Hiding Attestation with Linux Keyring in Confidential Virtual Machines

Mikko Ylinen, Intel, mythi

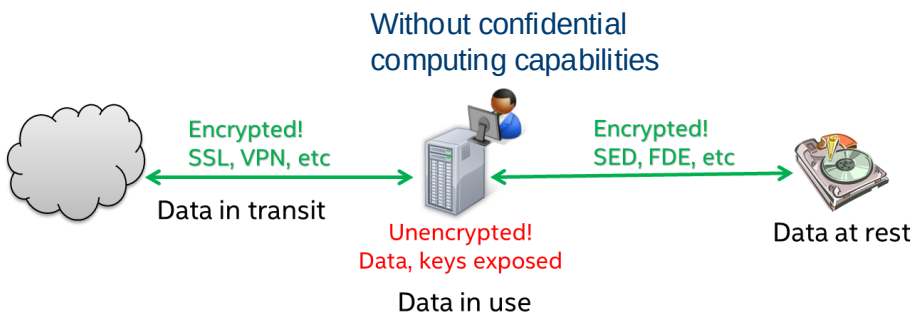
Storyline

- "Apps need an API to build TEE evidence for remote attestation to get secrets"
    - me: "Linux Keyring has the API with the necessary functionality, how about that?"
- "Sounds OK but it really depends on the use case!"

Agenda

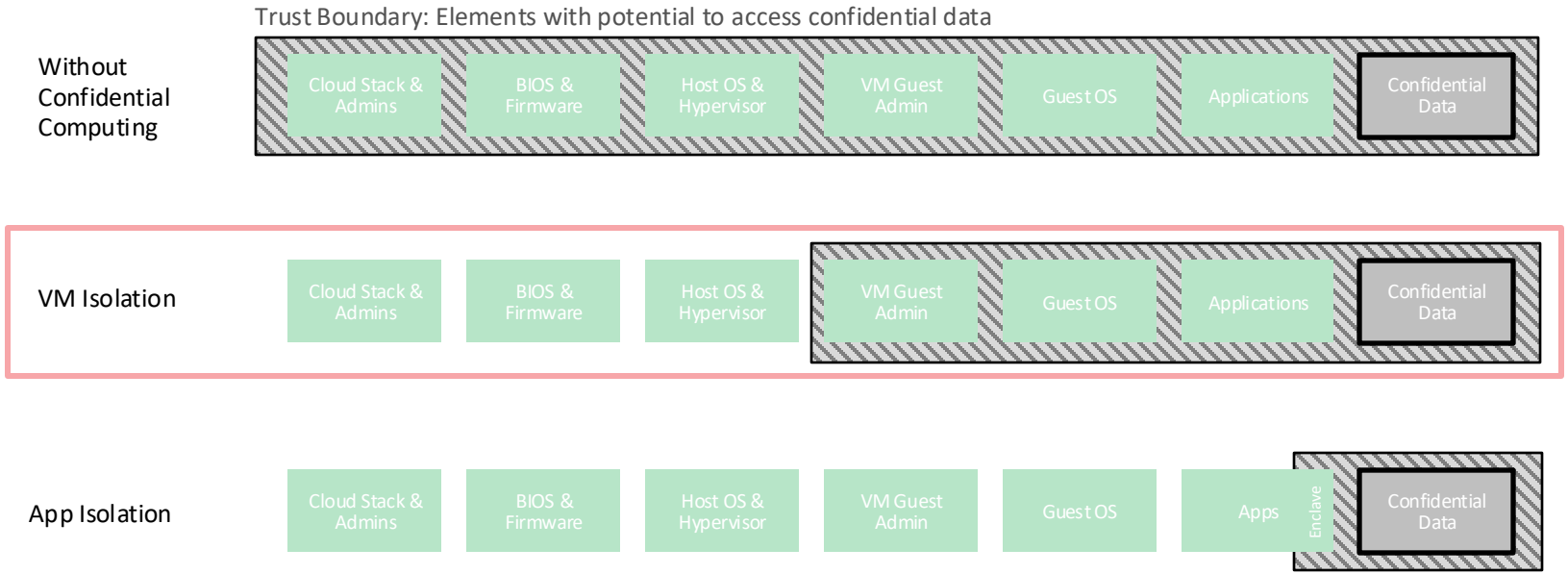
- Confidential Computing 1-0-1
  - Terminology, attestation *roles* and *topologies*
- Linux key management (Keyring) overview
- Use cases
  - `cryptsetup` LUKS passphrase retrieval with remote attestation
  - JWT Attestation token generation
- Summary and call to action

Terminology

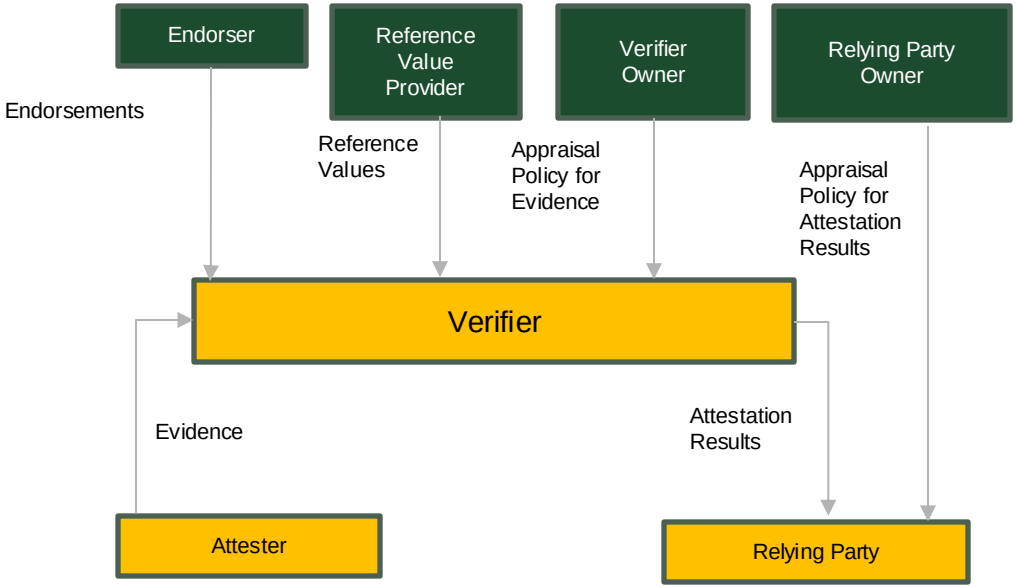


- Confidential Computing
  - Adds data security when data is in use
  - Protects privacy sensitive data in untrusted environments (e.g., public clouds)
- Trusted Execution Environment (TEE)
  - Hardware runtime environment that prevents unauthorized entities to tamper data confidentiality, integrity and code integrity
  - May include features like attestability to provide evidence of its origin/state
- Trusted Computing Base (TCB)
  - Components (HW, FW, SW) of a system that are critical to its security. A bug inside the TCB may break the system security
  - Design for small TCB

Trust Boundary of Confidential Computing (CC)

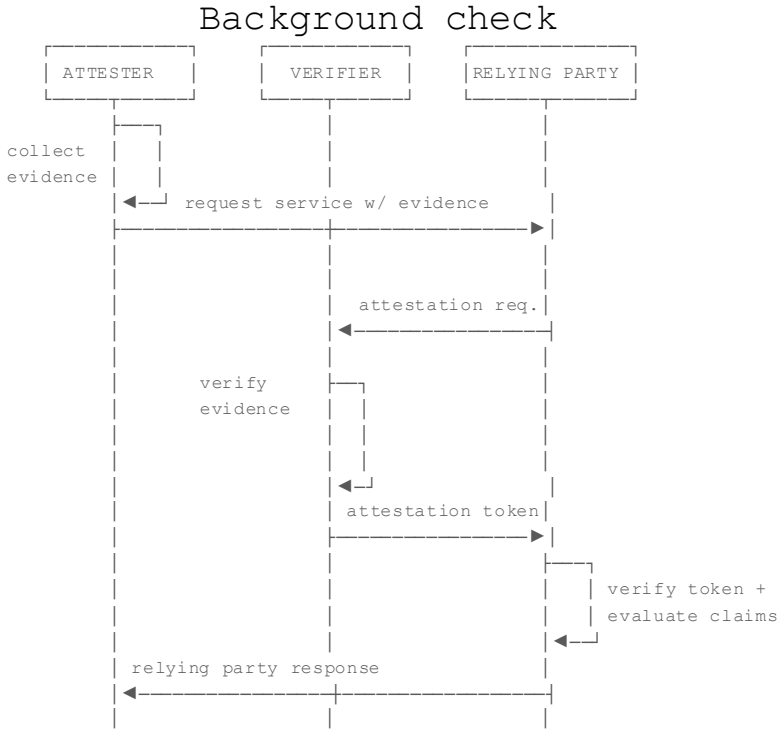


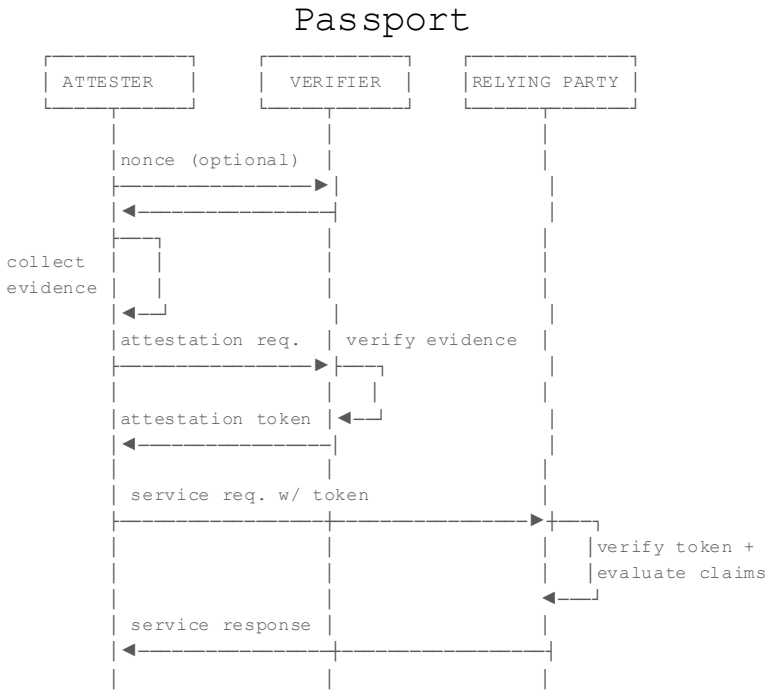
Attestation Roles



Remote ATtestation procedureS (RATS) Architecture https://datatracker.ietf.org/doc/rfc9334/

Attestation Topologies





Remote ATtestation procedureS (RATS) Architecture https://datatracker.ietf.org/doc/rfc9334/

Linux Key Management - `keyrings(7)`

- In-kernel store/cache for security data (e.g., keys, tokens) for
  - kernel components
  - *applications*
- **Key**: ID, type, description, payload, access control, expr., ref. cnt
- **Key types**: keyring, user, logon, big_key, etc...
- **Keyring types**:
  - Process (session, process, thread)
  - User (user session, session)
  - Persistent/special
- **System calls** to manage keys/keyrings
  - `add_key(2), request_key(2), keyctl(2)`

```
# keyctl show
Session Keyring
604308610 --alswrv      0     0  keyring: _ses
124009475 --alswrv      0 65534   \_ keyring: _uid.0
641130585 --alswrv      0     0   \_ user: coco:token
691860150 --als--v      0     0   \_ asymmetric: signer
```

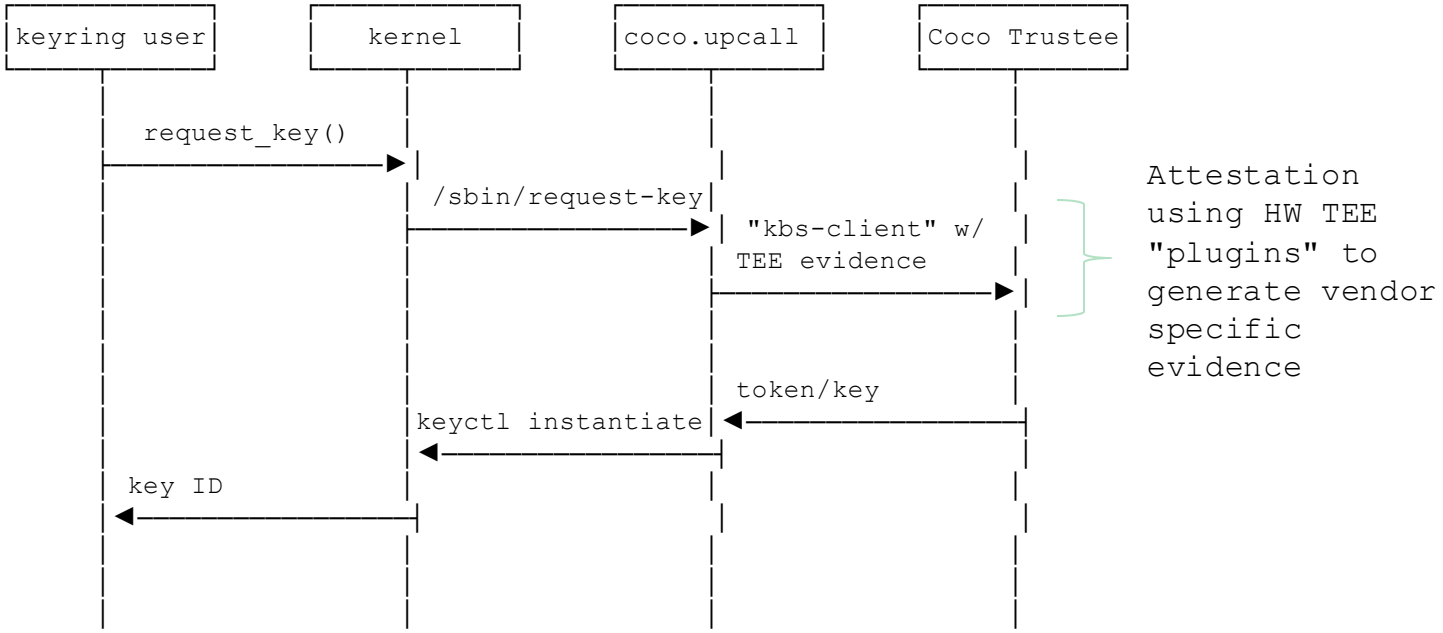
Key Request Service – `request_key(2)`

- "request a key from the kernel's key management facility":

```
key_serial_t request_key(const char *type, const char *description,
                         const char *_Nullable callout_info,
                         key_serial_t dest_keyring);
```

- The kernel can further request user-space to instantiate the key if it's not found and `callout_info` is not NULL.
  - Execute user-space "`/sbin/request-key`" provider to get the key instantiated
  - Can further call another program to help with it (based on `request-key.conf(5)`), e.g., connect to a remote HSM/KMS

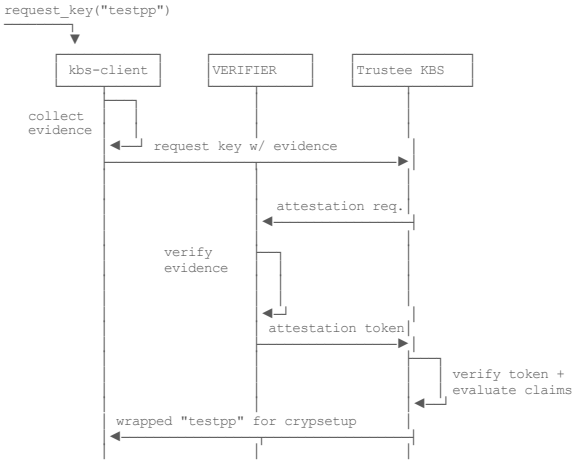
request_key(2) With Confidential Containers' Trustee



Use-case: LUKS Passphrase Retrieval with Attestation 1/2

- Confidential VMs encrypt storage volumes to protect from host/VMM managing the disk (Virtio-Block).
- `cryptsetup` can read keyslot passphrases from the Keyring BUT does not use `request_key()` with `callout_info` set [1].

```
# keyctl show
Session Keyring
604308610 --alswrv      0     0  keyring: _ses
124009475 --alswrv      0 65534   \_ keyring: _uid.0
# keyctl request2 user testpp from-coco-trustee
686166034
# keyctl show | grep testpp
```



Background check w/ Keyring

[1] https://gitlab.com/cryptsetup/cryptsetup/-/issues/897

Use-case: LUKS Passphrase Retrieval with Attestation 2/2

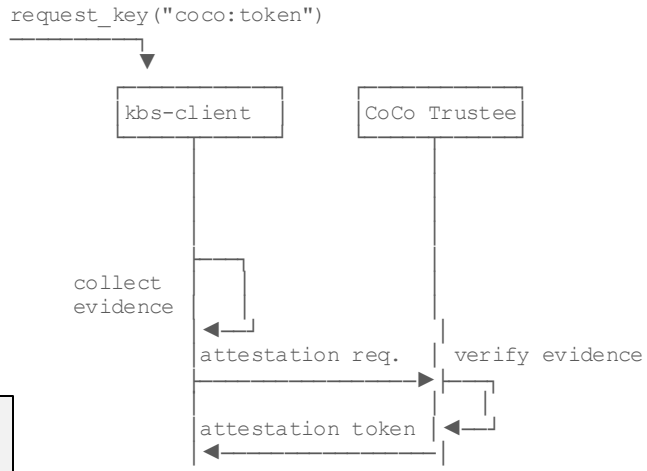
```
--- a/lib/utils_keyring.c
+++ b/lib/utils_keyring.c
@@ -216,7 +216,7 @@ key_serial_t keyring_request_key_id(key_type_t key_type,
        key_serial_t kid;

        do {
-               kid = request_key(key_type_name(key_type), key_description, NULL, 0);
+               kid = request_key(key_type_name(key_type), key_description, "cryptsetup", 0);
        } while (kid < 0 && errno == EINTR);

        return kid;
```

```
# keyctl show | grep testpp
# ./cryptsetup open /dev/loop0 myCrypt
# keyctl show | grep testpp
792607524 --alswrv      0     0    \_ user: testpp
# mount /dev/mapper/myCrypt /mnt/
# ls /mnt/
hello-world.txt  lost+found
```


RFC: https://gitlab.com/cryptsetup/cryptsetup/-/issues/897

Use-case: JWT Attestation Token Generation

- Get a fresh attestation token without applications needing to know the details (e.g., verifier configuration).
- TEE TCB Status re-verified for every request.
- `request-key` handler can add key metadata such as key expiration based on token lifetime.



Passport mode w/ Keyring

```
# keyctl show
Session Keyring
604308610 --alswrv      0     0  keyring: _ses
124009475 --alswrv      0 65534   \_ keyring: _uid.0
# keyctl request2 user coco:token jwt
966451172
# keyctl print 966451172  | jq -R 'split(".") | .[1] | @base64d | fromjson | .iss'
"CoCo-Attestation-Service"
# keyctl print 966451172  | jq -R 'split(".") | .[1] | @base64d | fromjson | .tee'
"tdx"
```

Summary + Call to Action

- `request_key(2)` can hide remote attestation details and helps to keep applications simple (implementation, verifier/relying party configuration).
- Simple use cases provided to demonstrate the use.
  - What's yours?
  - Backup slide: asymmetric keys for code signing (secure supply chain with TEEs)?
- Questions, comments?
- Material:
  `https://gist.github.com/mythi/20848caadf4628695499332d3c81779c`


Thank you!

Backup: asymmetric keys for code signing

```
# keyctl request2 asymmetric signer codesign @s
956541655
# echo coco-ftw | openssl sha256 -binary > coco.sha256
# keyctl pkey_sign %asymmetric:signer 0 coco.sha256 enc=pkcs1 hash=sha256 >coco.sig
# echo coco-ftw | openssl sha256 -verify ../pubkey.pem -signature coco.sig
Verified OK
```


Ref: https://fosdem.org/2024/schedule/event/fosdem-2024-3371-what-is-linux-kernel-keystore-and-why-you-should-use-it-in-your-next-application/