Algorithm Agility

– Discussion on TPM 2.0 ECC Functionalities

Liqun Chen (University of Surrey)
Rainer Urian (Infineon Technologies)

SSR 2016
December 5 – 6, 2016

Trusted Platform Modules (TPMs)

– TPM specifications were developed by the Trusted Computing Group

– TPMs are used as a cryptographic engine in various computers

– Over a billion TPMs have been shipped

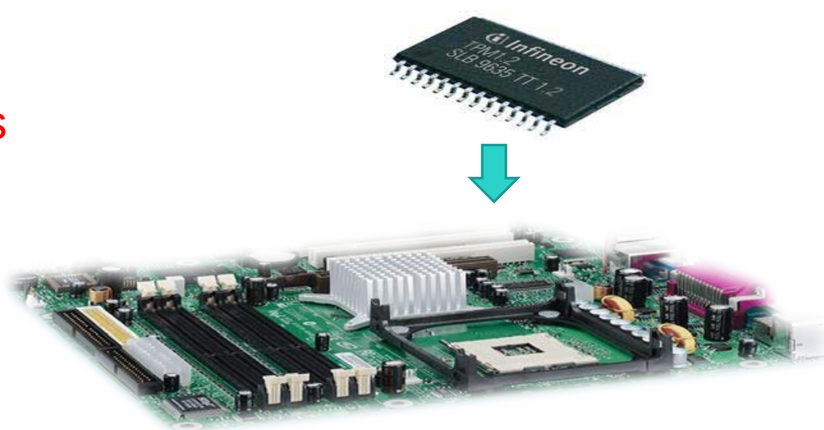
– A number of major applications, e.g. Microsoft BitLocker, FIDO and Secure Boot

– ISO/IEC 11889

– Two versions of TPMs:

  – TPM v1.2, supporting limited algorithms

  – TPM v2.0, supporting algorithm agility



Why is algorithm agility necessary?

Cryptographic algorithms in TPM 1.2

TPM 1.2 only supports a few cryptographic algorithms

- One hash algorithm –
  SHA1 (also used for HMAC)

- One asymmetric algorithm –
  RSA (for encryption and signature)

- One specially designed privacy-preserving signature algorithm –
  DAA (direct anonymous attestation)

- AES (not included in the early versions) and
  one-time-pad with XOR

Necessary Changes to TPM 1.2

The following are views on 2005 ……

– SHA1: signs of weakness and it is being deprecated

– NIST and ISO's action to respond

– Different geographies want different algorithms to be available

– Nobody trusts anybody else's algorithms

– Support the shift from RSA to ECC for asymmetric cryptography

– World's infrastructures still use a lot of RSA

– It was expected that change was happening

TCG's Reaction: Algorithm Agility in TPM 2.0

Each primitive can be implemented with different algorithms

- Mandatory algorithms:
  - RSA encryption and signature
  - ECC encryption and signature
  - ECC-DAA (RSA-DAA is no longer supported)
  - SHA-1 (not for signatures), SHA-256 and HMAC
  - AES and one-time-pad with XOR

- TCG Algorithm Registry

- Manufacturer can add any algorithms, e.g.,
  - China: SM2, SM3, SM4
  - Banks: Triple DES

How to achieve algorithm agility?

A naïve solution

– Each algorithm is implemented individually with specific commands

Any problem with this solution?

– Inflexible: many TPM versions are not compatible to each other

– Bad manageability: the specification can be too complex

– Bad performance: TPMs need to figure out which algorithm to perform

– Too expensive: it is not affordable

The TCG solution

– Each primitive is implemented with multiple choices of algorithms

– Multiple algorithms share the same set of TPM commands

Example: TPM2_Sign()

– RSA signature

– ECDSA

– EC-Schnorr

– SM2

– ECDAA

  – CL-ECDAA

  – q-SDH-ECDAA

– ……

What does this paper introduce?

Overview of TPM 2.0 Functionalities

– TPM commands for key handling:
  - TPM2_Create()
  - TPM2_Load()

– TPM commands for cryptographic algorithms:
  - TPM2_Commit()
  - TPM2_Sign()
  - TPM2_ECDH_KeyGen()
  - TPM2_ECDH_ZGen()

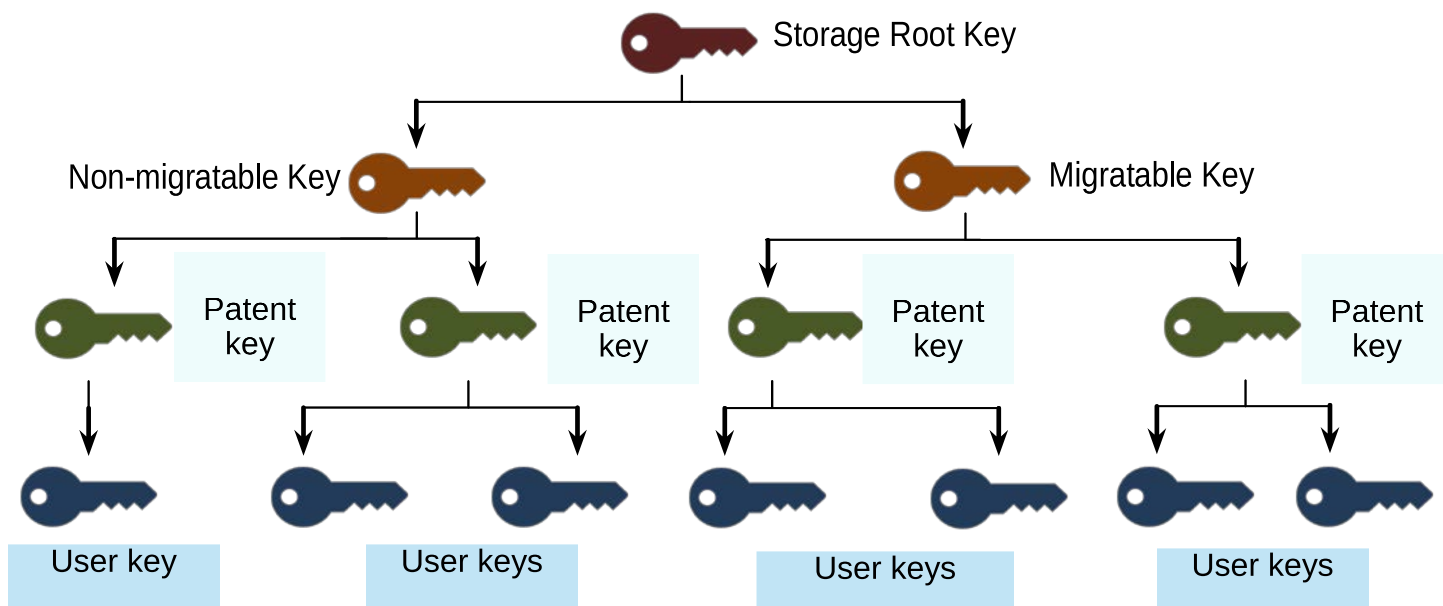
Overview of TPM 2.0 Key Structures

TPM key structure: keys are stored in a hey hierarchy

– key.name – external identity

– key.handle – internal identity

– key.blob – $(tsk)_{ek}||tpk||mac_{mk}((tsk)_{ek}||tpk.name)$; (ek, mk) = kdf(parentK)



Known ECC Cryptographic Use Cases for the TPM 2.0

- Conventional digital signatures
- Direct Anonymous Attestation (DAA)
- DAA with attributes (DAA-A)
- U-Prove
- Key exchange

New ECC Use Cases for the TPM 2.0

Asymmetric encryption

(Key Encapsulation Mechanism – KEM)

Four algorithms specified in ISO/IEC 18033-2:

– ECIES (Elliptic Curve Integrated Encryption Scheme)

– PSEC (Provably Secure Elliptic Curve encryption)

– ACE (Advanced Cryptographic Engine)

– FACE (Fast ACE)

ECIES (Elliptic Curve Integrated Encryption Scheme)

	KEM.KeyGen(q,G)	KEM.Encrypt(pk)	KEM.Decrypt(sk, C)
ECIES	$x \in [1,q)$ $Y = [x]G$ $sk \leftarrow x$ $pk \leftarrow Y$ Return (pk, sk)	$r \in [1, q)$ $C = [r]G$ $D = [r]Y$ $K = kdf(C\|\|D)$ Return (K, C)	$D = [x]C$ $K = kdf(C\|\|D)$ Return K

– In KEM.KeyGen(), perform

  – choose a parentK, run TPM2_Create(), return key.blob

    $(ek, mk) = kdf(parentK); (x)_{ek}||Y||mac_{mk}((x)_{ek}||key.name)$

– In KEM.Decrypt(), perform

  – TPM2_Load(key.blob)

  – TPM2_ECDH_ZGen(C), return [x]C

FACE (Fast Advanced Cryptographic Engine)

	KEM.KeyGen(q,G_1)	KEM.Encrypt(pk)	KEM.Decrypt(sk, C)
F A C E	$a_1,a_2 \in [0,q)$ $G_1 = [a_1]G$ $G_2 = [a_2]G$ $x_1,x_2,y_1,y_2 \in [0,q)$ $C = [x_1]G_1 + [x_2]G_2$ $D = [y_1]G_1 + [y_2]G_2$ sk $\leftarrow (x_1, x_2, y_1, y_2)$ pk $\leftarrow$ (C,D) Return (pk, sk)	$r \in [0, q)$ $U_1 = [r]G_1$ $U_2 = [r]G_2$ $\alpha = hash(U_1 \parallel U_2)$ $r' = \alpha \cdot r \bmod q$ $V = [r]C + [r']D$ $K \parallel T = kdf(V)$ $C = U_1 \parallel U_2 \parallel T$ Return (K, C)	Parse $C = U_1 \parallel U_2 \parallel T$ $\alpha = hash(U_1 \parallel U_2)$ $t_1 = x_1 + y_1 \cdot \alpha \bmod q$ $t_2 = x_2 + y_2 \cdot \alpha \bmod q$ $V = t_1 \cdot U_1 + t_2 \cdot U_2$ $K \parallel T' = kdf(V)$ Return K, if $T = T'$ Otherwise, return Fail

– In KEM.KeyGen(), call

  – TPM2_Create() 4 time to get $[x_1]G$, $[x_2]G$, $[y_1]G$, $[y_2]G$

  – TPM2_ECDH_KeyGen() twice to get G_1 and G_2

  – TPM2_ECDH_ZGen() 4 times to get $[x_1]G_1$, $[x_2]G_2$, $[y_1]G_1$, $[y_2]G_2$

– In KEM.Decrypt(), call TPM2_ECDH_ZGen() 4 time to get

$$X_1 = [x_1]U_1,\ X_2 = [x_2]U_2,\ Y_1 = [y_1]U_1,\ Y_2 = [y_2]U_2$$

Discussion on

– Limitations of algorithm agility, for EC digital signatures

- ECDSA, EC-GDSA, EC-KCDSA, EC-RDSA, SM2
- TPM implementation of these algorithms are not much integrated

– Compatibility issue

- EC-Schnorr in ISO/IEC 14888-3, ISO/IEC 11889, BSI TR-03111 and New TCG proposal are not compatible

– Performance

- difficult to provide meaningful performance measurements for TPM

What does this paper not cover?

Rigorous security analysis

Thank you!