Trusted and Encrypted Keys

Trusted and Encrypted Keys are two new key types added to the existing kernel key ring service. Both of these new types are variable length symmetric keys, and in both cases all keys are created in the kernel, and user space sees, stores, and loads only encrypted blobs. Trusted Keys require the availability of a Trust Source for greater security, while Encrypted Keys can be used on any system. All user level blobs, are displayed and loaded in hex ASCII for convenience, and are integrity verified.

Trust Source

A trust source provides the source of security for Trusted Keys. This section lists currently supported trust sources, along with their security considerations. Whether or not a trust source is sufficiently safe depends on the strength and correctness of its implementation, as well as the threat environment for a specific use case. Since the kernel doesn't know what the environment is, and there is no metric of trust, it is dependent on the consumer of the Trusted Keys to determine if the trust source is sufficiently safe.

- Root of trust for storage
  1. TPM (Trusted Platform Module: hardware device)

     Rooted to Storage Root Key (SRK) which never leaves the TPM that provides crypto operation to establish root of trust for storage.
  2. TEE (Trusted Execution Environment: OP-TEE based on Arm TrustZone)

     Rooted to Hardware Unique Key (HUK) which is generally burnt in on-chip fuses and is accessible to TEE only.
- Execution isolation
  1. TPM

     Fixed set of operations running in isolated execution environment.
  2. TEE

     Customizable set of operations running in isolated execution environment verified via Secure/Trusted boot process.
- Optional binding to platform integrity state
  1. TPM

     Keys can be optionally sealed to specified PCR (integrity measurement) values, and only unsealed by the TPM, if PCRs and blob integrity verifications match. A loaded Trusted Key can be updated with new (future) PCR values, so keys are easily migrated to new PCR values, such as when the kernel and initramfs are updated. The same key can have many saved blobs under different PCR values, so multiple boots are easily supported.
  2. TEE

     Relies on Secure/Trusted boot process for platform integrity. It can be extended with TEE based measured boot process.
- Interfaces and APIs
  1. TPM

     TPMs have well-documented, standardized interfaces and APIs.
  2. TEE

     TEEs have well-documented, standardized client interface and APIs. For more details refer to `Documentation/staging/tee.rst`.
- Threat model

  The strength and appropriateness of a particular TPM or TEE for a given purpose must be assessed when using them to protect security-relevant data.

Key Generation

Trusted Keys

New keys are created from random numbers generated in the trust source. They are encrypted/decrypted using a child key in the storage key hierarchy. Encryption and decryption of the child key must be protected by a strong access control policy within the trust source.

- TPM (hardware device) based RNG

  Strength of random numbers may vary from one device manufacturer to another.
- TEE (OP-TEE based on Arm TrustZone) based RNG

  RNG is customizable as per platform needs. It can either be direct output from platform specific hardware RNG or a software based Fortuna CSPRNG which can be seeded via multiple entropy sources.

Encrypted Keys

Encrypted keys do not depend on a trust source, and are faster, as they use AES for encryption/decryption. New keys are created either from kernel-generated random numbers or user-provided decrypted data, and are encrypted/decrypted using a specified â€˜masterâ€™ key. The â€˜masterâ€™ key can either be a trusted-key or user-key type. The main disadvantage of encrypted keys is that if they are not rooted in a trusted key, they are only as secure as the user key encrypting them. The master user key should therefore be loaded in as secure a way as possible, preferably early in boot.

Usage

Trusted Keys usage: TPM

TPM 1.2: By default, trusted keys are sealed under the SRK, which has the default authorization value (20 bytes of 0s). This can be set at takeownership time with the TrouSerS utility: "tpm_takeownership -u -z".

TPM 2.0: The user must first create a storage key and make it persistent, so the key is available after reboot. This can be done using the following commands.

With the IBM TSS 2 stack:

```
#> tsscreateprimary -hi o -st
Handle 80000000
#> tssevictcontrol -hi o -ho 80000000 -hp 81000001
```

Or with the Intel TSS 2 stack:

```
#> tpm2_createprimary --hierarchy o -G rsa2048 -c key.ctxt
[...]
#> tpm2_evictcontrol -c key.ctxt 0x81000001
persistentHandle: 0x81000001
```

Usage:

```
keyctl add trusted name "new keylen [options]" ring
keyctl add trusted name "load hex_blob [pcrlock=pcrnum]" ring
keyctl update key "update [options]"
keyctl print keyid

options:
   keyhandle=    ascii hex value of sealing key
                   TPM 1.2: default 0x40000000 (SRK)
                   TPM 2.0: no default; must be passed every time
   keyauth=      ascii hex auth for sealing key default 0x00...i
                 (40 ascii zeros)
   blobauth=     ascii hex auth for sealed data default 0x00...
                 (40 ascii zeros)
   pcrinfo=      ascii hex of PCR_INFO or PCR_INFO_LONG (no default)
   pcrlock=      pcr number to be extended to "lock" blob
   migratable=   0|1 indicating permission to reseal to new PCR values,
                 default 1 (resealing allowed)
   hash=         hash algorithm name as a string. For TPM 1.x the only
                 allowed value is sha1. For TPM 2.x the allowed values
                 are sha1, sha256, sha384, sha512 and sm3-256.
   policydigest= digest for the authorization policy. must be calculated
                 with the same hash algorithm as specified by the 'hash='
                 option.
   policyhandle= handle to an authorization policy session that defines the
                 same policy and with the same hash algorithm as was used to
                 seal the key.
```

"keyctl print" returns an ascii hex copy of the sealed key, which is in standard TPM_STORED_DATA format. The key length for new keys are always in bytes. Trusted Keys can be 32 - 128 bytes (256 - 1024 bits), the upper limit is to fit within the 2048 bit SRK (RSA) keylength, with all necessary structure/padding.

Trusted Keys usage: TEE

Usage:

```
keyctl add trusted name "new keylen" ring
keyctl add trusted name "load hex_blob" ring
keyctl print keyid
```

"keyctl print" returns an ASCII hex copy of the sealed key, which is in format specific to TEE device implementation. The key length for new keys is always in bytes. Trusted Keys can be 32 - 128 bytes (256 - 1024 bits).

Encrypted Keys usage

The decrypted portion of encrypted keys can contain either a simple symmetric key or a more complex structure. The format of the more complex structure is application specific, which is identified by 'format'.

Usage:

```
keyctl add encrypted name "new [format] key-type:master-key-name keylen"
    ring
keyctl add encrypted name "new [format] key-type:master-key-name keylen
    decrypted-data" ring
keyctl add encrypted name "load hex_blob" ring
keyctl update keyid "update key-type:master-key-name"
```

Where:

```
format:= 'default | ecryptfs | enc32'
key-type:= 'trusted' | 'user'
```

Examples of trusted and encrypted key usage

Create and save a trusted key named "kmk" of length 32 bytes.

Note: When using a TPM 2.0 with a persistent key with handle 0x81000001, append 'keyhandle=0x81000001' to statements between quotes, such as "new 32 keyhandle=0x81000001".

```
$ keyctl add trusted kmk "new 32" @u
440502848

$ keyctl show
Session Keyring
       -3 --alswrv    500   500  keyring: _ses
 97833714 --alswrv    500    -1   \_ keyring: _uid.500
440502848 --alswrv    500   500       \_ trusted: kmk

$ keyctl print 440502848
0101000000000000000001005d01b7e3f4a6be5709930f3b70a743cbb42e0cc95e18e915
3f60da455bbf1144ad12e4f92b452f966929f6105fd29ca28e4d4d5a031d068478bacb0b
27351119f822911b0a11ba3d3498ba6a32e50dac7f32894dd890eb9ad578e4e292c83722
a52e56a097e6a68b3f56f7a52ece0cdccba1eb62cad7d817f6dc58898b3ac15f36026fec
d568bd4a706cb60bb37be6d8f1240661199d640b66fb0fe3b079f97f450b9ef9c22c6d5d
dd379f0facd1cd020281dfa3c70ba21a3fa6fc2471dc6d13ecf8298b946f65345faa5ef0
f1f8fff03ad0acb083725535636addb08d73dedb9832da198081e5deae84bfaf0409c22b
e4a8aea2b607ec96931e6f4d4fe563ba

$ keyctl pipe 440502848 > kmk.blob
```

Load a trusted key from the saved blob:

```
$ keyctl add trusted kmk "load `cat kmk.blob`" @u
268728824

$ keyctl print 268728824
0101000000000000000001005d01b7e3f4a6be5709930f3b70a743cbb42e0cc95e18e915
3f60da455bbf1144ad12e4f92b452f966929f6105fd29ca28e4d4d5a031d068478bacb0b
27351119f822911b0a11ba3d3498ba6a32e50dac7f32894dd890eb9ad578e4e292c83722
a52e56a097e6a68b3f56f7a52ece0cdccba1eb62cad7d817f6dc58898b3ac15f36026fec
d568bd4a706cb60bb37be6d8f1240661199d640b66fb0fe3b079f97f450b9ef9c22c6d5d
dd379f0facd1cd020281dfa3c70ba21a3fa6fc2471dc6d13ecf8298b946f65345faa5ef0
f1f8fff03ad0acb083725535636addb08d73dedb9832da198081e5deae84bfaf0409c22b
e4a8aea2b607ec96931e6f4d4fe563ba
```

Reseal (TPM specific) a trusted key under new PCR values:

```
$ keyctl update 268728824 "update pcrinfo=`cat pcr.blob`"
$ keyctl print 268728824
010100000000002c0002800093c35a09b70fff26e7a98ae786c641e678ec6ffb6b46d805
77c8a6377aed9d3219c6dfec4b23ffe3000001005d37d472ac8a44023fbb3d18583a4f73
d3a076c0858f6f1dcaa39ea0f119911ff03f5406df4f7f27f41da8d7194f45c9f4e00f2e
df449f266253aa3f52e55c53de147773e00f0f9aca86c64d94c95382265968c354c5eab4
9638c5ae99c89de1e0997242edfb0b501744e11ff9762dfd951cffd93227cc513384e7e6
e782c29435c7ec2edafaa2f4c1fe6e7a781b59549ff5296371b42133777dcc5b8b971610
94bc67ede19e43ddb9dc2baacad374a36feaf0314d700af0a65c164b7082401740e489c9
7ef6a24defe4846104209bf0c3eced7fa1a672ed5b125fc9d8cd88b476a658a4434644ef
df8ae9a178e9f83ba9f08d10fa47e4226b98b0702f06b3b8
```

The initial consumer of trusted keys is EVM, which at boot time needs a high quality symmetric key for HMAC protection of file metadata. The use of a trusted key provides strong guarantees that the EVM key has not been compromised by a user level problem, and when sealed to a platform integrity state, protects against boot and offline attacks. Create and save an encrypted key "evm" using the above trusted key "kmk":

option 1: omitting 'format':

```
$ keyctl add encrypted evm "new trusted:kmk 32" @u
159771175
```

option 2: explicitly defining 'format' as 'default':

```
$ keyctl add encrypted evm "new default trusted:kmk 32" @u
159771175

$ keyctl print 159771175
default trusted:kmk 32 2375725ad57798846a9bbd240de8906f006e66c03af53b1b3
82dbbc55be2a44616e4959430436dc4f2a7a9659aa60bb4652aeb2120f149ed197c564e0
24717c64 5972dcb82ab2dde83376d82b2e3c09ffc

$ keyctl pipe 159771175 > evm.blob
```

Load an encrypted key "evm" from saved blob:

```
$ keyctl add encrypted evm "load `cat evm.blob`" @u
831684262

$ keyctl print 831684262
default trusted:kmk 32 2375725ad57798846a9bbd240de8906f006e66c03af53b1b3
82dbbc55be2a44616e4959430436dc4f2a7a9659aa60bb4652aeb2120f149ed197c564e0
24717c64 5972dcb82ab2dde83376d82b2e3c09ffc
```

Instantiate an encrypted key "evm" using user-provided decrypted data:

```
$ keyctl add encrypted evm "new default user:kmk 32 `cat evm_decrypted_data.blob`" @u
794890253

$ keyctl print 794890253
default user:kmk 32 2375725ad57798846a9bbd240de8906f006e66c03af53b1b382d
bbc55be2a44616e4959430436dc4f2a7a9659aa60bb4652aeb2120f149ed197c564e0247
17c64 5972dcb82ab2dde83376d82b2e3c09ffc
```

Other uses for trusted and encrypted keys, such as for disk and file encryption are anticipated. In particular the new format 'ecryptfs' has been defined in order to use encrypted keys to mount an eCryptfs filesystem. More details about the usage can be found in the file `Documentation/security/keys/ecryptfs.rst`.

Another new format 'enc32' has been defined in order to support encrypted keys with payload size of 32 bytes. This will initially be used for nvdimm security but may expand to other usages that require 32 bytes payload.

TPM 2.0 ASN.1 Key Format

The TPM 2.0 ASN.1 key format is designed to be easily recognisable, even in binary form (fixing a problem we had with the TPM 1.2 ASN.1 format) and to be extensible for additions like importable keys and policy:

```
TPMKey ::= SEQUENCE {
    type            OBJECT IDENTIFIER
    emptyAuth       [0] EXPLICIT BOOLEAN OPTIONAL
    parent          INTEGER
    pubkey          OCTET STRING
    privkey         OCTET STRING
}
```

type is what distinguishes the key even in binary form since the OID is provided by the TCG to be unique and thus forms a recognizable binary pattern at offset 3 in the key. The OIDs currently made available are:

```
2.23.133.10.1.3 TPM Loadable key.  This is an asymmetric key (Usually
                RSA2048 or Elliptic Curve) which can be imported by a
                TPM2_Load() operation.

2.23.133.10.1.4 TPM Importable Key.  This is an asymmetric key (Usually
                RSA2048 or Elliptic Curve) which can be imported by a
                TPM2_Import() operation.

2.23.133.10.1.5 TPM Sealed Data.  This is a set of data (up to 128
                bytes) which is sealed by the TPM.  It usually
                represents a symmetric key and must be unsealed before
                use.
```

The trusted key code only uses the TPM Sealed Data OID.

emptyAuth is true if the key has well known authorization "". If it is false or not present, the key requires an explicit authorization phrase. This is used by most user space consumers to decide whether to prompt for a password.

parent represents the parent key handle, either in the 0x81 MSO space, like 0x81000001 for the RSA primary storage key. Userspace programmes also support specifying the primary handle in the 0x40 MSO space. If this happens the Elliptic Curve variant of the primary key using the TCG defined template will be generated on the fly into a volatile object and used as the parent. The current kernel code only supports the 0x81 MSO form.

pubkey is the binary representation of TPM2B_PRIVATE excluding the initial TPM2B header, which can be reconstructed from the ASN.1 octet string length.

privkey is the binary representation of TPM2B_PUBLIC excluding the initial TPM2B header which can be reconstructed from the ASN.1 octed string length.