



WHITE PAPER • CONFIDENTIAL COMPUTING

Introducing TractorOS: Sealed RHEL 10 Images, Attested on Intel® TDX

How FarmGPU builds confidential VMs whose disk keys are released only after Intel® Trust Authority has appraised them as exactly the image we published, and what a tenant can verify for themselves.

INTEL® TDX

INTEL® TRUST AUTHORITY

RHEL 10 IMAGE MODE

AT A GLANCE

TractorOS is the operating system under FarmGPU Cloud: Red Hat® Enterprise Linux® 10 in image mode for every host and every guest, built from a Containerfile into a bootable image. On Intel® Xeon® 6 processors with Intel® Trust Domain Extensions (Intel® TDX) it runs confidential VMs whose disk keys are released only to a VM that Intel® Trust Authority has appraised as exactly the image we published.

01

The operating system is an image, not a server.

Hosts and guests boot an immutable image, so what runs is exactly what we built, and the measurement registers of a confidential VM can prove it.

02

The supply chain is verified, not asserted.

The build checks Red Hat's signatures and manifests for every package and seals its own SBOM into the measured root.

03

The operator is designed out.

A VM gets its disk key only after Intel Trust Authority appraises it. We never see the key, the console or the memory.

01

Why a neocloud needs its own operating system

Renting a machine from a cloud means trusting its operator with three things at once: the software it runs, the people who can reach it, and the data on its disks. Every provider says "trust us". We would rather give you the means to check.

TractorOS is our answer in three parts:

1. **The operating system is an image, not a server.** Hosts and guests boot an immutable RHEL 10 image built from a Containerfile, so what runs is exactly what we built, byte for byte, and the measurement registers of a confidential VM can prove it.
2. **The supply chain is verified, not asserted.** The confidential guest build checks Red Hat's signature on the base image, Red Hat's manifest for that exact build, and the signature on every package, then seals its own software bill of materials into the measured root. A tenant can walk that chain back to Red Hat without taking our word for any step.
3. **The operator is designed out.** A confidential VM on Intel TDX gets its disk key only after Intel Trust Authority has appraised it as the published image; we never see the key, the console or the memory. Deleting the VM deletes the key. Section 5 walks through exactly what the operator can and cannot do.

The rest of this paper goes through each part with the real artifacts: the boot chain, the measurement registers, the verification report and the attestation flow.

The OS is an image

TractorOS is built with RHEL 10 image mode: the operating system is a bootable OCI image, and `bootc` boots it. One Containerfile per variant, one build chain for all of them, and the same tools we already use for application images.

Variants, not snowflakes. A shared base carries SSH, telemetry and operator access for the fleet. Host variants add what a role needs: the KVM GPU host with VFIO passthrough and NVSwitch fabric management, the Intel TDX host with QEMU, the DCAP quote service and the Intel Trust Authority client. Guest variants are leaner still, and the newest confidential guest is built directly from Red Hat's `rhel-bootc` image, pinned by digest, with nothing from any third-party repository.

One chain from Containerfile to running machine. A build runs on an entitled RHEL builder: `podman build` with the host's subscription mounted in, every version pinned in one file, every image stamped with its build date and git hash, tagged by version and pushed to our registry. Hosts move between releases with `bootc switch`, and the previous image stays on disk for rollback. Guests become disks with `bootc install to-disk` straight from the image. Every push is announced to the team, and a daily watch runs `dnf check-update` inside each published image against Red Hat's advisories and says in plain words whether to rebuild now, schedule it, or do nothing.

Why this matters for trust. An image built this way has a single, knowable content: the composefs digest of its root tree. That digest is what the rest of this paper measures, seals and attests.

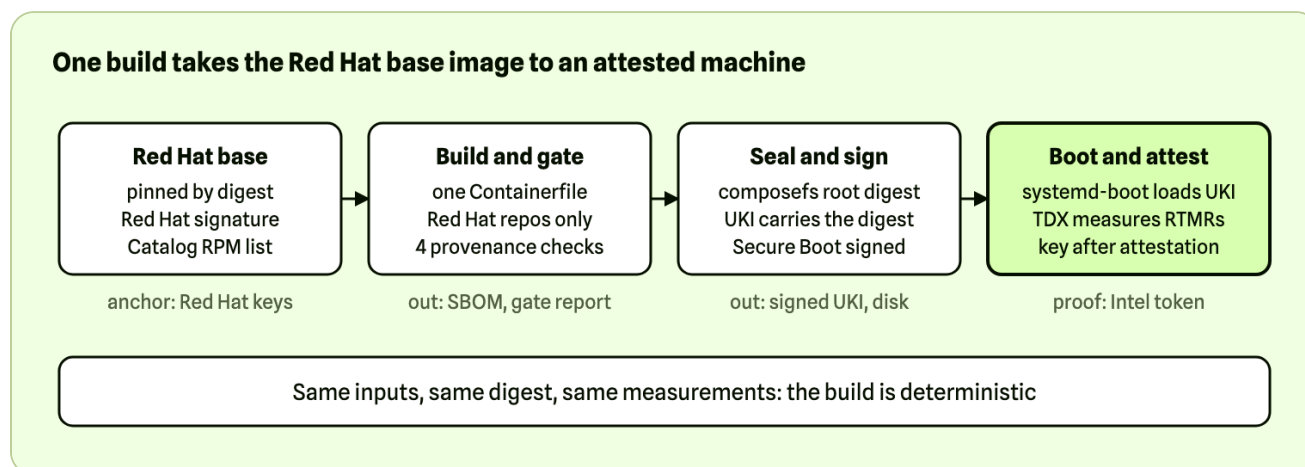


FIGURE 1 Each stage consumes what the previous one signed: Red Hat keys anchor the base, the gate and the SBOM describe the build, the UKI carries the root digest, and Intel TDX measures the result.

Sealed images: the whole root in one measured number

A conventional VM image is a disk the host can read and rewrite. A sealed TractorOS guest is different in three ways, and together they make "this exact image" a provable statement.

- **One signed boot object.** The kernel, the initramfs and the kernel command line are packed into a Unified Kernel Image (UKI) and signed with our Secure Boot key, as is the systemd-boot loader in front of it. There is no GRUB and no shim in the guest, and the loader changes only when a new image ships.
- **The root filesystem is a digest.** The command line inside the UKI carries `composefs=<digest>`, the digest of the entire root tree. At boot the initramfs mounts the root as composefs with fs-verity set to `require`: a file whose bytes differ from the digest cannot be read. The root is read-only by construction.
- **State lives behind a key the host never has.** Everything writable, `/var` and `/etc`, sits on a LUKS2 volume created on first boot and unlocked in the initramfs with a key released by our key broker only after attestation. Configuration reaches the VM the same way; cloud-init runs with its datasource set to none, so the host has no channel to inject anything.

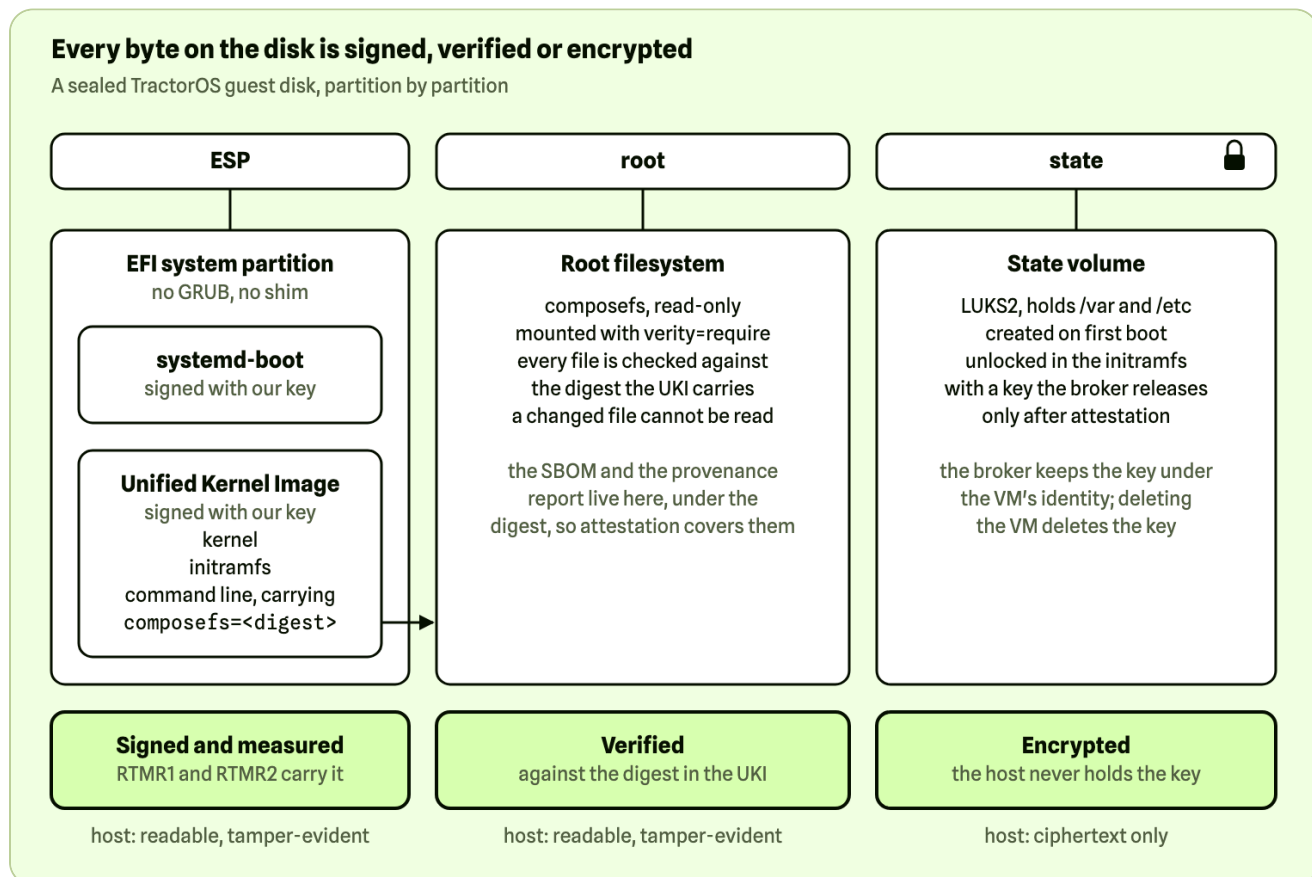


FIGURE 2 The host holds the whole disk. It can read the boot and root partitions, which are the published image anyway, it cannot change a byte of them without a register moving, and it holds the state volume only as ciphertext.

On Intel TDX the firmware and the boot stub measure each stage into registers that Intel signs into every attestation token, alongside the owner fields set at launch. With a sealed image the mapping is clean:

MEASUREMENT	WHAT IS MEASURED AND ATTESTED	WHERE THE REFERENCE VALUE COMES FROM
MRTD	FarmGPU's TDVF firmware binary	Red Hat's edk2 TDVF build, with our Secure Boot keys enrolled
RTMR0	firmware configuration, including Secure Boot policy and VM memory and CPU layout	one recording per image and shape; 10 shapes today
RTMR1	bootloader and UKI	the signed UKI of the release
RTMR2	kernel, initrd and command line, and with it the root filesystem digest	the same build, by construction
RTMR3	runtime measurements	unused by the image; free for the guest to extend at runtime
MROWNER	the owner ID the provider assigned at launch: the tenant	set by our API when the VM is created
MROWNERCONFIG	the instance ID the provider assigned at launch: this VM	set by our API when the VM is created

The event log the firmware keeps lets us replay all three runtime registers on the host and compare them with Intel's token, which our release process does for every recording. The reference values go into a Reference Integrity Manifest at Intel Trust Authority, one entry per image and shape, and a VM whose registers do not match an entry gets no key.

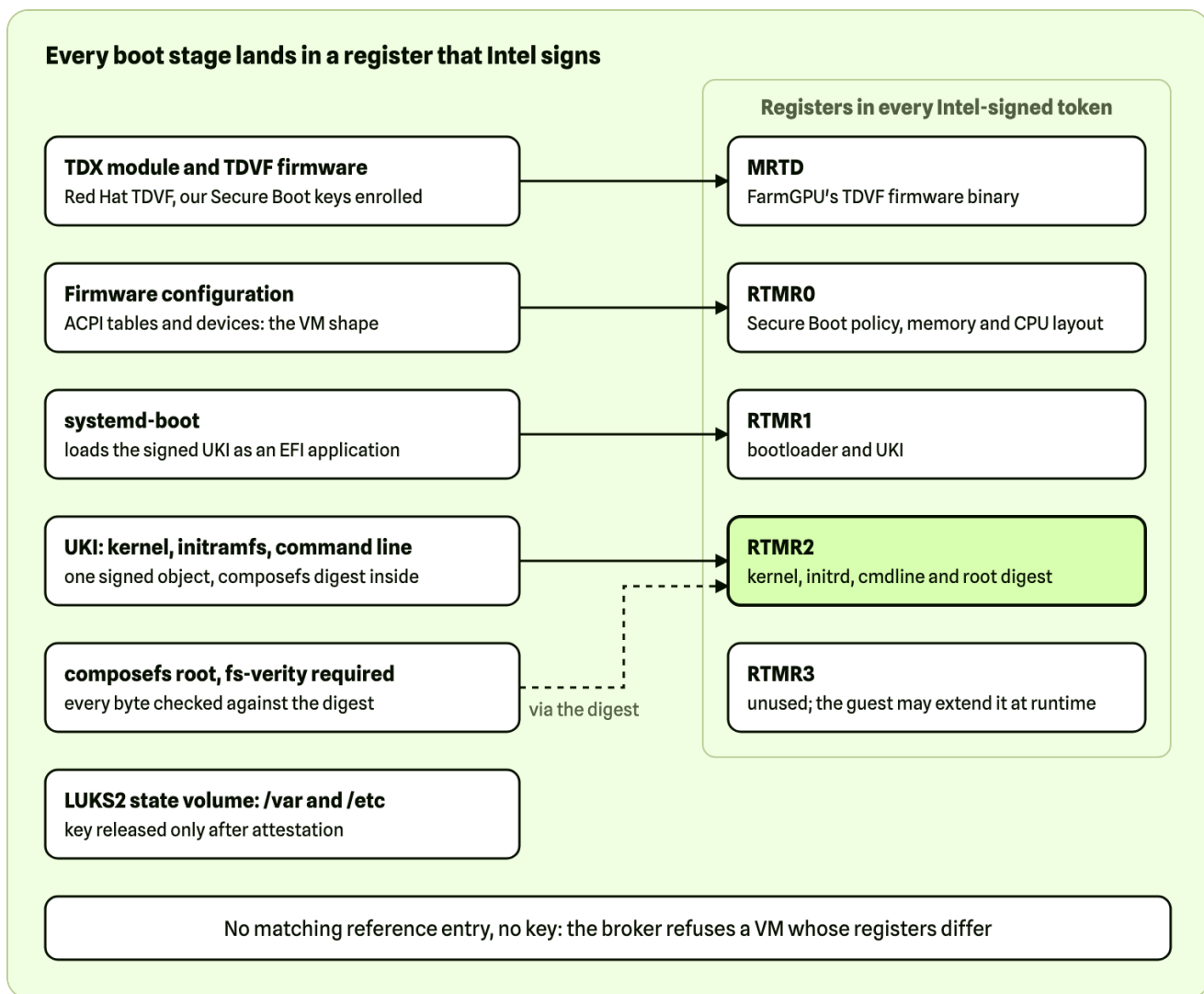


FIGURE 3 The root digest rides inside the UKI, so RTMR2 binds the entire root filesystem. The state volume is never measured because it is never used without a key that attestation released.

Change anything and a value Intel signs changes with it

What a hostile host would have to change, and which signed value gives it away

	MRTD	RTMR0	RTMR1	RTMR2	Owner
Swap the virtual firmware	●				
Turn off Secure Boot or enroll another key		●			
Change the VM's shape or attach a device		●			
Replace the boot loader			●		
Swap the kernel, initramfs or command line			○	●	
Change one file in the root filesystem			○	●	
Boot it as another tenant or instance					●

● where it shows up ○ changes with it

No key
no configuration

FIGURE 4 Each row is something a hostile host could change, and the dot is the signed value that gives it away. The state volume needs no row: it is encrypted, and the key is exactly what a mismatch withholds.

04

Proving where every byte came from

A measured root digest answers "is this the image we built?" It does not answer "what is in it, and who vouches for that?" For the RHEL-only confidential guest we close that gap at build time, and we leave the evidence inside the sealed tree so the attestation covers it too.

The build gate. Before the root is sealed, a gate runs inside the build with the network switched off, against a directory of public Red Hat material fetched and pre-verified on the build host. It fails the build unless four things hold:

1. **The base image is Red Hat's.** Red Hat publishes a detached signature for every image it ships. The one for our pinned `rhel-bootc` digest must verify with Red Hat release key 2 and name that repository and that digest.
2. **The base layer matches Red Hat's manifest.** The Red Hat Ecosystem Catalog publishes the RPM list of every image build. Every package in our pristine base must appear in it.
3. **Every RPM is signed by an allow-listed key.** For this variant the allow list has one entry: Red Hat release key 2. One unsigned or foreign package fails the build.
4. **Every package is traced to a Red Hat inventory.** Red Hat ships a GPG-signed SPDX SBOM per stream. Between it and the Catalog manifest, the gate records which Red Hat document vouches for each package in the image.

The keys are pinned by fingerprint in the repository, not fetched by URL and trusted. A tampered evidence directory fails the build the same way a tampered package would.

Our SBOM. The same build writes an SPDX 2.3 SBOM straight from the RPM database: a purl per package that scanners such as gype match against Red Hat's security data, the header checksum, the license, the source RPM, and an annotation naming the signing key and which Red Hat inventory covers it. Each Red Hat package also references the element describing it in Red Hat's own SBOM through an SPDX external document reference, checksum included. Nine non-RPM components we build or verify ourselves, from the attestation agent to the Intel Trust Authority client, are listed with their SHA-256. Nothing in it depends on the wall clock except the build date it is handed, so the same inputs give the same bytes.

Verifying from inside. The SBOM and the gate's report live under the measured root, and their digests are stamped in the OS version file. A tenant runs one command in the VM (output abridged):

```
$ tractoros-verify
TractorOS tdx-guest-rhel-sealed v0.2.0-rc1 (built 2026-10-02T...Z, git ...)
base image      registry.redhat.io/rhel10/rhel-bootc@sha256:d35c3981...
root            overlay (ro,...,verity=require)
composefs digest eaea01fa...
SBOM            /usr/share/tractoros/sbom/tractoros.spdx.json
                sha256 ... recorded ... match
upstream report /usr/share/tractoros/provenance/upstream-verification.json
                sha256 ... recorded ... match; result PASS
rpm signers (live) 199e2f91fd431d51=493
verdict         OK
```

And from outside, the chain reads top to bottom: a fresh Intel-signed token with the tenant's own nonce, the reference entry it matched, the image digest that entry was recorded from, the SBOM digest sealed in that image, and Red Hat's signed document behind it. No step asks the tenant to take our word for it.

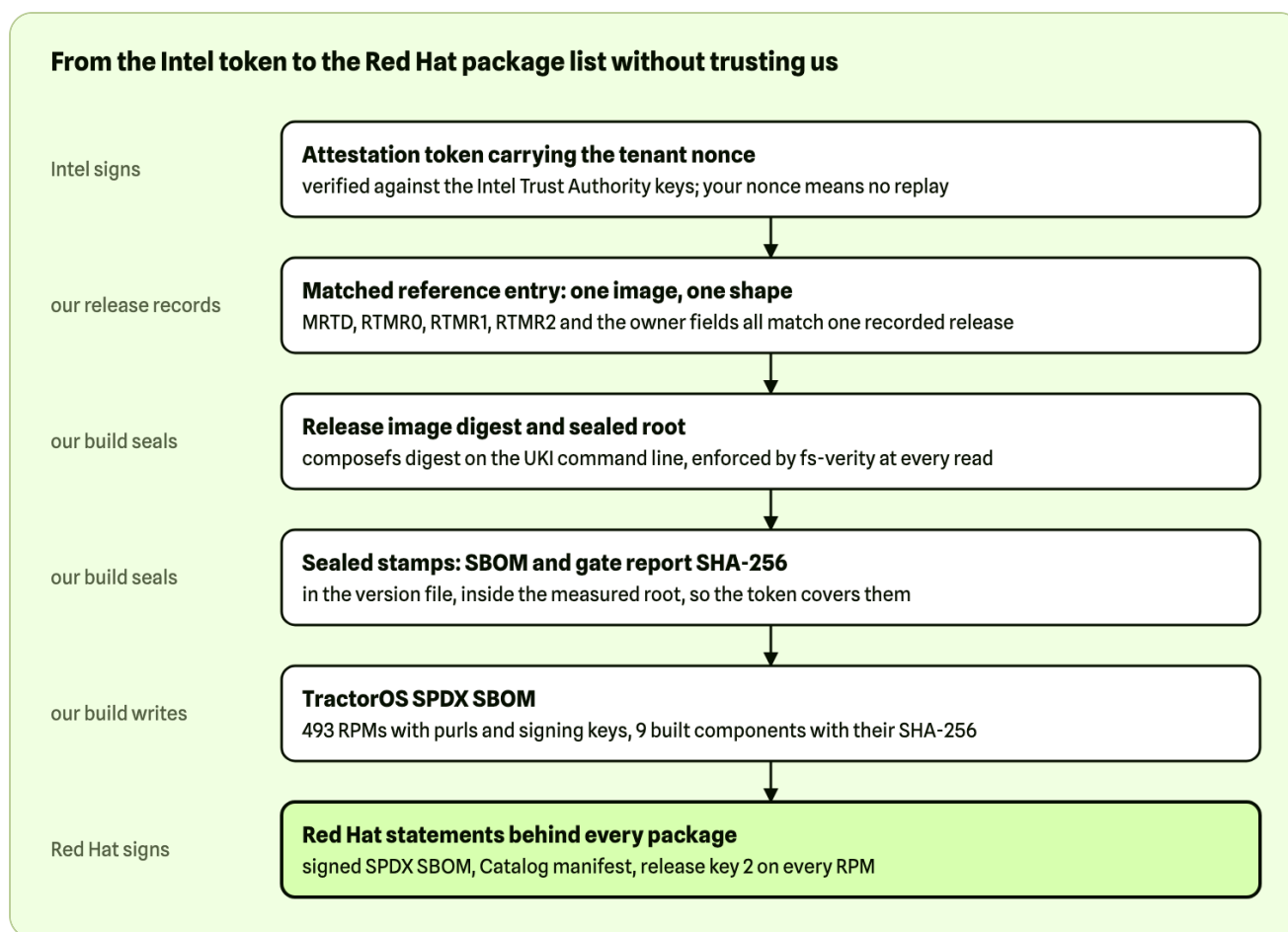


FIGURE 5 Every link is a signature or a digest the tenant checks themselves. We appear only as the party that recorded a release and ran a build whose outputs are now sealed.

05

Confidential VMs with Intel® TDX and Intel® Trust Authority

Intel Trust Domain Extensions (Intel TDX) run a VM, a trust domain, in memory the host cannot read, with CPU state the hypervisor cannot inspect. That removes the cloud provider from the trust boundary. It does not, on its own, tell the tenant what is running inside. Attestation does, and we built the rest of the stack so that attestation is the only thing a tenant has to trust.

Every TD has an identity. When a VM is created we set MROWNER and MROWNERCONFIG, the TDX owner fields, from the tenant and the instance. They are part of the measurement Intel signs, so a token proves not only "a TractorOS image booted" but "this tenant's instance booted". Tenant VMs carry no serial console at all: there is no path from the host into the guest's terminal.

The key comes from attestation, not from the host. On first boot the initramfs asks the TDX module for a quote and opens one attested session with our key broker, built on the Confidential Containers project's Trustee. The broker sends the quote to Intel Trust Authority, which verifies the platform, appraises the measurements against our Reference Integrity Manifest under our appraisal policy, and returns a signed JWT. The broker checks that token against Intel's published keys, applies its own policy, and only then

releases the key for the VM's state volume and the VM's configuration over that same session. A VM whose firmware, UKI, shape or identity does not match gets nothing. The host sees an opaque VM talking to a broker and never holds the key. The VM keeps Intel's token too, so it carries its own proof of what it had to show.

Deleting a VM is cryptographic. The state volume is the only writable storage, and its key lives in the broker under the VM's identity. Deleting the VM removes the broker entry first and the disk second, so even a disk that outlives its VM is noise.

Tenants attest on demand. Our API exposes attestation as a first-class call. The tenant supplies a nonce. Our in-guest agent, reached over vsock with no network path, produces a quote whose report data is the hash of that nonce; the host checks that the nonce really is in the quote, sends the quote to Intel for appraisal, and returns the quote, the event log and Intel's token. The tenant verifies the token against Intel's published keys themselves. A token with their nonce in it cannot be replayed from last week, and the VM holds no Intel credential of its own. The token carries the registers, the owner fields and the policy entries that matched, which is what the verification chain in the previous section walks.

One Intel-signed token names the image, the tenant and the moment

Decoded from a running trust domain; long values abridged

```
{
  "iss": "https://portal.trustauthority.intel.com",
  "iat": 1791487219, "exp": 1791487519,
  "jti": "c5bcb647-...",
  "policy_ids_matched": [
    { "id": "dc1abd94-...", "version": "v3" } ],
  "tdx": {
    "attester_type": "TDX",
    "tdx_mrtid": "2057eec6368d0348...",
    "tdx_rtmr0": "d99c254265fcc1b7...",
    "tdx_rtmr1": "ebedb2b9e46d677b...",
    "tdx_rtmr2": "c63476af61c70d4e...",
    "tdx_rtmr3": "0000000000000000...",
    "tdx_mrowner": "4f0fcfa9...",
    "tdx_mrownerconfig": "c7bc3f69...",
    "tdx_report_data": "80e5dc335451ed40...",
    "tdx_is_debuggable": false,
    "tdx_seamsvn": 775,
    ...
  }
}
```

Signed by Intel with PS384, under a key in Intel's published JWKS; valid five minutes

Our appraisal policy matched: the registers equal a Reference Integrity Manifest entry for a published image and shape

Firmware, VM shape, loader, and the UKI with the root digest, exactly as the release process recorded them

Tenant and instance, set at creation: this tenant's VM, not merely this image

SHA-512 of the tenant's nonce: no replays

Debug off, and the TDX module version Intel vouches for

FIGURE 6 This is the token one such call returned, decoded. Everything the tenant needs is inside it: Intel's signature, the policy that matched, the registers, the owner fields and the hash of their own nonce. The host only forwarded it.

The host side is TractorOS too. The Intel TDX host variant is a bootc image on stock RHEL 10 that carries QEMU with TDX support, Red Hat's TDVF firmware with our Secure Boot keys enrolled, the DCAP quoting

stack, and a small management API. Every VM is created through that API, which sets the owner identity, starts the trust domain and reads boot status from the guest agent rather than a console.

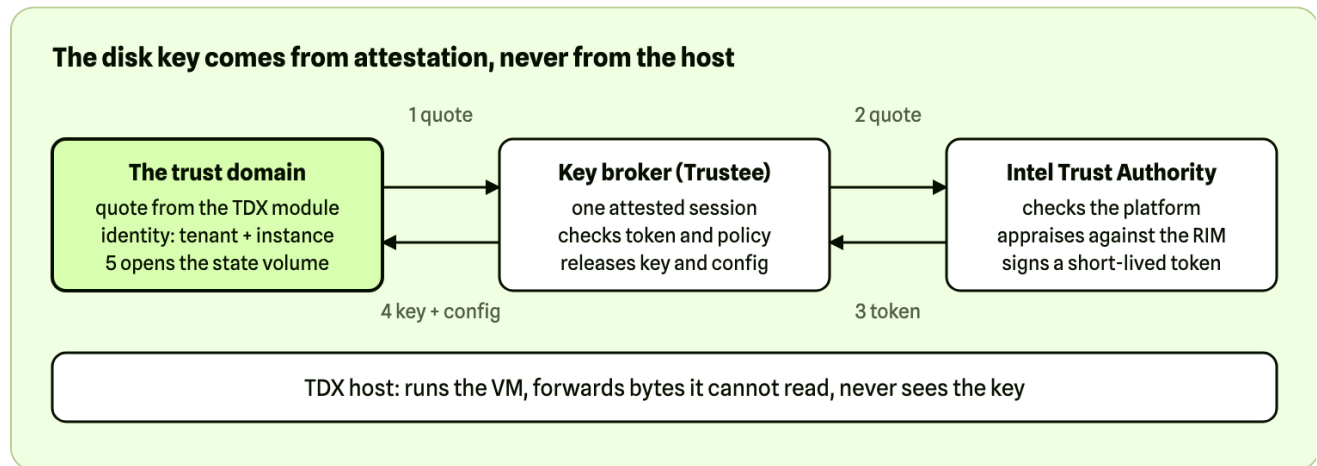


FIGURE 7 The VM talks only to the key broker, and the broker talks to Intel. The host forwards bytes it cannot read, and a mismatch in any register, in the owner fields or in the policy ends with no key.

Two policies you can read in a minute decide every key release

The two Rego policies, comments removed and one constant abridged

1· Intel Trust Authority decides what is running

```
import rego.v1
default match := false
match if {
    input.tdx.tdx_is_debuggable == false
    input.tdx.attester_tcb_status == "UpToDate"
    some img in data.tractoros.tdx_guest_cpu_sealed.images
    img.mrtd == input.tdx.tdx_mrtd
    img.rtmr0 == input.tdx.tdx_rtmr0
    img.rtmr2 == input.tdx.tdx_rtmr2
}
```

- Debug attributes off
- Platform at Intel's current TCB level
- Firmware, shape and the UKI with the root digest must all equal one entry of the Reference Integrity Manifest

Intel's signed token, our policy id in it

2· The key broker decides who is asking

```
package policy
default allow = false
zeros64 := "0000...0000"
allow if {
    data.plugin == "resource"
    tenant := data["resource-path"][0]
    vm := data["resource-path"][1]
    input.tdx.tdx_is_debuggable == false
    input.tdx.tdx_mrowner == concat("", [
        lower(replace(tenant, "-", "")), zeros64])
    input.tdx.tdx_mrownerconfig == concat("", [
        lower(replace(vm, "-", "")), zeros64])
}
```

- A UUID padded to 48 bytes with zeros
- The key's path names a tenant and a VM: tenant/vm/Luks
- Debug attributes off, checked again
- The owner field must be that tenant
- The owner-config field must be that VM

The manifest is data: adding or revoking an image changes the manifest, never the policy or the id a tenant pins.

FIGURE 8 These are the two policies in full. Intel's decides what is running and never changes, because the reference values live in the manifest as data. The broker's decides who is asking, by matching the owner fields against the path of the key being requested.

How the operator is designed out

Putting the pieces together, this is the whole lifecycle of a confidential VM, and the operator's place in it. FarmGPU still runs the hardware: it powers machines, schedules VMs and moves their network traffic. What it cannot do is read or alter anything inside a trust domain without a register moving and the key being withheld.

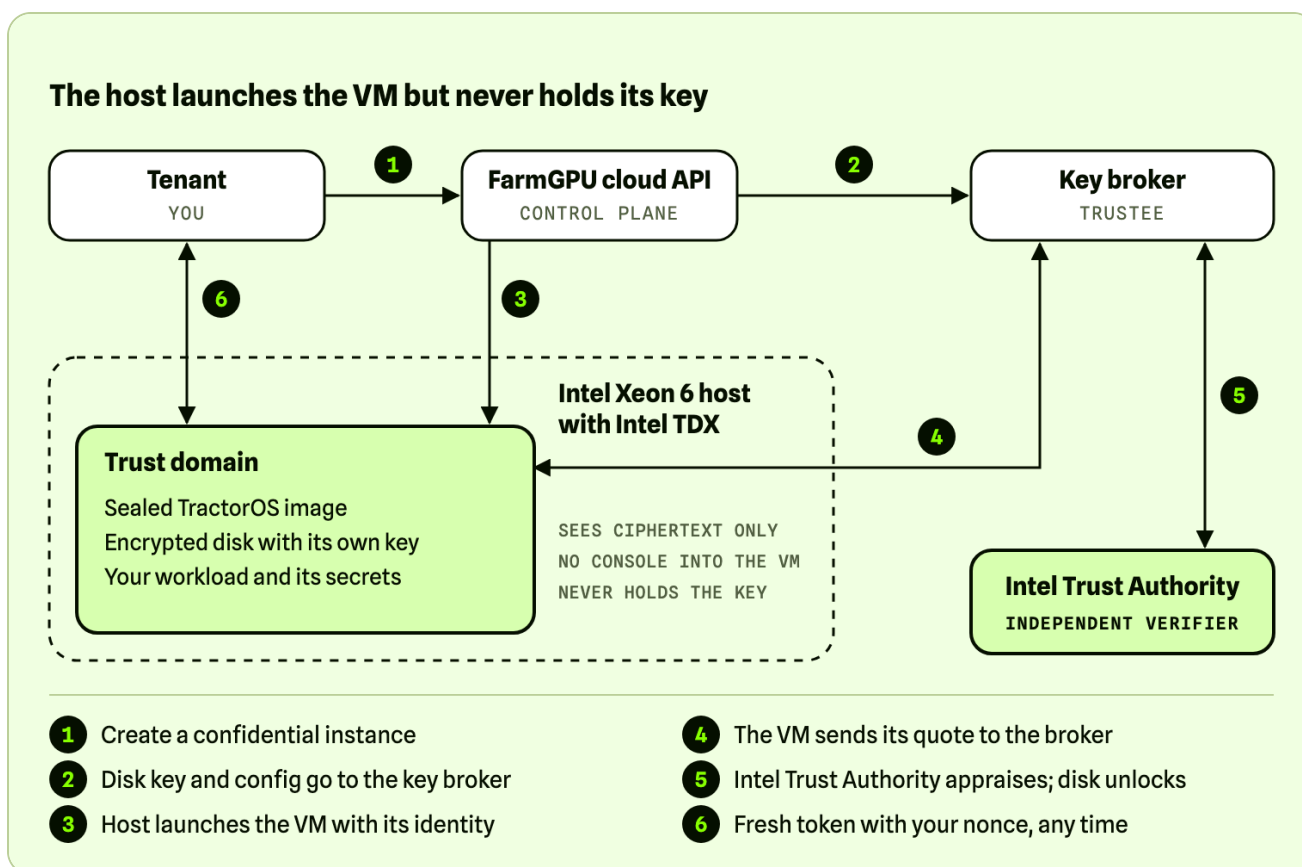


FIGURE 9 The tenant creates an instance through the cloud API, the key broker holds the VM's disk key and configuration, and Intel Trust Authority decides whether the VM that asks for them is the one that was published. The host launches the VM but never holds its key.

WHAT THE OPERATOR CONTROLS

- Power, placement and scheduling of VMs
- The network path, as bytes it cannot read
- Publishing images and their reference values

WHAT THE OPERATOR CANNOT DO

- Read VM memory or CPU state (Intel TDX)
- Open a console: tenant VMs have none
- Inject configuration: it arrives only after attestation
- Hold or reuse the disk key: it is released only to an appraised VM, and deleted with it

Things we found along the way.

- Measurement replay matters. Our release process records every image and shape, replays the firmware's event log against Intel's token, and only publishes reference values that reproduce. A register that does not replay is a bug, not a reference.
- A sealed root plus attestation turns the SBOM question into a measurement question. We did not expect that to feel as clean as it does.

From our fleet to FarmGPU Cloud

Everything in this paper runs today on FarmGPU's Intel Xeon 6 hosts with Intel TDX: sealed RHEL 10 guests, disk keys that exist only after Intel Trust Authority has appraised the VM, attestation on demand through our API, cryptographic deletion, and a build gate that ties every package back to Red Hat's signatures. The same Containerfile-to-bootc chain builds our GPU hosts, so every machine in the fleet, confidential or not, boots an image we can name by digest. TractorOS and confidential computing VMs launch on FarmGPU Cloud by the end of 2026.

What that will mean for a tenant, without taking our word for any of it:

- Boot a confidential VM and get a sealed RHEL 10 guest whose firmware, kernel and root filesystem are all in Intel's signed token.
- Hand our API a nonce and check Intel's signature, the measurement registers and the owner fields against your own expectations.
- Run `tractoros-verify` inside the VM and walk the software bill of materials back to Red Hat's signed data.
- Delete the VM and know the key went before the disk did.

Look out for the launch of TractorOS and confidential computing VMs on FarmGPU Cloud by the end of this year. If you run a confidential workload and want to see the attestation chain end to end before then, write to us at farmgpu.com. We would rather show it than describe it.

Further reading

- [bootc](#) and [RHEL image mode](#): the operating system as a bootable image. Red Hat documents sealed images as a Technology Preview in RHEL 10, with bootc's composefs backend doing the sealing.
- [composefs](#): the read-only, fs-verity-backed root filesystem behind the digest on our kernel command line.
- [Unified Kernel Images](#): the specification our UKIs follow. It describes measurement into TPM PCR 11; on Intel TDX, systemd-stub makes the same measurements into RTMR2 through the firmware's confidential-computing measurement protocol.
- [Intel Trust Domain Extensions](#) and the [TDX Virtual Firmware Design Guide](#): the hardware, the TDX module, and what each measurement register holds.
- [Intel Trust Authority](#): attestation, appraisal policies and Reference Integrity Manifests.
- [Trustee](#): the Confidential Containers key broker our release process and guests talk to.
- [Red Hat security data](#): the signed SPDX SBOMs, VEX files and keys our build gate verifies against.
- [SPDX 2.3](#): the SBOM format we emit and the one Red Hat publishes.



ABOUT FARMGPU

FarmGPU builds and runs its own neocloud software stack, from the operating system on every machine to the cloud API its customers use. Two areas set it apart from other neoclouds and from the hyperscalers: confidential computing and storage for AI.

TractorOS and confidential computing VMs launch on FarmGPU Cloud by the end of 2026.

Learn more at farmgpu.com.

Notices

Intel, the Intel logo, Xeon and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel technologies may require enabled hardware, software or service activation. No product or component can be absolutely secure.

Red Hat, Red Hat Enterprise Linux and RHEL are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. Other names and brands may be claimed as the property of others. © 2026 FarmGPU. This paper describes FarmGPU's implementation as of October 2026.