Mar/15/2022

CCC Attestation Meeting:

RA-TLS and Gramine

by Dmitrii Kuvaiskii

Legal Disclaimers

Intel provides these materials as-is, with no express or implied warranties.
All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice.
Intel processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at http://intel.com.
Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.
Intel and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

Note: This presentation concentrates on **ECDSA/DCAP** SGX attestation

(there is also **EPID** SGX attestation)

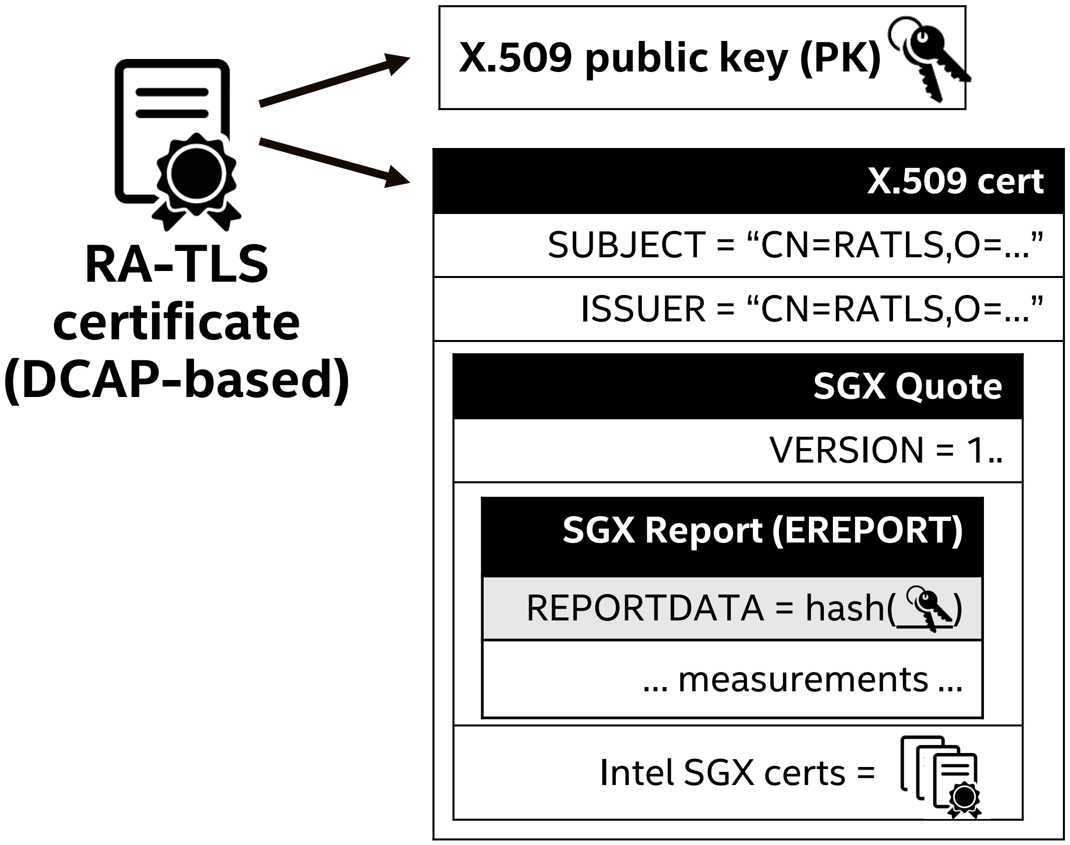
RA-TLS motivation

- Remote attestation allows to verify attester by remote party
  - Almost always followed by secure channel establishment
  - Moreover, RA must be coupled with secure channel establishment to prevent MITM attacks
  - RA-TLS **combines** SGX attestation and TLS secure-channel protocol

- TLS protocol and its implementations **should not be modified**
  - TLS uses X.509 certs which can carry arbitrary-data OID extensions
  - TLS library (OpenSSL, WolfSSL, mbedTLS) is unchanged
  - TLS library must provide a hook to verify custom X.509 OID extensions

RA-TLS overview

- RA-TLS is an extension:
  - to the TLS handshake protocol (verification hook), and
  - to the X.509 certificate fields (new non-standard OID that embeds SGX quote)
    - This was the easiest/quickest path forward
    - In the future, this way may be standardized (or RA-TLS uses some other standard way)
- First published in January 2018 on Arxiv
  - “Integrating Remote Attestation with Transport Layer Security” by Thomas Knauth, etc.
  - https://arxiv.org/abs/1801.05863
- Reference implementations with OpenSSL, WolfSSL, mbedTLS
  - No modifications to the TLS protocol/library, but 2 new lines of code in the TLS application
  - https://github.com/cloud-security-research/sgx-ra-tls (stale; latest is in Gramine repo)



RA-TLS certificate (DCAP-based)

X.509 public key (PK) 🔑

X.509 cert

SUBJECT = “CN=RATLS,O=…”

ISSUER = “CN=RATLS,O=…”

SGX Quote

VERSION = 1..

SGX Report (EREPORT)

REPORTDATA = hash(🔑)

… measurements …

Intel SGX certs = 📜

Self-signed certificate

Evidence

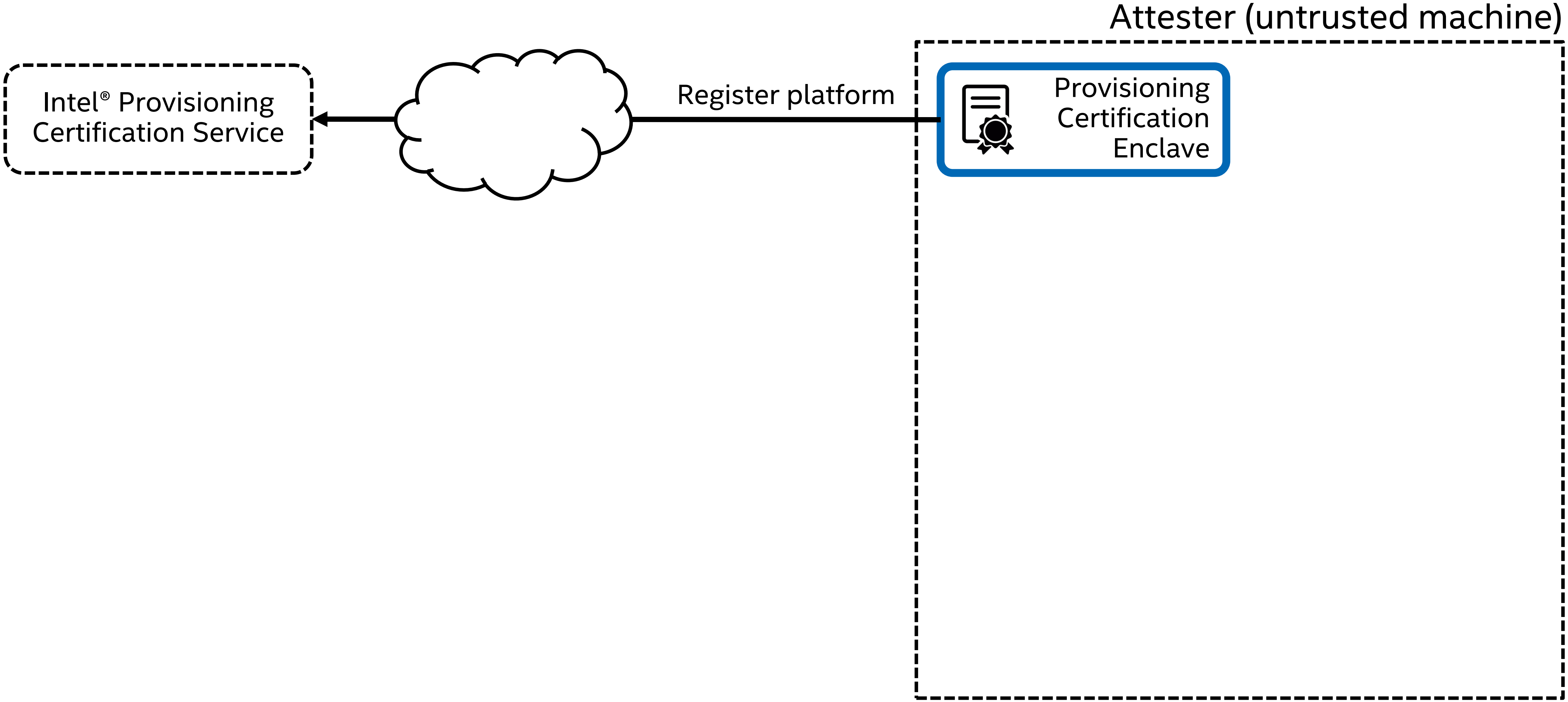
Compare PK against this hash

Must verify against expected

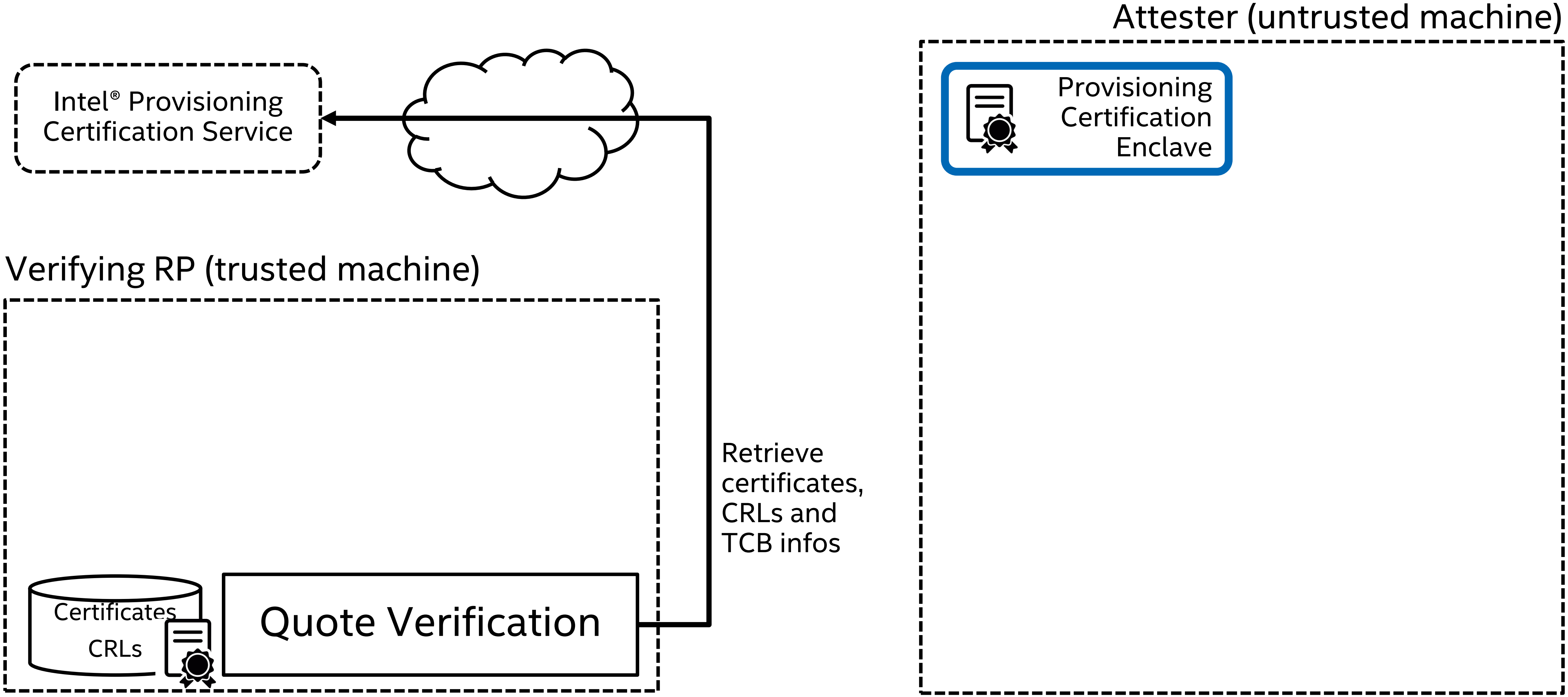
Must verify against Intel PCS provided attestation certificates

(comparing evidence against policy)

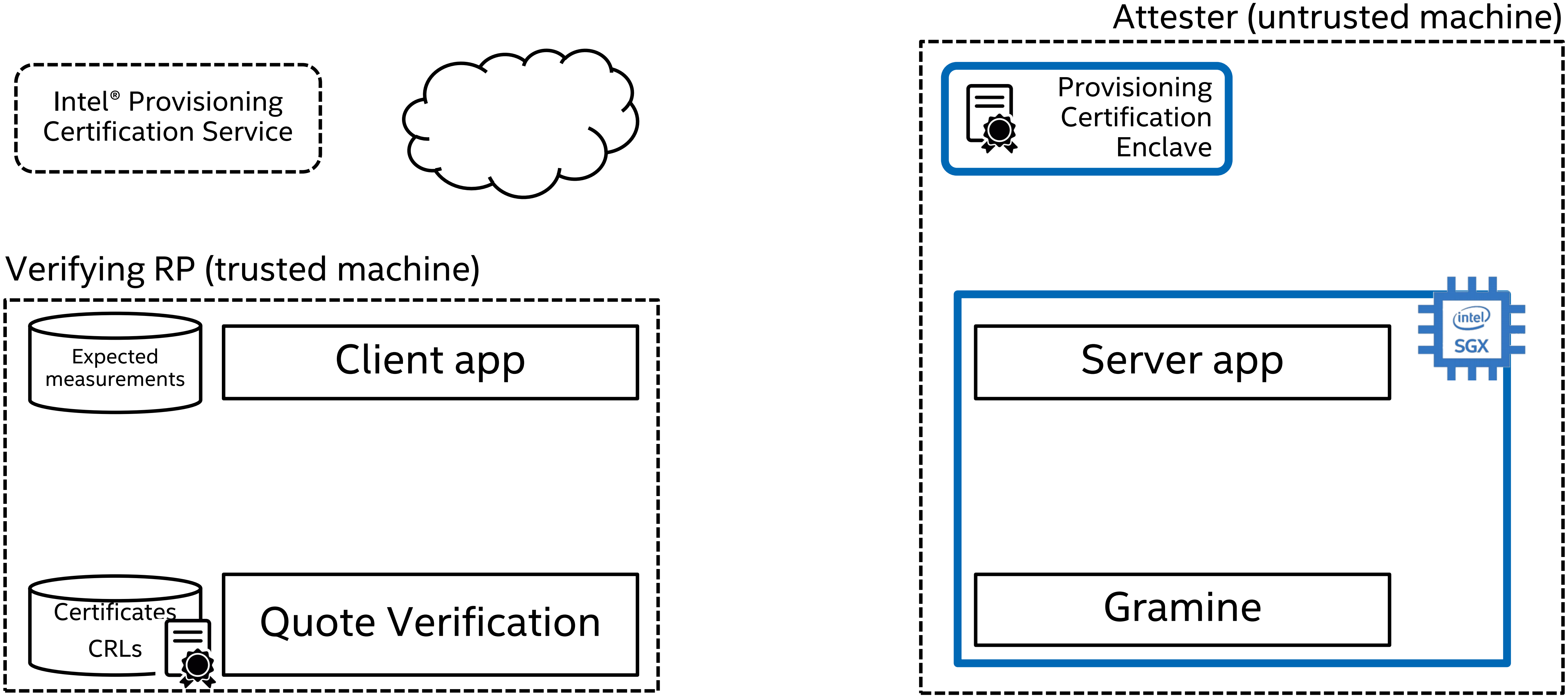
SGX attestation flow


Attester (untrusted machine)
Intel® Provisioning Certification Service
Register platform
Provisioning Certification Enclave

SGX attestation flow

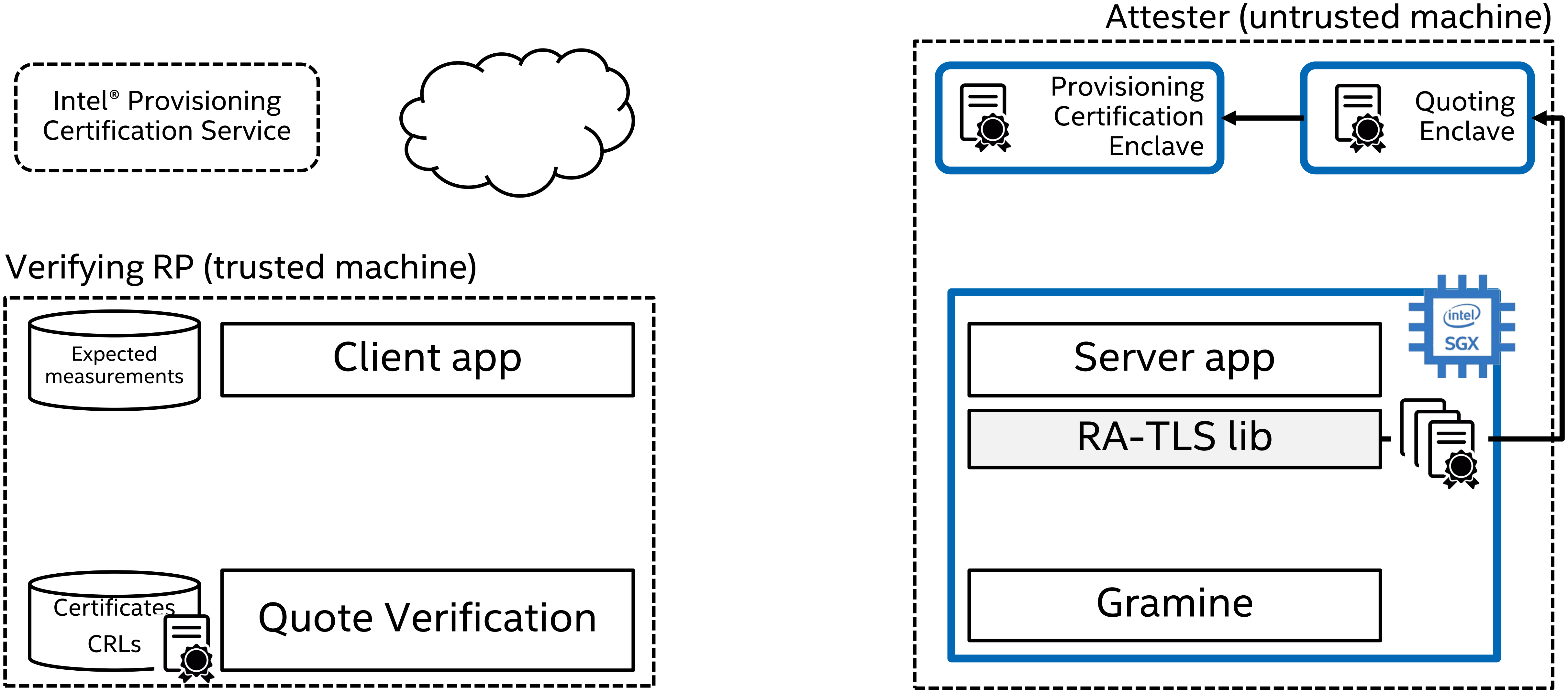

Attester (untrusted machine)
Intel® Provisioning Certification Service
Provisioning Certification Enclave
Verifying RP (trusted machine)
Retrieve certificates, CRLs and TCB infos
Certificates
CRLs
Quote Verification

SGX attestation flow

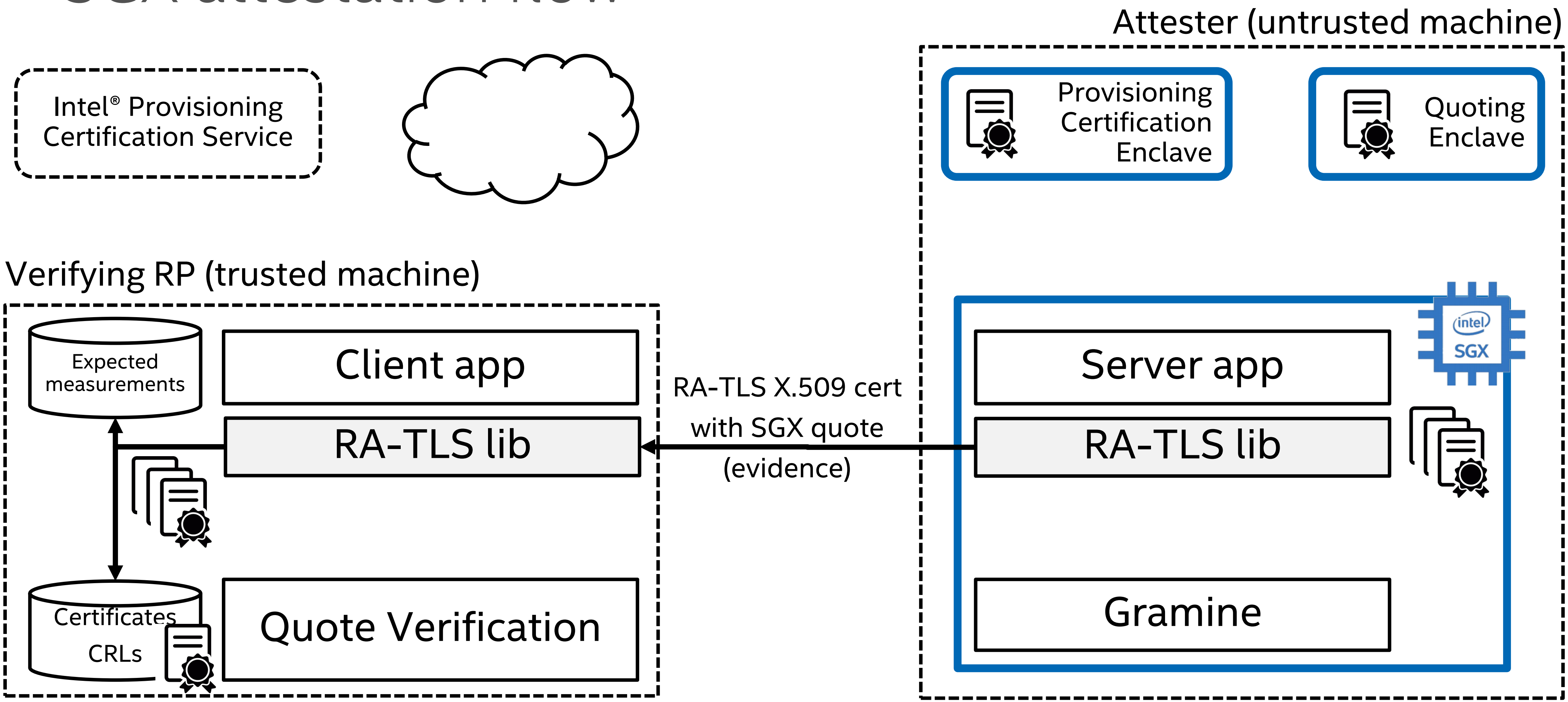


Intel® Provisioning Certification Service
Attester (untrusted machine)
Provisioning Certification Enclave
Verifying RP (trusted machine)
Expected measurements
Client app
intel SGX
Server app
Certificates CRLs
Quote Verification
Gramine

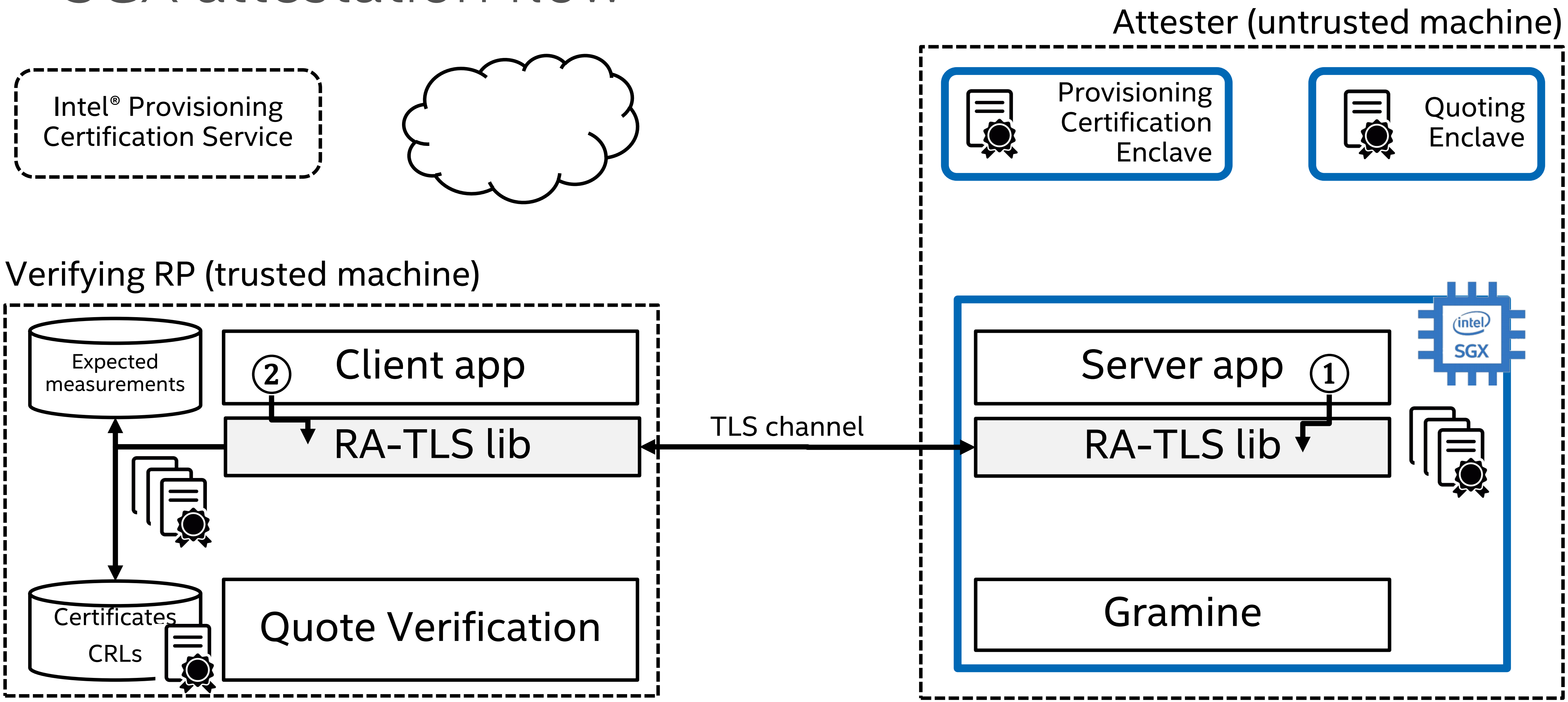
SGX attestation flow


Attester (untrusted machine)
Provisioning Certification Enclave
Quoting Enclave
Intel® Provisioning Certification Service
Verifying RP (trusted machine)
Expected measurements
Client app
Certificates
CRLs
Quote Verification
intel
SGX
Server app
RA-TLS lib
Gramine

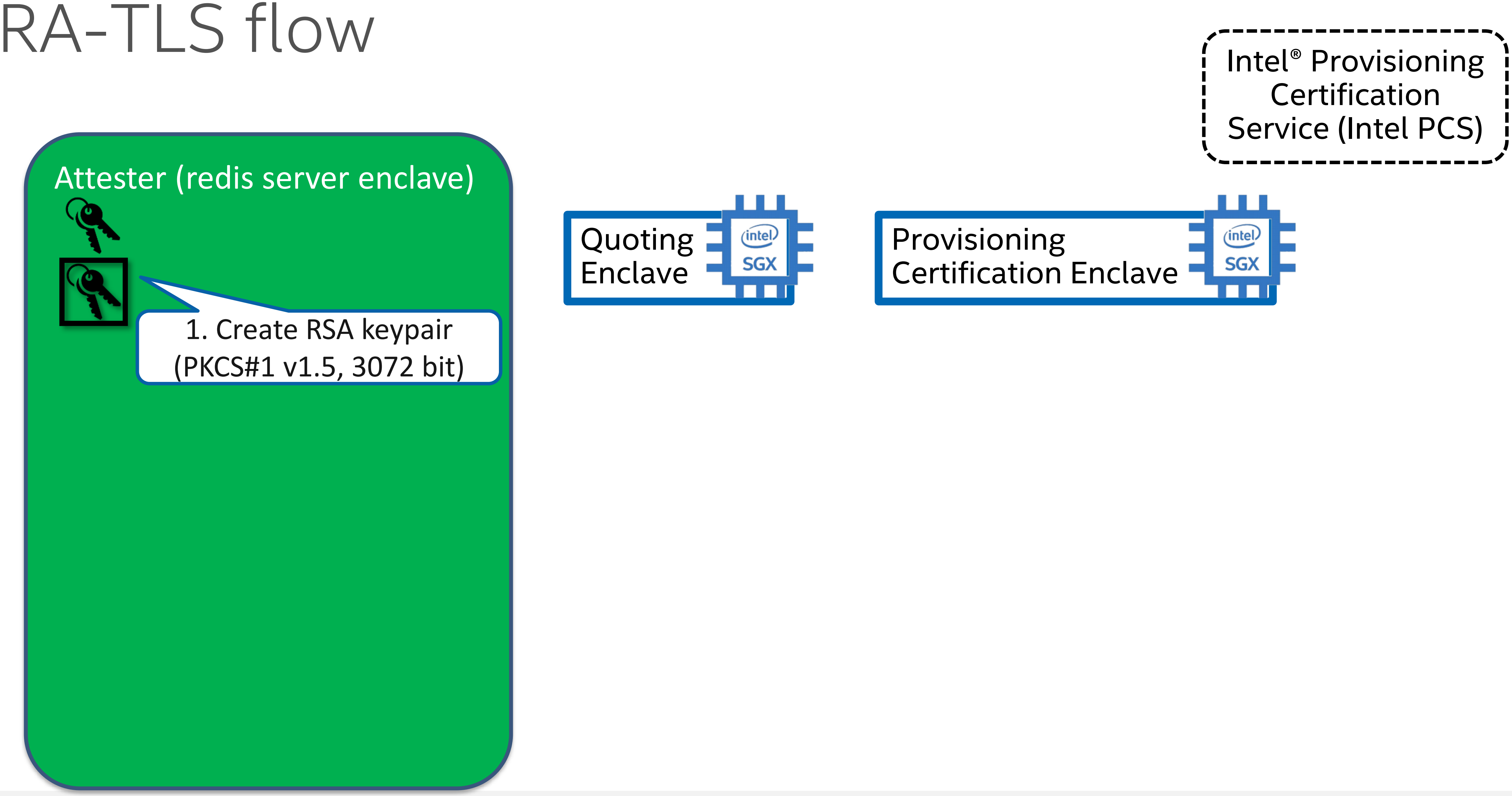
SGX attestation flow


Attester (untrusted machine)
Intel® Provisioning Certification Service
Provisioning Certification Enclave
Quoting Enclave
Verifying RP (trusted machine)
Expected measurements
Client app
RA-TLS lib
Certificates
CRLs
Quote Verification
RA-TLS X.509 cert with SGX quote (evidence)
intel SGX
Server app
RA-TLS lib
Gramine

SGX attestation flow

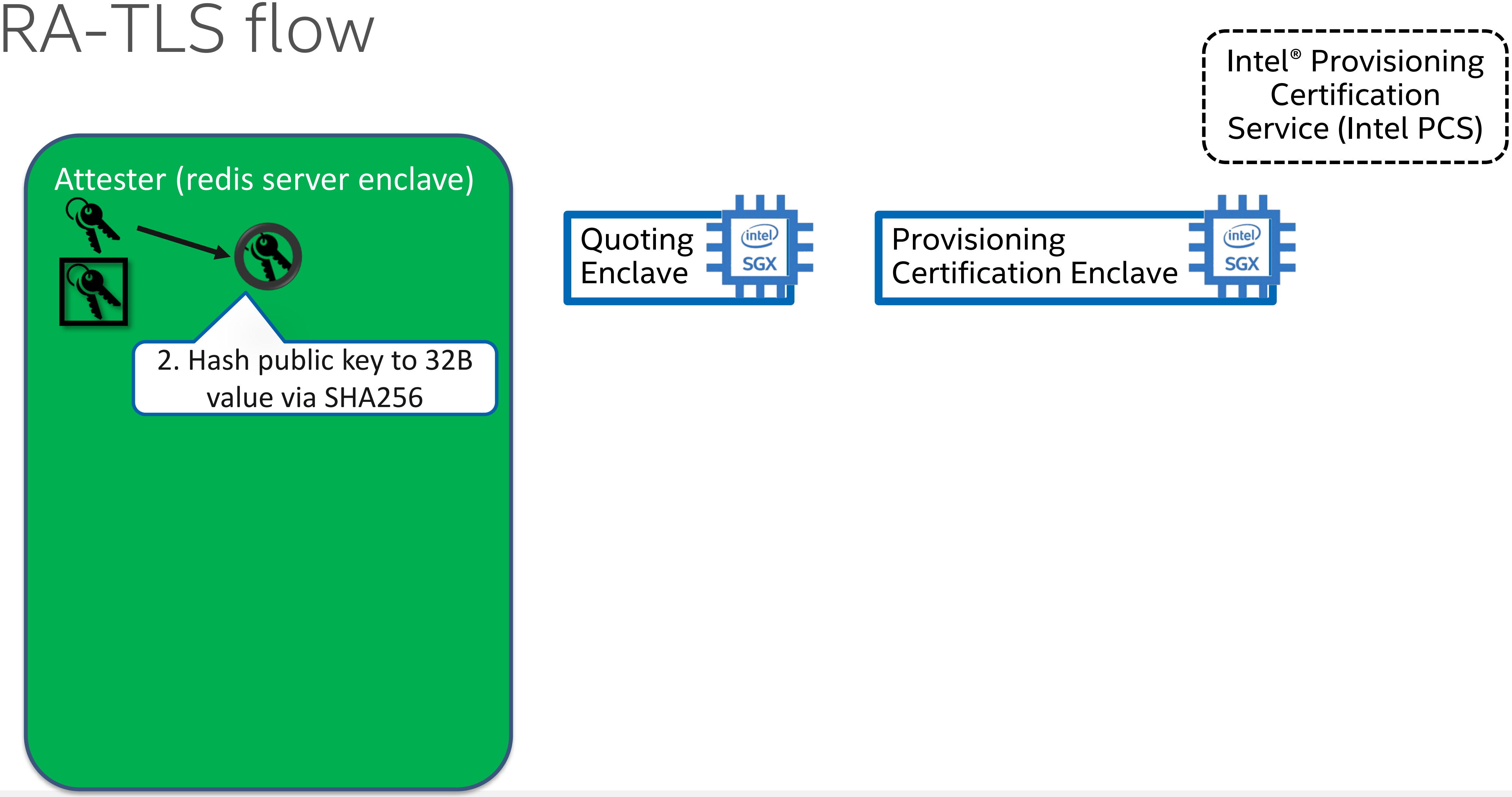


RA-TLS flow



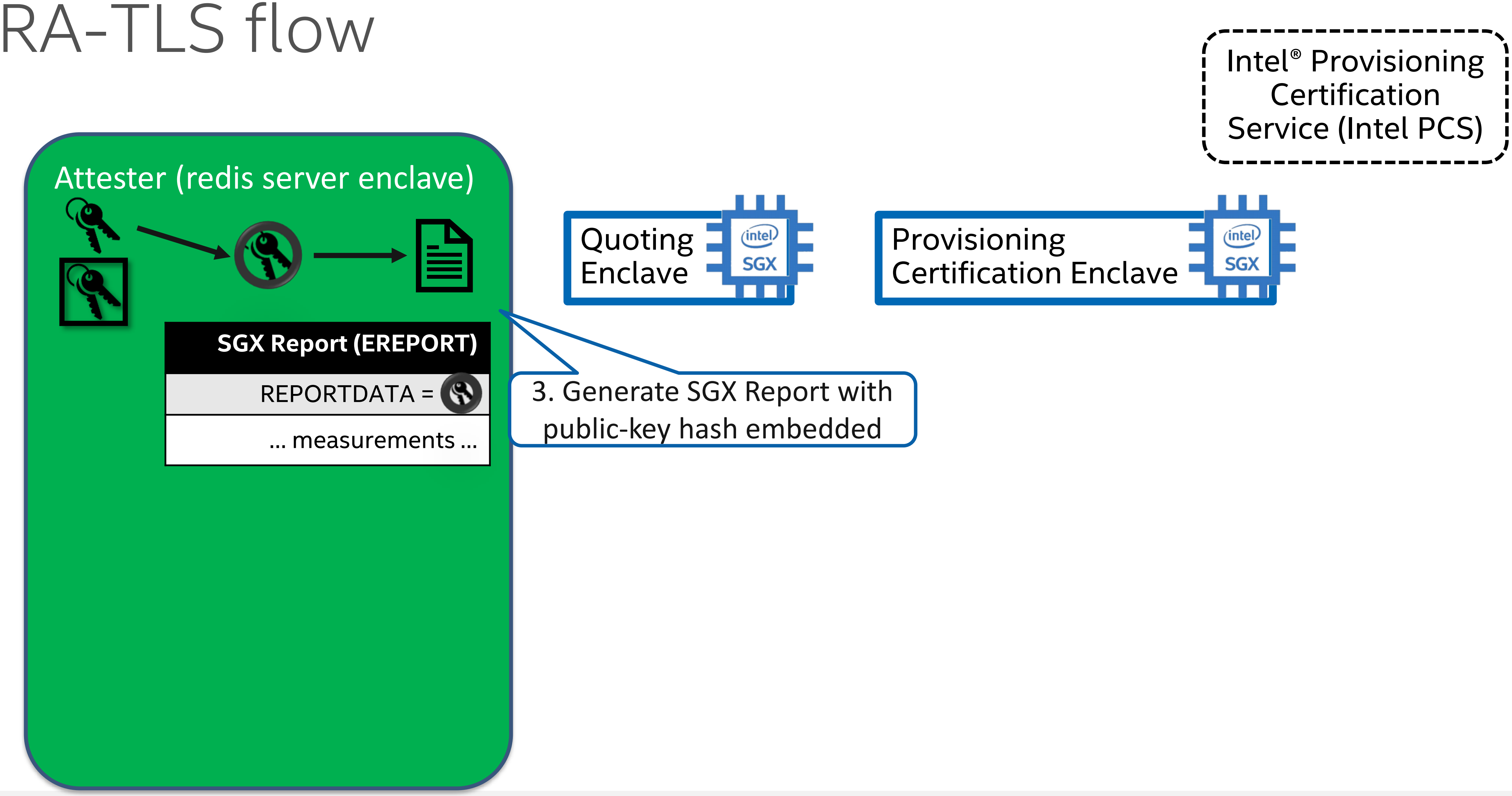
Intel® Provisioning Certification Service (Intel PCS)
Attester (redis server enclave)
1. Create RSA keypair (PKCS#1 v1.5, 3072 bit)
Quoting Enclave
intel SGX
Provisioning Certification Enclave
intel SGX

RA-TLS flow


Intel® Provisioning Certification Service (Intel PCS)
Attester (redis server enclave)
Quoting Enclave
intel
SGX
Provisioning Certification Enclave
intel
SGX

2. Hash public key to 32B value via SHA256

RA-TLS flow



Intel® Provisioning Certification Service (Intel PCS)

Attester (redis server enclave)

SGX Report (EREPORT)

REPORTDATA =

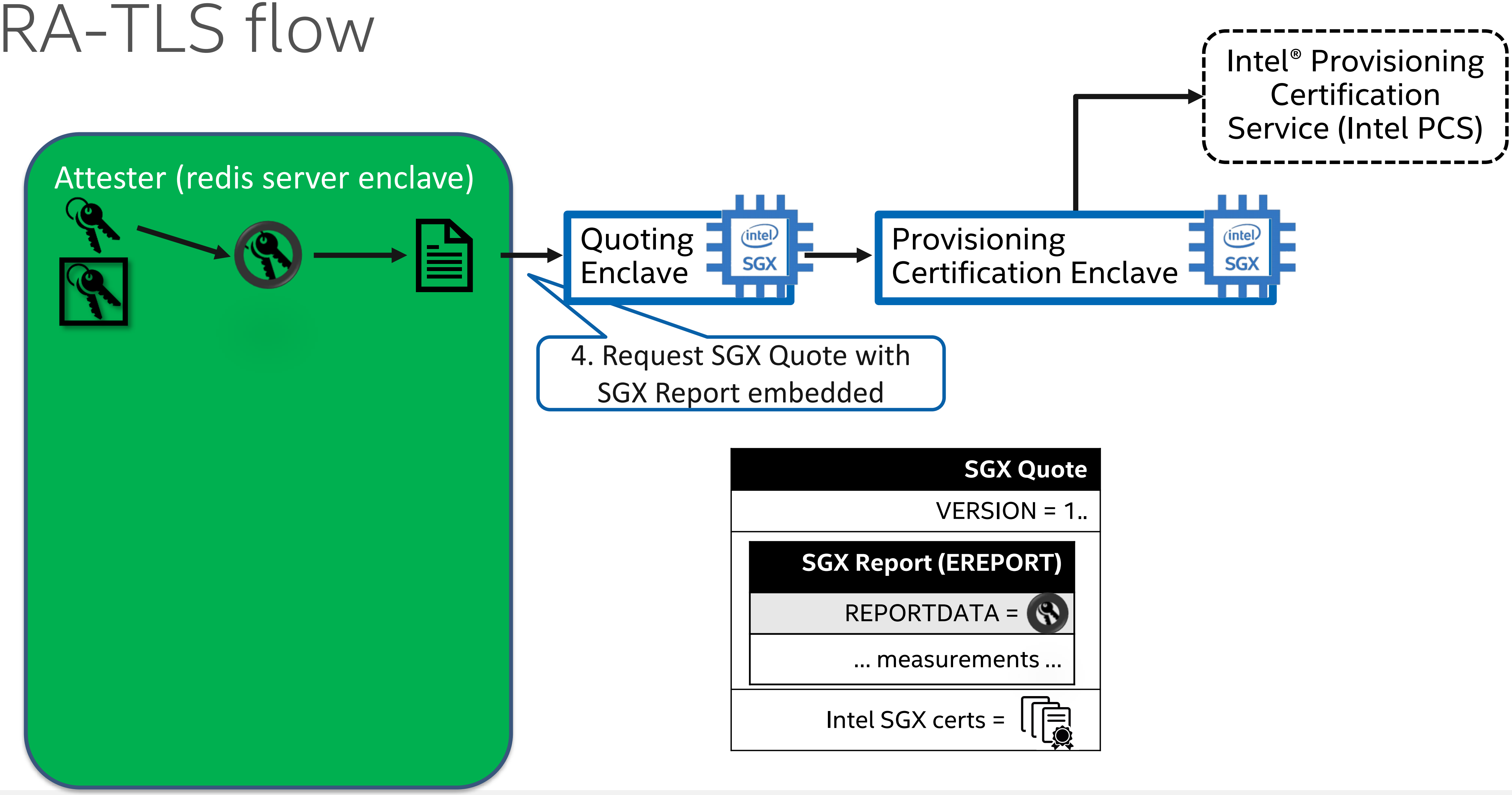
... measurements ...

Quoting Enclave

Provisioning Certification Enclave

3. Generate SGX Report with public-key hash embedded

RA-TLS flow


Attester (redis server enclave)
Quoting Enclave
intel SGX
Provisioning Certification Enclave
intel SGX
Intel® Provisioning Certification Service (Intel PCS)

4. Request SGX Quote with SGX Report embedded

SGX Quote

VERSION = 1..

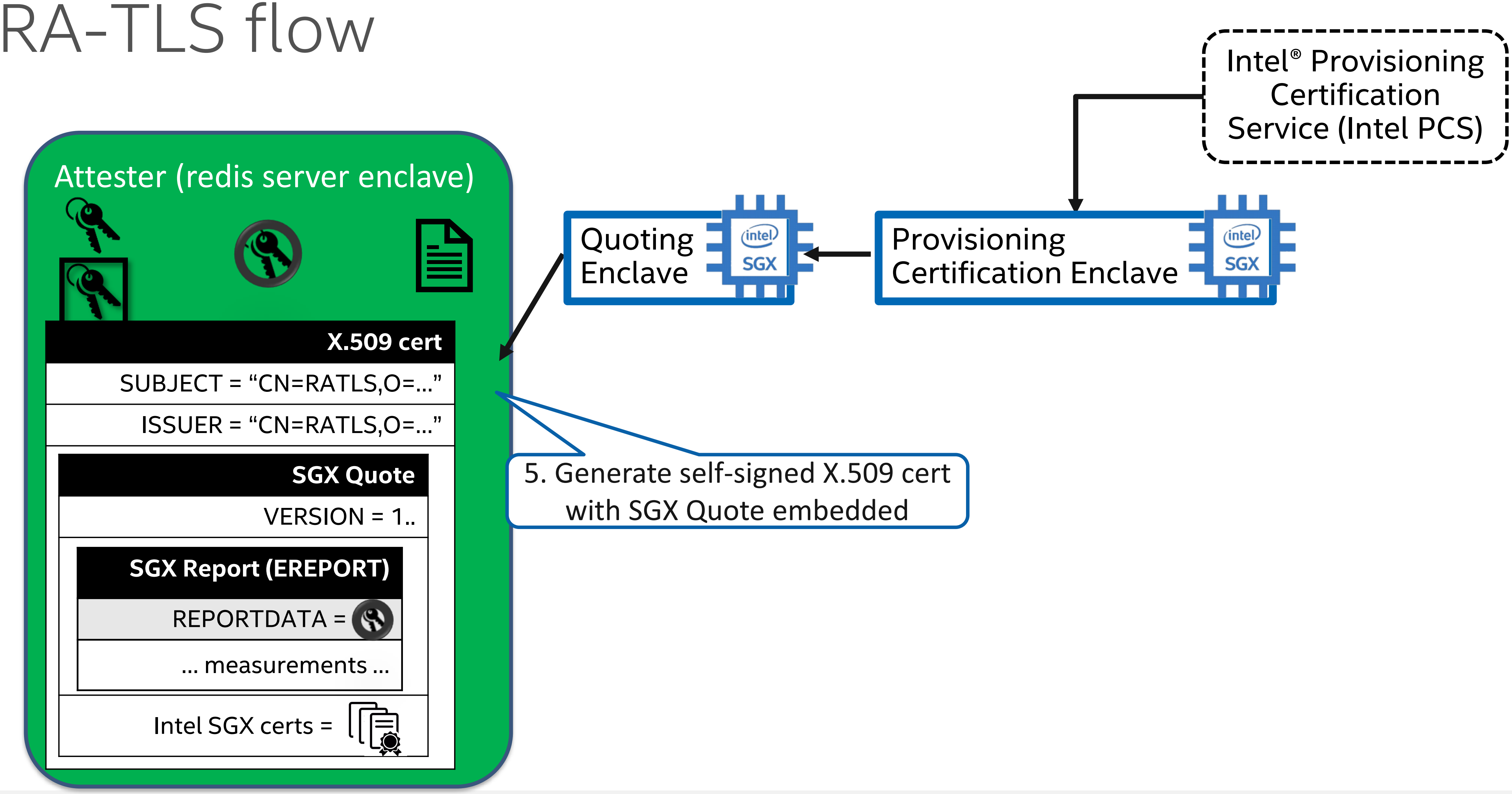
SGX Report (EREPORT)

REPORTDATA =

... measurements ...

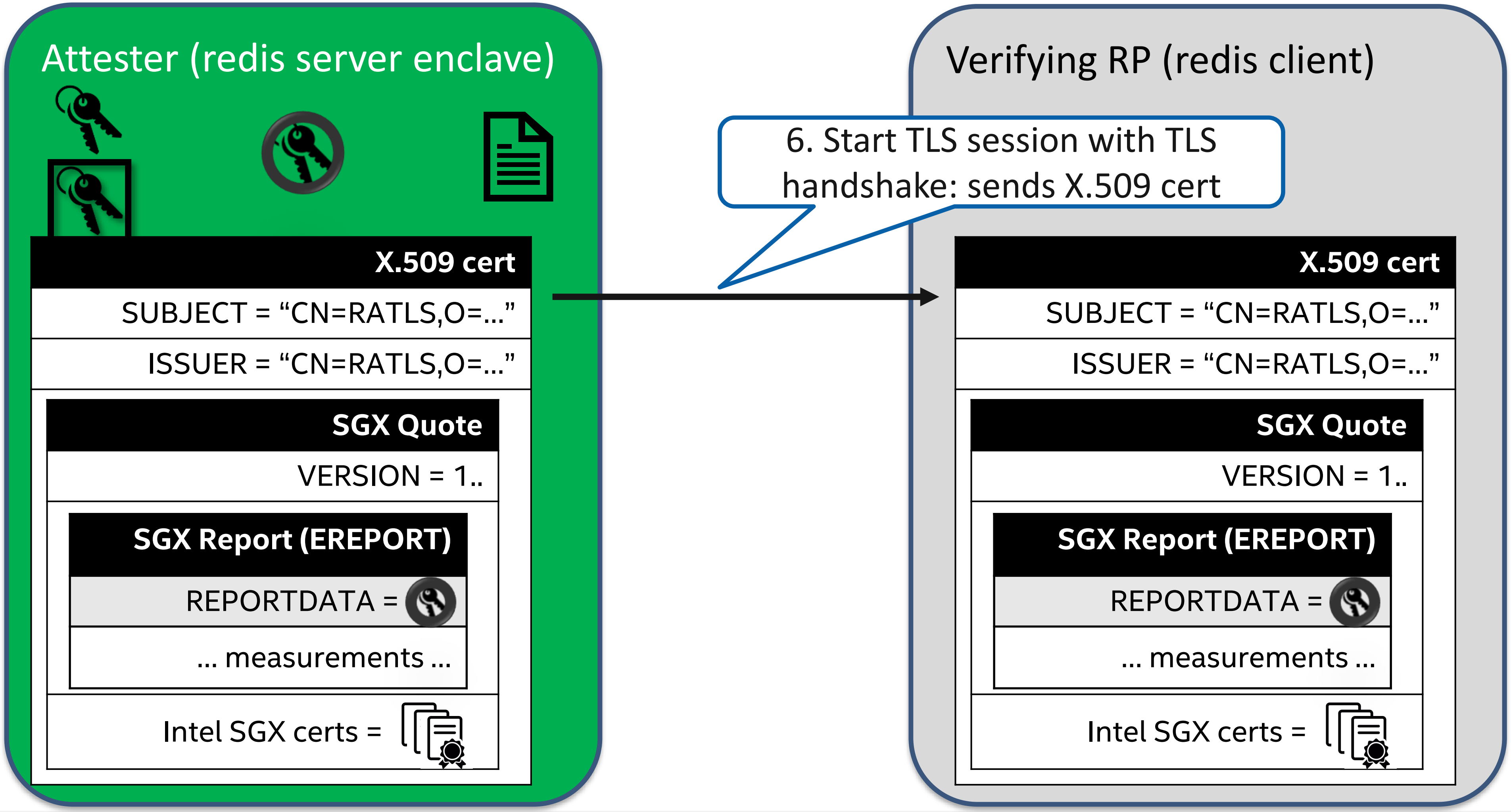
Intel SGX certs =

RA-TLS flow


Intel® Provisioning Certification Service (Intel PCS)
Attester (redis server enclave)
Quoting Enclave
intel SGX
Provisioning Certification Enclave
intel SGX
X.509 cert
SUBJECT = "CN=RATLS,O=..."
ISSUER = "CN=RATLS,O=..."
SGX Quote
VERSION = 1..
SGX Report (EREPORT)
REPORTDATA =
... measurements ...
Intel SGX certs =

5. Generate self-signed X.509 cert with SGX Quote embedded

RA-TLS flow



Attester (redis server enclave)

X.509 cert

SUBJECT = “CN=RATLS,O=…”

ISSUER = “CN=RATLS,O=…”

SGX Quote

VERSION = 1..

SGX Report (EREPORT)

REPORTDATA =

… measurements …

Intel SGX certs =

6. Start TLS session with TLS handshake: sends X.509 cert

Verifying RP (redis client)

X.509 cert

SUBJECT = “CN=RATLS,O=…”

ISSUER = “CN=RATLS,O=…”

SGX Quote

VERSION = 1..

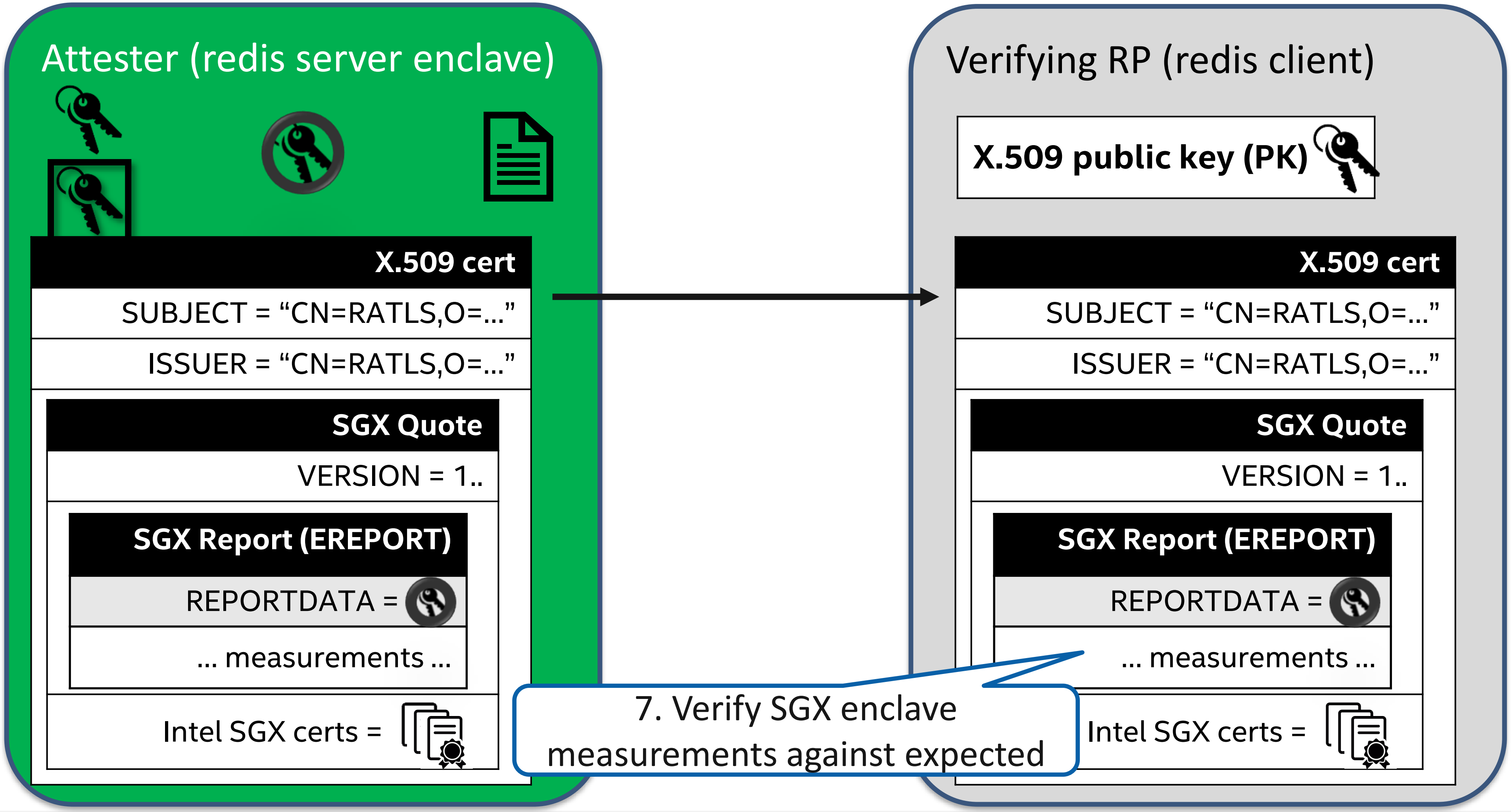
SGX Report (EREPORT)

REPORTDATA =

… measurements …

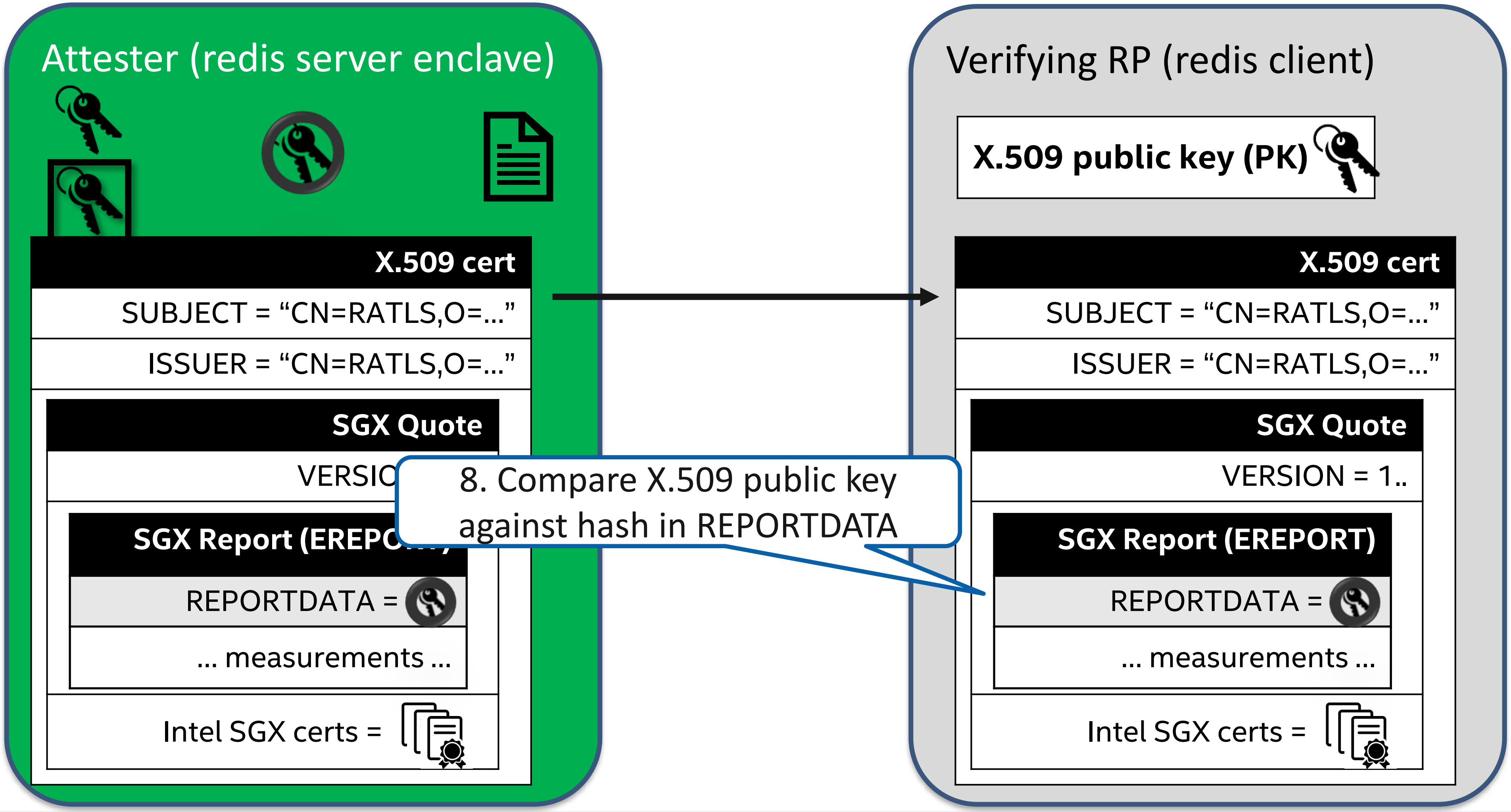
Intel SGX certs =

RA-TLS flow



7. Verify SGX enclave measurements against expected

RA-TLS flow



Attester (redis server enclave)

X.509 cert

SUBJECT = “CN=RATLS,O=...”

ISSUER = “CN=RATLS,O=...”

SGX Quote

VERSIO

SGX Report (EREPO

REPORTDATA =

... measurements ...

Intel SGX certs =

8. Compare X.509 public key against hash in REPORTDATA

Verifying RP (redis client)

X.509 public key (PK)

X.509 cert

SUBJECT = “CN=RATLS,O=...”

ISSUER = “CN=RATLS,O=...”

SGX Quote

VERSION = 1..

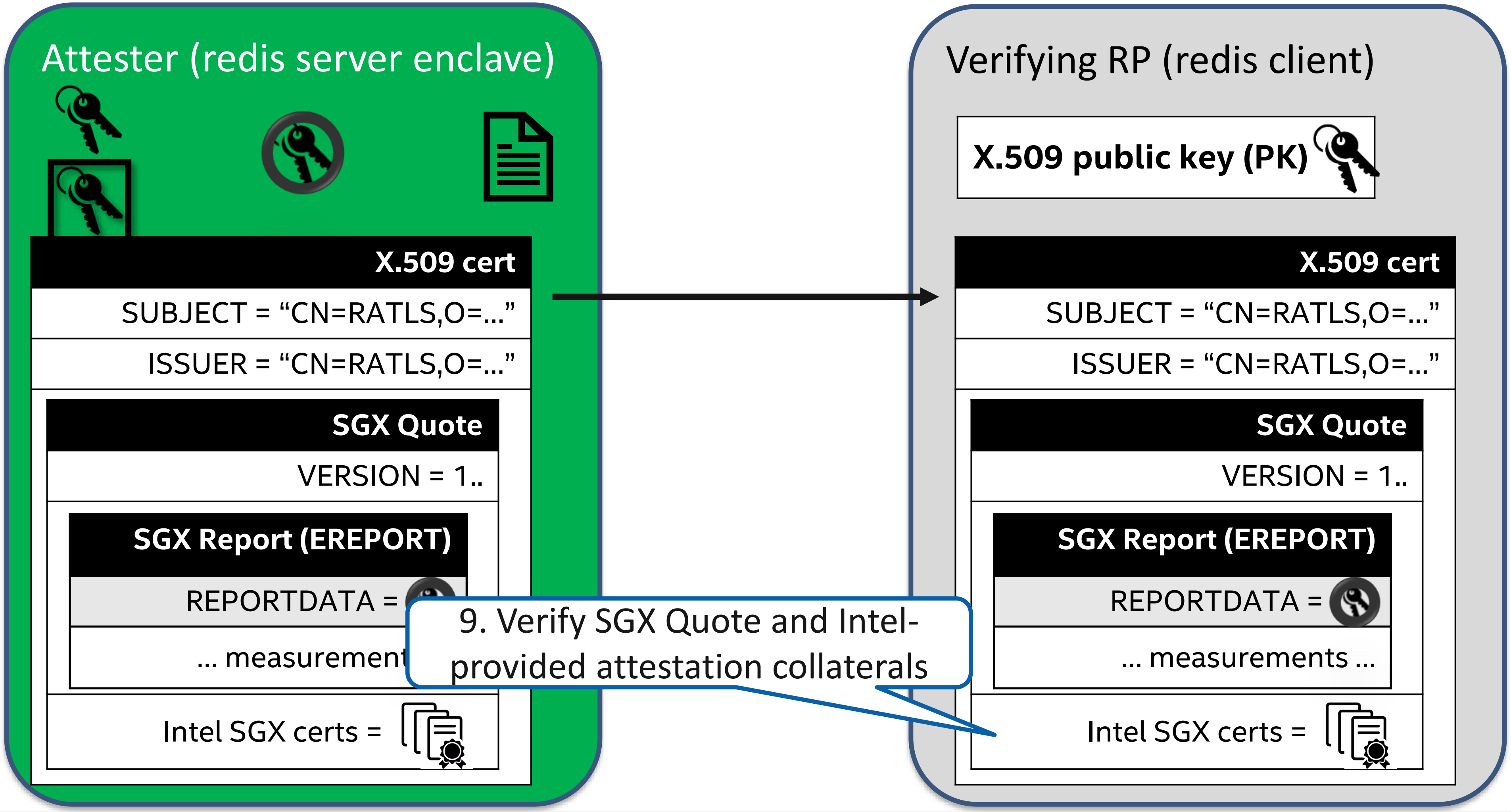
SGX Report (EREPORT)

REPORTDATA =

... measurements ...

Intel SGX certs =

RA-TLS flow



Attester (redis server enclave)

X.509 cert

SUBJECT = “CN=RATLS,O=…”

ISSUER = “CN=RATLS,O=…”

SGX Quote

VERSION = 1..

SGX Report (EREPORT)

REPORTDATA =

… measurements …

Intel SGX certs =

Verifying RP (redis client)

X.509 public key (PK)

X.509 cert

SUBJECT = “CN=RATLS,O=…”

ISSUER = “CN=RATLS,O=…”

SGX Quote

VERSION = 1..

SGX Report (EREPORT)

REPORTDATA =

… measurements …

Intel SGX certs =

9. Verify SGX Quote and Intel-provided attestation collaterals

SGX attestation in Gramine

- Gramine is a **multi-process Library OS** and portability framework
  - Unmodified Linux applications running on several backends
  - Current backends: Linux (direct), Linux-SGX
- Depending on the app needs, Gramine provides three tiers of attestation
  - Low-level interface: `/dev/attestation` files
  - Mid-level interface: RA-TLS library (TLS certificates with SGX quote embedded)
  - High-level interface: Secret Provisioning library (automatic RA-TLS channel)
- **Attestation examples** shipped with Gramine:
  - RA-TLS example (minimal changes to application)
  - Secret provisioning example (no changes to application)

* see https://gramine.readthedocs.io/en/latest/attestation.html

Enabling RA-TLS in manifest

```
sgx.remote_attestation = true

# - app source code must be modified to use RA-TLS API
# - app build must be modified to link against RA-TLS libs
sgx.trusted_files = [
  “file:libra_tls_attest.so”,
  “file:libra_tls_verify_dcap.so”,
]
```

* example at https://github.com/gramineproject/gramine/tree/master/CI-Examples/ra-tls-mbedtls

Enabling Secret Provisioning in manifest

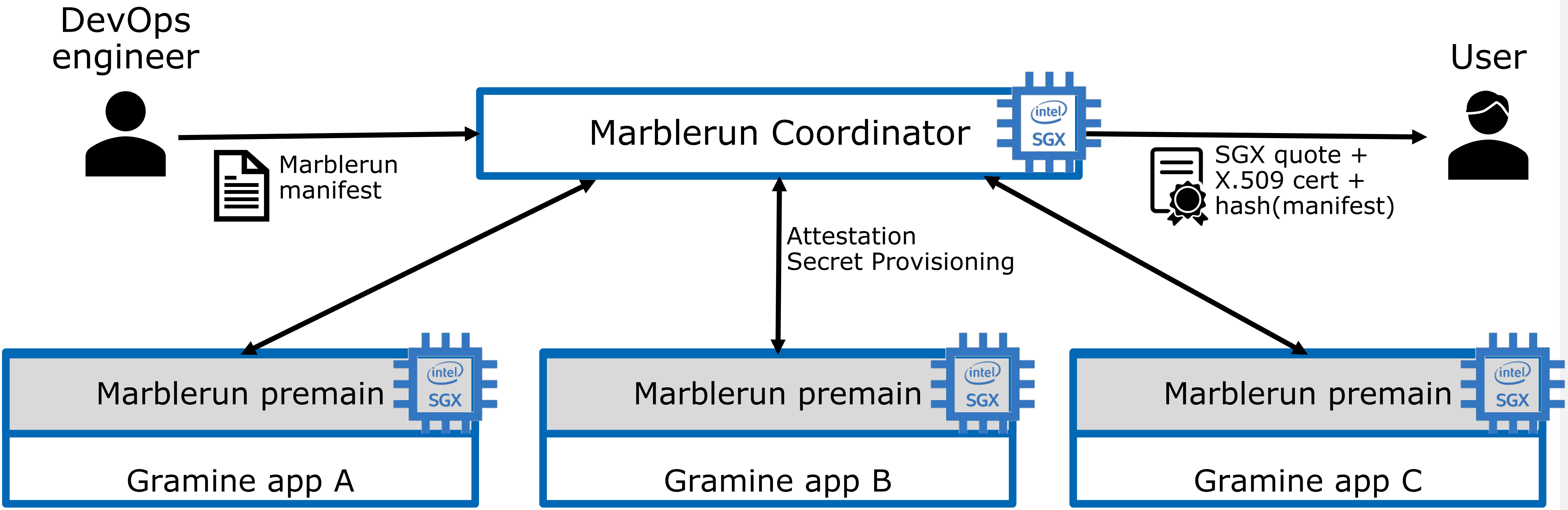
```
sgx.remote_attestation = true

# app doesn’t need to be modified
loader.env.LD_PRELOAD = "libsecret_prov_attest.so“

loader.env.SECRET_PROVISION_CONSTRUCTOR   = "1"
loader.env.SECRET_PROVISION_CA_CHAIN_PATH = “secret-prov-server-ca.crt"
loader.env.SECRET_PROVISION_SERVERS       = “ms-azure-server-a:4433"
```

* example at https://github.com/gramineproject/gramine/tree/master/CI-Examples/ra-tls-secret-prov

Edgeless Marblerun 3rd party solution



References

- https://gramine.readthedocs.io/en/latest/attestation.html
- https://arxiv.org/ftp/arxiv/papers/1801/1801.05863.pdf
- RA-TLS interface (C header file)
- Secret Provisioning interface (C header file)

Gramine project:
http://www.gramineproject.io

GitHub repo:
https://github.com/gramineproject/gramine/

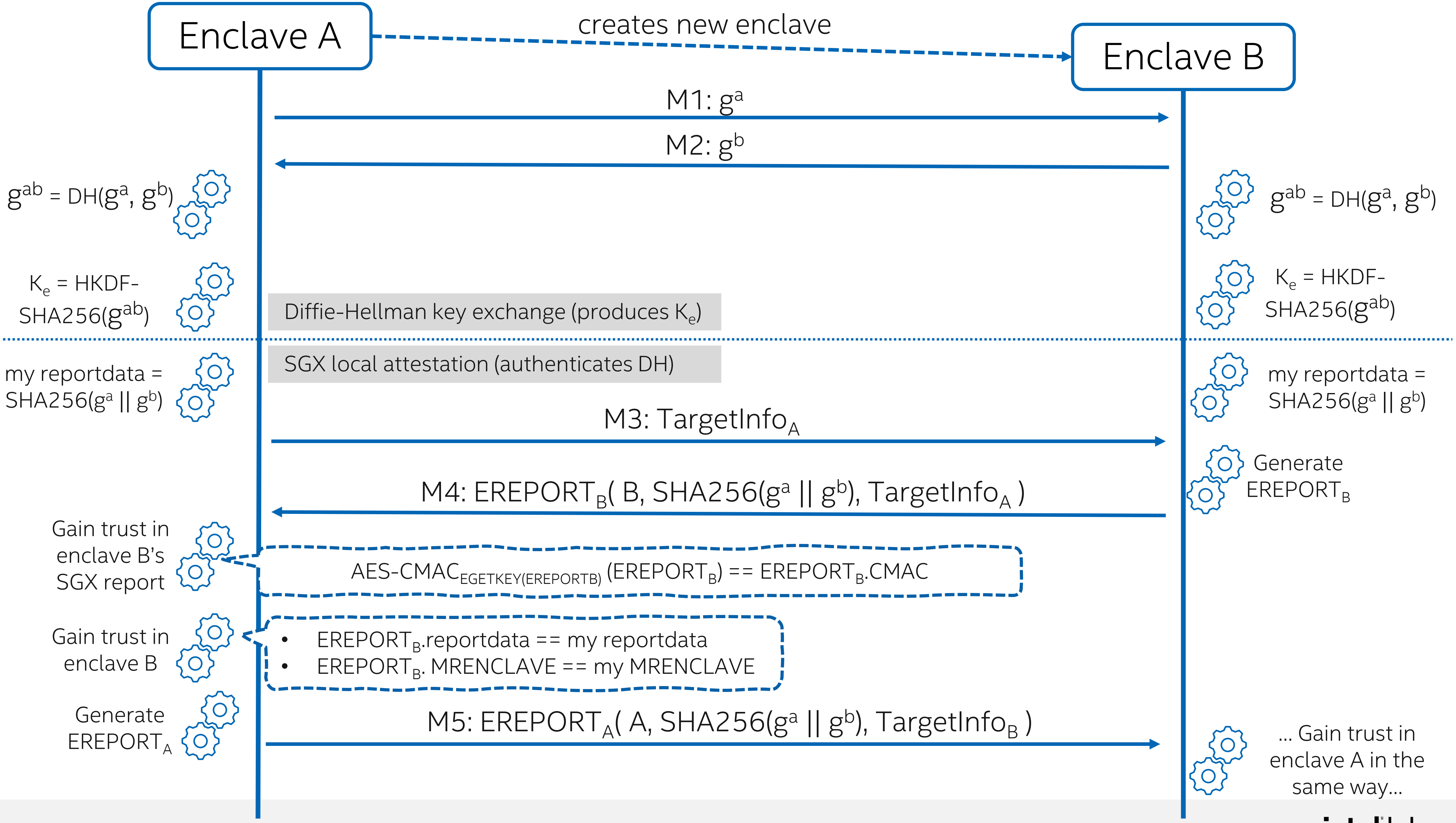
Backup slides

RA-TLS details

- RA-TLS integrates SGX RA with the establishment of the standard TLS secure channel protocol
  - It is NOT a TLS library
  - It provides an ephemeral key and X.509 certificate generation API
- RA-TLS internally uses mbedTLS (light-weight embedded TLS lib)
  - Relies on the same mbedTLS version + config as rest of Gramine
  - Currently mbedTLS v2.26.0 with minimal config and HW entropy source (currently upgrading to 3.1)
  - `mbedtls_rsa_gen_key()`, `mbedtls_sha256()`, `mbedtls_x509write_crt()`, …
- Integrates with existing TLS libraries like mbedTLS, OpenSSL, …
  - TLS protocol is unchanged
  - Hook into existing TLS library:
    - Server side: `ra_tls_create_key_and_crt(&private_key, &cert)`
    - Client side: `mbedtls_ssl_conf_verify(ra_tls_verify_callback)`

RA-TLS details (continued)

- Generate ephemeral TLS keypair inside SGX enclave
  - Current implementation: RSA PKCS#1 v1.5, 3072 bits
  - Source of entropy: rdrand instruction (via Gramine's "`/dev/{u}random`")
- Bind TLS keypair to enclave through SGX user report data
  - SHA256 hash over RSA 3072-bit public key using `mbedtls_sha256()`
  - This hash is copied into first 32B of `sgx_report.user_report_data`
- Create self-signed X.509 cert with this TLS keypair
  - Subject/issuer fields: "CN=RATLS,O=GramineDevelopers,C=US"
  - Timestamp fields: tunable, 2010-2030 by default
  - Signature algorithm: SHA256
  - Embed SGX Quote as X.509 extension: "06 09 2A 86 48 86 F8 4D 8A 39 06"



Enclave A

creates new enclave

Enclave B

M1: g^a

M2: g^b

$g^{ab} = DH(g^a, g^b)$

$g^{ab} = DH(g^a, g^b)$

K_e = HKDF-SHA256(g^{ab})

Diffie-Hellman key exchange (produces K_e)

K_e = HKDF-SHA256(g^{ab})

my reportdata = SHA256(g^a || g^b)

SGX local attestation (authenticates DH)

my reportdata = SHA256(g^a || g^b)

M3: TargetInfo_A

Generate EREPORT_B

M4: EREPORT_B(B, SHA256(g^a || g^b), TargetInfo_A)

Gain trust in enclave B's SGX report

$\text{AES-CMAC}_{\text{EGETKEY(EREPORTB)}}$ (EREPORT_B) == EREPORT_B.CMAC

Gain trust in enclave B

- EREPORT_B.reportdata == my reportdata
- EREPORT_B. MRENCLAVE == my MRENCLAVE

Generate EREPORT_A

M5: EREPORT_A(A, SHA256(g^a || g^b), TargetInfo_B)

… Gain trust in enclave A in the same way…

Gramine usage

```
# install Gramine
$ sudo apt-get install gramine

# prepare the signing key (3072 RSA key as required by Intel SGX)
$ gramine-sgx-gen-key private.pem

# let’s try Redis example
$ cd CI-Examples/redis
```

Gramine usage

```
# prepare the manifest file for your app (see next slide)
$ vim redis-server.manifest.template

# generate Gramine- and SGX-specific files
$ gramine-manifest redis-server.manifest.template redis-server.manifest
$ gramine-sgx-sign --key private.pem --manifest redis-server.manifest \
                   --output redis-server.manifest.sgx
$ gramine-sgx-get-token --sig redis-server.sig \
                        --output redis-server.token

# run Gramine in SGX mode
$ gramine-sgx redis-server
1:C 14 Mar 2022 09:50:23.866 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 14 Mar 2022 09:50:23.866 # Redis version=6.0.5, ...
1:M 14 Mar 2022 09:50:23.867 # Server initialized
1:M 14 Mar 2022 09:50:23.902 * Ready to accept connections
```

Gramine manifest for Redis

```
libos.entrypoint = “redis-server”

loader.env.LD_LIBRARY_PATH = “/lib”

fs.mount.lib.type = “chroot”
fs.mount.lib.path = “/lib”
fs.mount.lib.uri = “file:/usr/local/lib/gramine”

sgx.enclave_size = “1024M”
sgx.thread_num = 8

sgx.trusted_files = [ “file:/usr/local/lib/gramine/libc.so.6” ]
…
```