratable
  - 2048 bits
  - SRK parent
- Outputs key blob and certificate signing request
  - Note: key blob same format as CreateWrapKey output!
  - CSR is not x.509 standard; custom AIK certification protocol

Key Storage Hierarchy Illustration



What We'll Be Covering

- Creating TPM Keys
- Certifying TPM Keys
- Using TPM Keys

Certifying TPM Keys

A very complex subject!

- TCG has well-designed, high-security, verifiable protocols
- Completely non-standard!
- Not supported by today's commercial CAs
- TPM keys *cannot* participate in standard certification protocols

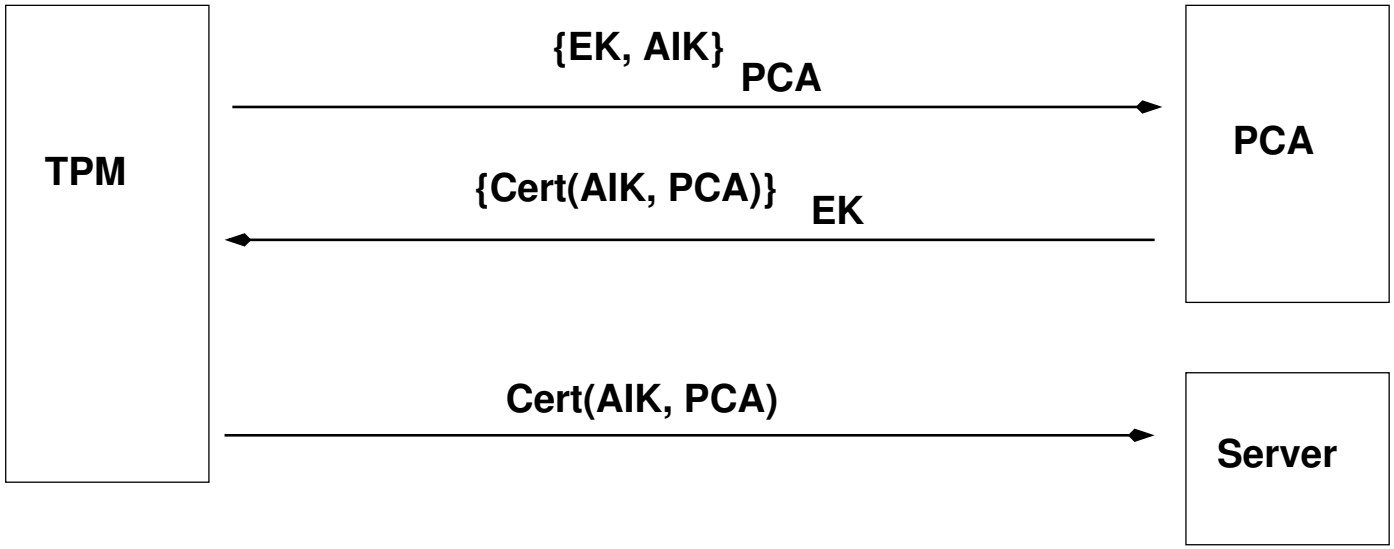
Discussing Certification

- Explore TCG's certification design
- Enterprise PKI and commercial CA expectations
- Compromises and recommendations

TCG Certification Vision

- TPM EK certified by manufacturer
- Identity keys tied to EK with pseudonymizing PCA protocol
- Identity keys certify other TPM keys using CertifyKey certificates
- Strong cryptographic bindings at each step!
- TPM key heirarchies can be verified without CA interaction per-key

PCA Protocol

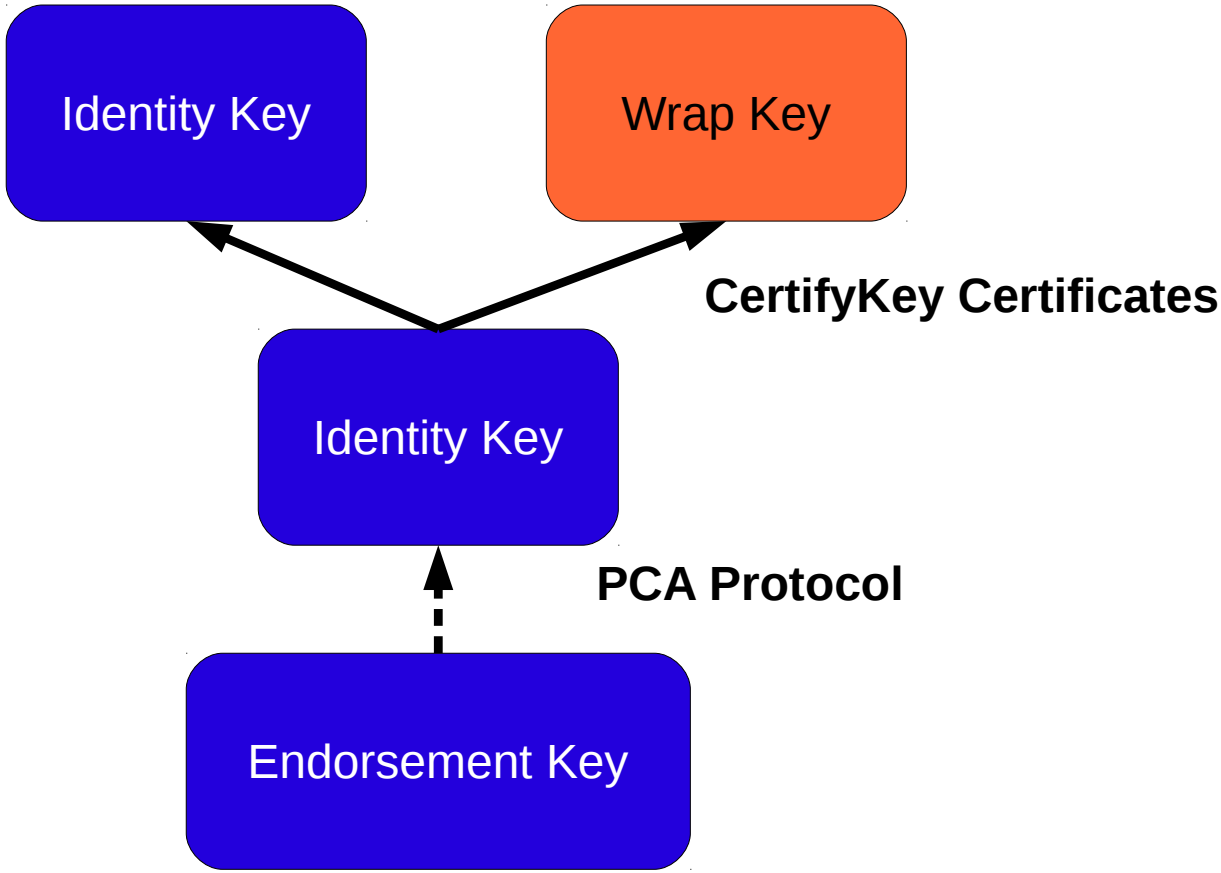


- PCA: Privacy Certificate Authority
- Issues certificate for *any* AIK if EK legitimate
  - EK will only decrypt cert if AIK actually in TPM
  - `ActivateIdentity` command decrypts certs
- Only PCA (trusted party) can associate EK and AIK
  - PCA cannot prove association unless sees cert
  - Recipients cannot associate different AIKs from same TPM
- Strong cryptographic binding between EK and AIK; pseudonymity

CertifyKey

- `TPM_CertifyKey` command: issue cert for TPM key
- Non-migratable keys only!
- Claim: "This is a genuine TPM key with the following properties"
- Certificate includes:
  - Key type
  - Key length
  - PCR and locality constraints
  - . . . i.e., anything a remote party would need to know to trust!
- Signed by AIK
  - Can be signed by signing or legacy key, but not trustworthy

Key Certification Hierarchy Illustration



Discussing Certification

- Explore TCG's certification design
- Enterprise PKI and commercial CA expectations
- Compromises and recommendations

What Enterprises Generally Expect

- x.509!
  - Certificate Signing Requests are self-signed
  - Certificates are public; CAs often publish them directly
- World divided into CAs, which certify all keys; and other entities, which may not certify anything
- Standard protocols, supported by commercial CAs

Where Problems Arise

- Most TPM keys are incapable of creating a self-signed CSR
  - Breaks x.509 request protocols
- CA-published certificates would break PCA protocol entirely
  - Security relies on the secrecy of the certificate until TPM decrypts
  - A simple CSR format module isn't sufficient to fix
  - PCA protocol only way to reliably certify AIKs post-provisioning!
- TPM CertifyKey certificates break the model
  - AIKs are CAs, but only for this TPM
  - Completely non-standard format to boot: apps won't recognize

Discussing Certification

- Explore TCG's certification design
- Enterprise PKI and commercial CA expectations
- Compromises and recommendations

Solving Certification: Temporary Patch

- Keys of any type can be certified during provisioning
  - Modular CA update: non-self-signed CSRs
  - Some CAs support variations already
- Must be implemented regardless, for certifying EKs!
- Equally trustworthy, if done at same time in same environment

Solving Certification: Longer Term

- Add extensions to commercial CAs to support PCA protocol
  - Or separate CA operating in non-standard mode
- Add CA extension that treats CertifyKey certs as special CSRs
  - Issue x.509 certs based on verified CertifyKey certs
  - Include CertifyKey info or even full cert
  - Aware apps can utilize; legacy apps can ignore

What We'll Be Covering

- Creating TPM Keys
- Certifying TPM Keys
- Using TPM Keys

Using TPM Keys

- We will *not* be covering what to use each key for yet.
  - Specialized per key; we'll cover in detail tomorrow
- This section just covers common techniques for using any TPM key

Quick Review: Key Storage

- The EK is stored in the TPM permanently
  - Never used directly
- The SRK is stored in the TPM permanently
  - Predefined location in key storage
- Other keys (identity, wrap) are stored in partially-encrypted blobs on disk
  - "Disk" is generalization– anywhere outside the TPM will work
  - For some apps, might not even be local to machine!

Loading Keys (1/2)

In order for a key to be used, it must first be loaded into the TPM.

- Loading decrypts the private half and places it in volatile storage
- User receives a *key handle* pointing to the key's location
- Handle can be provided for any future commands using the key
- *Handles are volatile*.
  - Keys may be removed from storage on reboot
  - Keys cycled out if key storage filled
- Most TPM services do behind-the-scenes key and handle management, but keys must still be explicitly loaded to use
  - As a result, handle manipulation is a potential attack
  - Recommend not reusing authorization values for high-value keys

Loading Keys (2/2)

- `TPM_LoadKey2`[2]
- Two arguments: the key (blob) to be loaded, and the parent
  - Parent is handle of an already loaded key (often SRK)
  - Used to decrypt private half
- TPM performs integrity and sanity checks, then loads key and returns handle
- Important note: This counts as using the **parent**, not the key being loaded
  - Constraints on keys, like PCR values, are checked on use
  - A key can be loaded if constraints are violated; just can't do anything

[2] LoadKey deprecated

Using Keys

- Once a key is loaded, the handle may be provided to other TPM commands
- Many TPM commands will only accept certain kinds of keys
  - Cannot sign with a storage key, or use signing key as parent
- Every time a key is used:
  - Any PCR constraints must be met
  - Any locality constraints must be met
  - Any authorization data must be provided

For the rest of the class, when we say "Use a X key", we mean "Have an X key, load it, and provide it to the relevant command".

Questions?