Meta

Private Processing for WhatsApp Overview

Technical White Paper and Security Guide

V2 updated on March 16, 2026

V1 published on June 10, 2025

Contents

Meta

Contents cont.

Meta

Introducing Private Processing for WhatsApp

AI has revolutionized the way people interact with technology and information, making it possible for people to automate complex tasks and gain valuable insights from vast amounts of data. However, the current state of AI processing – which relies on large language models (LLMs) often running on servers, rather than mobile hardware – requires that users' requests are visible to the provider. Although that works for many use cases, it presents challenges in enabling people to use AI to process private messages while preserving the level of privacy afforded by end-to-end encryption.

We set out to enable AI capabilities with the privacy that people have come to expect from WhatsApp, so that AI can deliver helpful capabilities, such as summarizing messages, without Meta or WhatsApp having access to them.

This white paper provides a technical overview of Meta's Private Processing system as used in WhatsApp.

Private Processing is our name for a server-based processing system built on Trusted Execution Environments (TEEs) with the goal to ensure that sharing messages with Private Processing does not make them available to Meta, WhatsApp, or anyone else. This system is meant to preserve the same level of privacy afforded by end-to-end encryption and on-device processing. However, it is backed by different technology which requires different security guarantees. The goal of Private Processing is to raise the bar for privacy and security for use cases where end-to-end encryption is not possible and on-device processing capabilities are insufficient.

The Private Processing system enables users to leverage AI models like LLMs which require GPUs or other acceleration hardware in a privacy-preserving way. We have designed this initially with WhatsApp use cases in mind, like summarizing unread messages or getting writing suggestions, however, the platform can be beneficial to future products beyond these use cases.

Private Processing is built on many layers of hardware and software technologies with a careful understanding of product goals, adherence to security and privacy principles, threat modelling, and substantial technical work to protect against these threats. This paper outlines these areas in detail.

We welcome feedback from the security and research community, and we are committed to addressing bug reports and continuing to harden Private Processing.

What is a Trusted Execution Environment (TEE)?

The Confidential Computing Consortium (CCC) describes a TEE as an environment that assures these properties:

- **Data confidentiality:** Unauthorized entities (e.g., Meta or WhatsApp) cannot view data while they are in use in the TEE. Unauthorized entities could include applications on the host, the host operating system and hypervisor, system administrators, the infrastructure owner, or anyone else with physical access to the hardware
- **Data integrity:** Unauthorized entities cannot add, remove, or alter data while it is in use in the TEE. Attestation allows others to verify the identity and integrity of a TEE instance. When combined with other mechanisms and processes discussed later in this white paper, it enables Meta to provide transparency and verifiability of operations performed.
- **Code integrity:** Unauthorized entities cannot add, remove, or alter code executing in the TEE. Code integrity is an important characteristic of TEEs, which enables us to enforce purpose limitation guarantees by ensuring the valid and correct code is being run in the TEE. Guarantees on the use of a TEE and on the code it runs are provided through attestation (cf. Section 3.1 of the CCC's whitepaper).

This is a general definition and the specific guarantees vary based on the threat models of both hardware vendors and the holistic system's threat model. This paper details our specific system, so that readers can understand in detail the threats and mitigations specific to Private Processing.

Core security & privacy principles

We have designed Private Processing using core principles that surpass traditional cloud-based AI security models. The system is built to ensure that: “Sharing messages with Private Processing does not make them available to Meta, WhatsApp, or anyone else.”

We build this guarantee on the following principles:

User-controlled & transparent

Optionality: Using Meta AI through WhatsApp, including features that use Private Processing, must be optional.

Transparency: We must provide transparency when our features use Private Processing. When Private Processing is used to process a request, it’s clearly displayed to inform users.

User control: Meta is unable to trigger sharing of any data to Private Processing without the user’s action. Additionally, they must be able to prevent messages from being used for AI features like mentioning Meta AI in chats, with the help of WhatsApp’s Advanced Chat Privacy feature.

Private

Encryption: Data must always be encrypted end-to-end between the client and the Private Processing application, so that only a single Private Processing endpoint, and no one in between – including Meta, WhatsApp, or any third-party relay – can decrypt the data.

Confidential processing: Private Processing must be built in such a way that prevents any other system from accessing the user’s data while it is being processed.

Enforceable guarantees: Attempts to modify that confidential processing guarantee must cause the system to fail in such a way that either no data is shared to it (i.e., fail closed) or the attempt to modify the guarantee becomes publicly discoverable via verifiable transparency.

Verifiable transparency: Users and security researchers must be able to audit the behavior of Private Processing to independently verify our privacy and security guarantees.

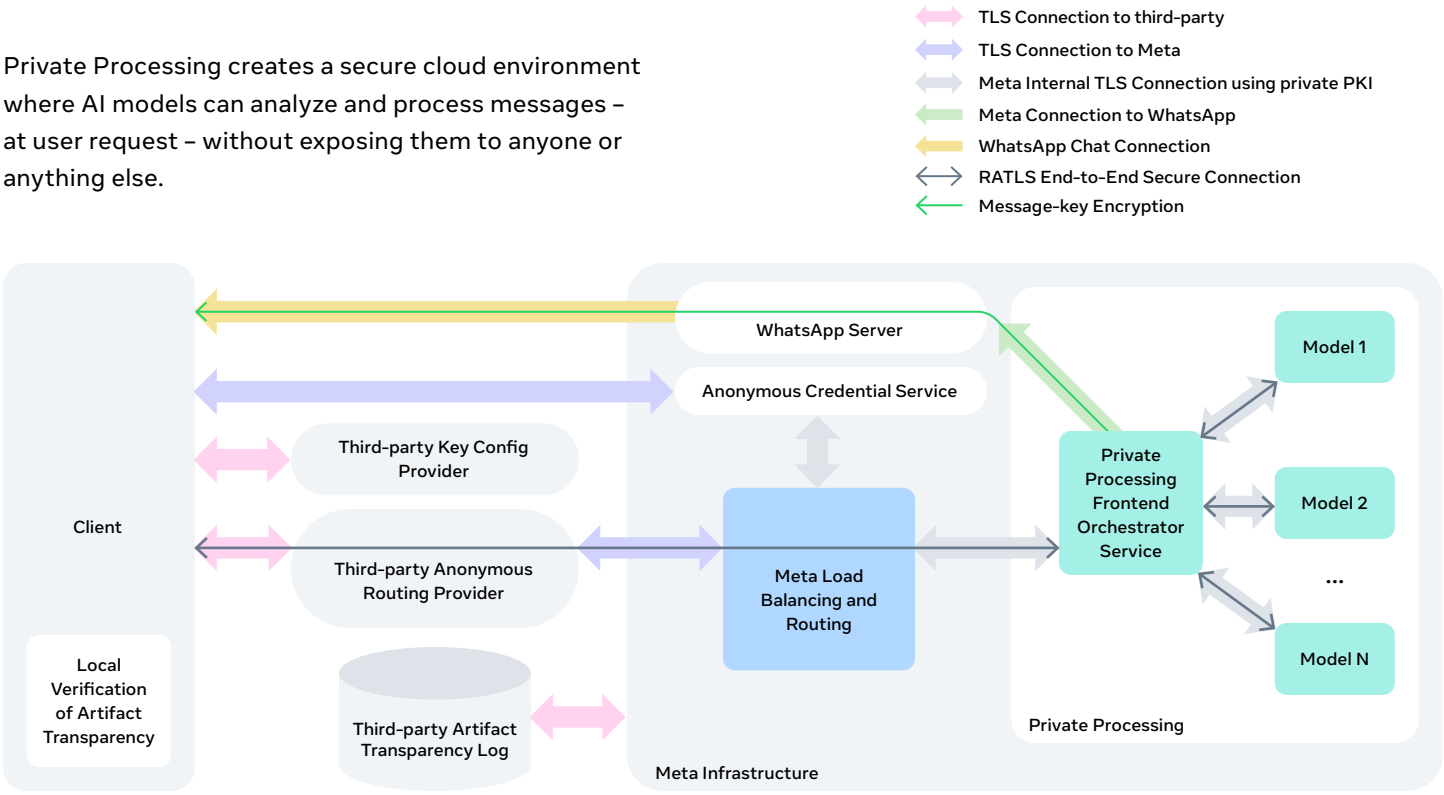
Secure

Non-targetability: An attacker should not be able to target a particular user for compromise without attempting to compromise the entire Private Processing system.

Stateless and forward secure: Private Processing must not retain access to messages once the session is complete, ensuring that an attacker cannot gain access to historical requests or responses.

How Private Processing works

Private Processing creates a secure cloud environment where AI models can analyze and process messages – at user request – without exposing them to anyone or anything else.



The above diagram illustrates how different components of our Private Processing system fit together as of this publication. This is how it works:

Authentication: First, Private Processing obtains anonymous credentials to verify that the future requests are coming from authentic WhatsApp clients.

Third-party routing and load balancing: In addition to these credentials, Private Processing fetches HPKE (hybrid public key encryption) public keys from a third-party CDN (currently Fastly) in order to support Oblivious HTTP (OHTTP).

Wire session establishment: Private Processing establishes an OHTTP connection from the user's device to a Meta gateway via a third-party relay which hides requester IP from Meta and WhatsApp.

Application session establishment: Private Processing establishes a Remote Attestation + Transport Layer Security (RA-TLS) session between the user's device and a TEE chosen by a load balancer while it has no knowledge of the user. The attestation verification step cross-checks the measurements against a third-party log (currently Cloudflare) to ensure that the client only connects to code which satisfies our verifiable transparency guarantee.

Request to Private Processing: After the above session is established, the device makes a request to Private Processing (e.g., message summarization request), that is encrypted between the device and Private Processing with an ephemeral key that Meta and WhatsApp cannot access. In other words, no one except the user's device or the selected TEE can decrypt the request.

Multiple models: We have designed Private Processing in such a way that multiple models can collaborate to produce requested outputs. The Private Processing Orchestrator service brings together the results of the operation of the various models to provide an output back to the client. Connections between these services are encrypted with RA-TLS.

Non-observability: Some models may classify their inputs. To uphold the principle of confidential processing, we have designed Private Processing in a way that Meta or WhatsApp cannot observe information, such as size of the traffic between the Orchestrator service and any classification models, that could enable inference about the output of any classification models used.

Response from Private Processing: The processed results are encrypted with a key that only the users' device(s) and the pre-selected Private Processing server ever have access to, and returned to the user's device either via the OHTTP channel directly or via WhatsApp's chat servers. Private Processing does not retain access to messages after the session is completed.

High-level security technologies

Meta's Private Processing infrastructure includes these security technologies:

- **Confidential compute hardware** - Enables Meta to run software workloads in such a way that nothing other than a specific VM or container can access the information that is being processed.
- **Attested, encrypted communications** - Ensures that data is encrypted and sent only to a valid Private Processing endpoint. Attestation provides assurances on what is done with that data and the security of the workload.
- **Artifact transparency** - Used for binaries, model weights, and more, this prevents Meta (or any other party) from deploying compromised software meant to bypass our guarantees without external discovery. This can be verified using attestation.
- **Secure software** - Meta hardens the software stack running on the Private Processing systems in order to limit the viability of compromise via software vulnerabilities.
- **Anonymous routing** - The system utilizes Anonymous Credentials and Oblivious HTTP (OHTTP) routing through a third-party OHTTP relay proxy in order to ensure user de-identification and prevent Meta from routing a specific user's request to a specific machine. This ensures that an attacker is unable to target a particular user for compromise, without attempting to compromise the entire Private Processing system.
- **Ephemeral data processing in the TEE** - The TEE does not manage any storage, and is unable to store data in a way which gives it durable access to user messages.

Confidential Computing hardware

The hardware foundations of Meta Private Processing ensure user data confidentiality and integrity. In our initial iteration of this system, we rely on AMD and NVIDIA hardware to power Private Processing.

CPUs from AMD and their Confidential Computing technology (AMD SEV-SNP) along with GPUs from NVIDIA Confidential Computing provide protection at the hardware layer with the H100 Tensor Core GPU as well as attestation capabilities to provide confidentiality and integrity for data in use. Both rely on hardware-enforced access restrictions and encryption when data leaves the hardware package. This protects data within a guest Virtual Machine (VM) – also known as a confidential virtual machine (CVM) – including on any associated GPU.

AMD Confidential Computing

AMD has enabled Confidential Computing through its Infinity Guard suite of technologies, which includes AMD Secure Encrypted Virtualization - Secure Nested Paging (SEV-SNP). Built on top of earlier AMD versions (AMD SEV and AMD SEV-ES), SEV-SNP provides strong confidentiality and integrity guarantees for code and data residing inside an AMD SEV-SNP guest Virtual Machine (VM).

For a VM running under SEV-SNP, the AMD EPYC™ CPU with its embedded AMD Secure Processor (ASP) enforces Trusted Computing Base (TCB) attestation and versioning, VM attestation, memory encryption, and page table protections to protect the guest VM from the host, including the hypervisor. The TCB captures all the upgradable firmware components including ASP API versions, CPU microcode, and more.

The AMD EPYC™ with its ASP combined with SEV-SNP protect the system code and data from a malicious host by enforcing these security controls:

- **Trusted Computing Base (TCB) versioning and CPU authenticity.** Each SP has a unique secret fused in a secure location that is combined with the version information of all TCB components to create the Versioned Chip Endorsement Key (VCEK). The SP uses the VCEK to, among other things, sign the attestation report. This process cryptographically proves the current version of all TCB components (patch level), prevents TCB rollbacks, and that it is an authentic AMD platform.
- **VM attestation.** The CPU measures the content of the memory pages where the VM image is located, as well as page metadata to generate the launch digest. This digest is included as part of the attestation report, which is signed by the SP with the VCEK. The attestation report and its signature together provide both integrity guarantees for the VM image, and proof that it is running on a genuine AMD SEV-SNP CPU.
- **Page table protections.** In SEV-SNP, AMD added additional page table protections to further secure a guest VM from a malicious hypervisor by adding a Reverse Map Table and performing additional checks during page table walks. These additional controls prevent a malicious hypervisor from replaying pages, corrupting page content, aliasing pages, and re-mapping pages. This moves the hypervisor fully outside of the trust boundary of the guest, and the main remaining attack vector from a malicious hypervisor is guest Denial of Service. These provide integrity and confidentiality protection.

- **Memory encryption.**[1] Main memory is encrypted with AES in a mode that includes the physical address to prevent blocks from unauthorized relocation. The SP generates and manages the different AES keys, which are unique per both VM and host. These keys are communicated between SP and on-die memory controllers over a secure channel, never exposing them to the host or hypervisor in clear text. The memory encryption ensures confidentiality for the system memory where the VM code and data resides.

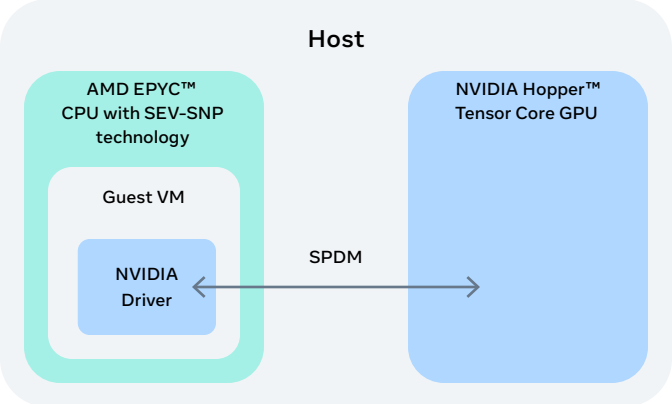
Other Considerations:

- AMD considers certain vulnerability classes out of scope for SEV-SNP[2], with the most prominent being side-channel attacks such as Prime and Probe[3]. These classes of issues can be addressed via compensating controls and, in some cases, software designs and implementations immune to these classes of attacks.

Nvidia Confidential Computing

NVIDIA has built Confidential Computing capabilities[4] to ensure confidentiality and integrity for data and models residing on the NVIDIA Hopper™ Tensor Core GPU. This is accomplished by authenticating the GPU boot flow, attesting the GPU, GPU memory protections, and lockout of debug functionality. The GPU driver within the guest VM also creates a secure channel with the GPU to protect data in transit. These features help safeguard any models and user data residing, even ephemerally, on the GPU for inference purposes.

To ensure data can be securely moved between the guest VM and the GPU, a secure tunnel (SPDM) is established by the NVIDIA driver running in the guest.



Any information passed between the guest VM and the GPU is sent through this secure tunnel.

The security controls and guarantees of the NVIDIA Confidential Computing include:

- **Authenticated and measured boot.** All firmware and microcode on the GPU is signed by NVIDIA, and verified by an on-die wafer (part of the Root of Trust) before it is executed as part of the boot flow. This ensures that only software approved by NVIDIA can be executed on the GPU, preventing altered software from accessing data or models. In addition, the state of the GPU and the software components are measured as part of the boot flow, allowing for later attestation of said state.

[1] https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf and https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

[2] https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

[3] https://ieeexplore.ieee.org/document/7363305

[4] https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.3.0.pdf

- **GPU attestation.** Before the guest VM can trust the GPU and extend the trust boundary to include it, it must verify the attestation of the GPU, software components, and configuration. This process involves multiple steps.
    a. In advance, we fetch "golden measurements" for the deployed firmware and drivers from NVIDIA's endpoint, through the "Reference Integrity Manifest Service API".
    b. As part of the boot process, the host shares the golden measurements with the guest.
    c. The CVM then retrieves the signed attestation report from the GPU, and validates and compares the GPU Attestation report against the golden measurements. We limit the possible VBIOS and NVSwitch versions to the versions Meta deploys in the datacenters by hardcoding these versions within the validation process. If all checks are successful, boot continues. Otherwise, the VM shuts down. This ensures that the NVIDIA GPU is genuine, its cryptographic material can still be trusted, and its software versions and configuration are as expected.
- **Protected GPU memory.** The GPU takes additional steps to protect memory during certain events (Function Level Reset for example) to mitigate cold boot attacks. This helps keep data and models secure while being handled by the GPU.
- **External interface lockout.** Once the GPU is in Confidential Compute (CC) mode, certain external interfaces that could allow a malicious host to access data or leak information about it are locked to prevent access. This includes interfaces such as JTAG or GPU performance counters.
- **Secured CPU / GPU communication.** Once the SEV-SNP guest VM is running, and the NVIDIA driver in the VM has verified the attestation that the NVIDIA GPU is running in CC mode, it creates a secure communication path between the guest VM and the GPU (over PCIe). This communication channel is based on the Security Protocol and Data Model (SPDM)[5] specification, which relies on AES in GCM mode to provide confidentiality and integrity.

Other Considerations:

- **Unencrypted NVRAM:** The HBM of the NVIDIA GPU is not encrypted, opening it up for potential interposing and cold boot attacks. This is mitigated by the SoC packaging, the high bandwidth of the memory interface, and the fact that the GPU locks memory access during certain events such as card resets.
- **Unencrypted NVLink:** For certain use cases that require multiple GPUs for inference, NVLink will not be encrypted on NVIDIA Hopper's platform. This may provide an avenue for interception and decoding of user data or its derivatives. This is mitigated in part by the high speed of NVLink and the current lack of commercially available sniffing hardware. We continue to evaluate solutions as hardware which supports NVLink encryption such as the Blackwell platform becomes available.

[5] https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.3.0.pdf

Attested, encrypted communications

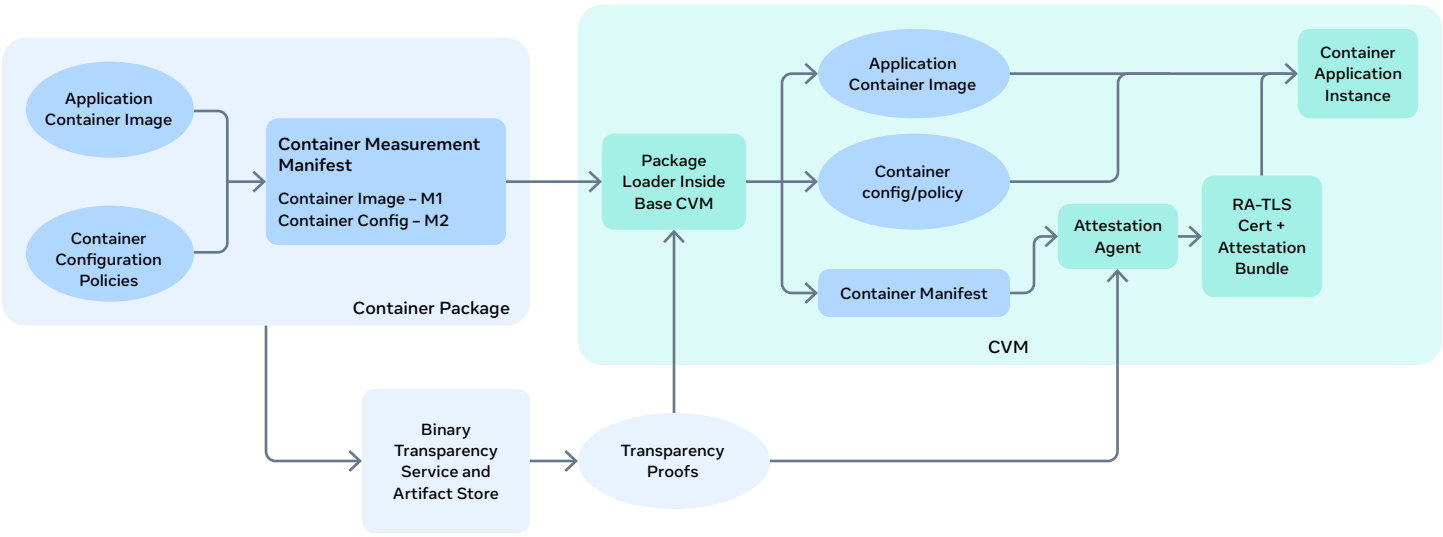
An attestation consists of a cryptographic *signature* over a *measurement*, by which the hardware provides a cryptographic guarantee that a certain piece of measured software is running over a certain input. For Private Processing, the *measurement* captures various aspects of the instance, such as the system microcode, system firmware versions, what optional CPU features are enabled, all the way to the code powering the service application(s).

Measurement and associated attestations are performed throughout the technology stack of Private Processing. When clients connect to Private Processing, or when any one Private Processing node connects to any other node to process user data, such as a frontend orchestrator service to a backend predictor service, we use an attested, encrypted communication channel, which ensures only the CVM or the client sending or receiving the data can decrypt the data.

To provide attested, encrypted communications from client to server or server to server, it is important first to understand the measurement and attestation that starts in the boot process of a particular service, then how that attestation is translated into securing the connection.

Attestation and the boot process

In order to provide strong assurances in attestation, we rely on a chained attestation process where one layer is verified as trusted, which in turn verifies subsequent layers. This starts in the hardware root of trust provided by the CPU manufacturer.



During boot, for our system based on AMD SEV-SNP, we go through multiple layers of these verifications.

1. **Launch Digest:** The AMD Security Processor (SP) calculates the launch digest for attestation, which includes initrd, kernel, firmware, kernel command line.
2. **initrd:** Verified against the hash included as part of the launch digest.
3. **rootfs:** initrd checks the rootfs hash digest against the sha256 hash included in initrd, and includes a "package loader" and "attestation agent".
4. **GPU (if applicable):** CVM attests the GPU, and cross checks the version, and measurement against golden reference measurements from NVidia's "Reference Integrity Manifest Service API".
5. **Container:** The attestation agent, loaded as part of rootfs, verifies the measurements and transparency proofs of all containerized artifacts, and includes those attestation. And because these are dynamically loaded, the agent includes the transparency proofs as part of any attestations.

Package loader

In addition to the attestation through the boot process, the rootfs includes an additional service we call the "Package Loader" which allows loading additional artifacts.

The package loader does the following:

a. **Load package manifest from rootfs:** If dynamically loading packages, the package family manifest file lists packages and their expected measurements for loading and extraction at CVM initialization. The manifest file includes:
   i. Package names and versions
   ii. Expected measurements (i.e. hash digests) of each package

b. **Verify Manifest Digest:** Before CVM startup, the host provides the package, its manifest, and the transparency proof in the shared folder. At initialization, the package loader verifies the package's measurement against the manifest and transparency proofs. This step ensures that the loaded packages match the expected measurements and can be included in the attestation report.

Remote attestation and connection

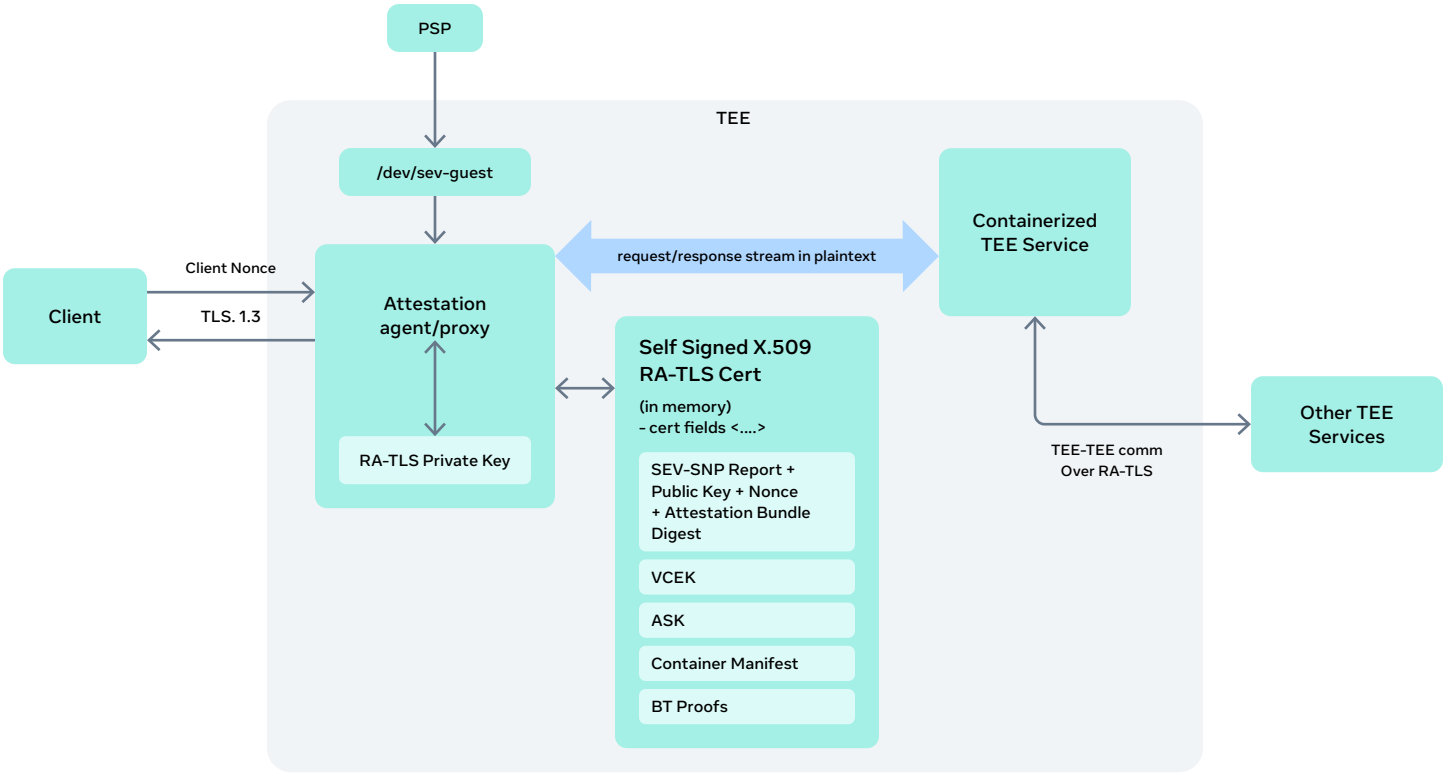
Remote attestation over TLS (RA-TLS) is a protocol that enables a device to securely verify the authenticity and integrity of that device's connection to a TEE. It allows a verifier to ensure that a CVM is genuine, properly configured, and running the expected software stack.

Generating attestation

As part of the rootfs layer, to enable secure communication with other systems (whether other nodes, or clients), the CVM includes an Attestation Agent that generates an RA-TLS certificate for each received request.

The Attestation Agent, upon receiving a request from a TLS client:

1. Generates a public-private keypair (Prime256v1 at time of writing) to be used later to selfsign a certificate.
2. Generates a Preliminary Attestation Bundle to capture the current CVM state, consisting of:
    a. A client provided-nonce extracted from the client HELLO message
    b. TEE certificate chain (VCEK, ASK)
    c. Transparency proofs for TEE measurements and any additional artifacts (e.g., prompts) capturing the complete state of the CVM



3. Generates and signs an SEV-SNP attestation report.
    a. The attestation agent communicates with the Platform Security Processor (PSP) to create a Guest SEV-SNP Attestation Report using the Versioned Chip Endorsement Key (VCEK) specific to the chip. This report includes:
        i. Report data consisting of the digests of:
            1. The public key to prove the key was generated in the CVM, so communications using it will only be available to the CVM.
            2. The Preliminary Attestation Bundle
        ii. Hardware security versions and platform configuration (e.g.. TCB values)
        iii. Measurements of software components, such as kernel, initrd, and firmware
        iv. PSP-generated signature over the measurements and report data
4. Finalizes the Attestation Bundle by incorporating the Guest SEV-SNP Attestation Report into the Preliminary Attestation Bundle.
5. Generates a Self-Signed X.509 Certificate.
    a. Create Self-Signed Certificate: The agent creates a self-signed X.509 certificate, signed with the private key. The certificate includes:
        i. The public key.
        ii. Subject field set to CVM's identity.
        iii. Extensions containing the Finalized Attestation Bundle.
6. Returns the newly generated RA-TLS certificate to the client.

Establishing an attested connection

At a high level, clients use the HTTP POST method to request to establish a connection with the target Private Processing service. This request goes through multiple layers of load balancers and proxy servers to provide scalability, reliability, and target node selection of the connection.

The Private Processing target node is selected from a list of available TEE nodes based on region, cluster, and load. Once a Private Processing node is selected, the RA-TLS request is forwarded to the attestation agent inside the target service to establish a TLS connection between the client and the target service (i.e. TEE Orchestrator service running inside a TEE), which responds with an RA-TLS certificate.

Verifying attestation

Before clients or other TEEs create any connection to a TEE, they first need to verify the RA-TLS certificate they received using a service-specific policy and extract the data from the corresponding Object Identifier (OID) 1.3.6.1.4.1.40981.2.2.20 that uniquely identifies the extension in the certificate extensions.

Service-specific policies are defined based on the topology of the request flow and allows identifying each TEE based on the transparency namespace they belong to. For instance, the clients only trust the Private Processing frontend's namespace policy for the CVM measurement proofs and reject any connections that provide a transparency proof on a different namespace. Furthermore, each policy defines the transparent artifacts and associated namespace that must be included in the RA-TLS certificate.

For any RA-TLS connections, following assertions must be verified:

1. The TLS certificate is self-signed, valid and properly constructed, includes an attestation report, expected transparency proofs for that service, has a public key *PK*, and the signature verifies with the public key *PK*.
2. The cryptographic hash of *PK* is embedded in the AMD SEV-SNP attestation report as additional data, along with the attestation bundle that contains the measurements of the CVM and its artifacts.
3. The request's nonce matches the one included in the attestation report.
4. All expected transparency proofs are correctly verified using the service-specific policy and not revoked.
    a. As part of the RA-TLS cert, there is also a revocation list that is transparent and belongs to a specific namespace.

5. Attestation report is valid and signed by VCEK – a chip-specific private key for AMD SEV-SNP, and the Guest and the Platform satisfied trusted configurations as enforced in the policy.
6. VCEK certificate chain is valid with a path to the root: AMD SEV Signing Key (ASK) and AMD Root Key (ARK). The attestation bundle includes the VCEK and ASK, and ASK is not revoked.
7. Verification of Attestation Report
    a. Verify Report Signature: The client checks the SEV-SNP report's signature using the AMD certificate chain (VCEK, ASK, ARK), ensuring the report was generated by the PSP.
    b. Match Public Key: The client ensures the public key in the report matches the RA-TLS public key, binding the report to the session.

Client trust anchors

Clients hardcode:

- ARK (AMD Root Key) – for verifying attestations and ensuring the TEE is genuine AMD.
- Certificate Revocation Lists (CRLs) for the ASK certificates.
- Minimum TCB (Trusted Compute Base) version numbers.
- Trusted configurations including Guest and Platform Policy.
- Artifact transparency provider's public signing-key – for ensuring the TEEs are published transparently.

Policy framework

During attestation verification, the connecting client performs additional, hardcoded checks as part of a verification policy, including:

- **Attestation policy for AMD SEV-SNP:** Defines the requirements for AMD SEV-SNP TEEs including: guest policy, which security features of the CPU must be enabled/disabled, and minimum TCB version.
- **Transparency Policy:** Defines which keys are trusted for transparency proofs as well as their freshness requirements. Freshness requirements are based on the artifact type. Each artifact has their own namespace to distinguish the requirements between them. Similarly, namespaces are also used to separate services (e.g., Orchestrator has a namespace as well as the Predictor). This separation allows Meta to define policies that define which TEEs are allowed to connect to other TEE services.
- **Revocation Policy:** Relies on the Transparency Policy to verify a provided Revocation List and check for artifact revocation.
- **Trusted Time:** Since TEEs do not have a trusted time source, policies can include a current time field that can be used by any connecting devices to ensure the freshness checks discussed above are accurate.

Policy flow and use of "most restrictive policy"

In each request, the client includes a policy containing:

- The timestamp.
- TCB values for AMD SEV-SNP instances.
- Trusted artifact transparency provider's public signing-key(s).

Upon receiving the request, the Private Processing service combines these values with its local policy to ensure that:

1. It has a trusted time source for time checks such as transparency proof expiration.
2. Its internal TCB policy is up to date.
3. All trusted signing-keys for artifact transparency are also trusted by the requestor.

This combination is based on "most restrictive policy" where the Private Processing service chooses the max timestamp and TCB values of the two that was sent by the client and that was sourced on the TEE itself.

The final policy is then used to verify the RA-TLS certs and attestation bundles for the TEEs that the Orchestrator communicates with.

Stateless and forward secure

Stateless

Orchestrator binaries are stateless by design with no database access. Orchestrator does not write request payload to any storage and only uses it to fulfill the current request. As such, each request made to the orchestrator is treated completely independently from prior requests. To enable multi-turn conversational experience, conversational history is sent to the orchestrator from the client with every request.

Predictor for running LLM inference is configured as a stateless HTTP service. Specifically, it turns off persistent KV Cache in LLM inference so that all computation and cache state for a request has the same lifecycle as the request. Therefore, each request in Predictor is completely independent of another request.

CVM lifecycle

In Meta Private Processing deployments, AMD SEV-SNP CVMs play an important role in ensuring the confidentiality and integrity of sensitive data. Components inside the CVMs are designed to be stateless and do not store any persistent data. This statelessness is also reflected in the containerized services running within the CVM. Dynamically loaded artifacts like models, cache, prompts, service containers, platform certs are loaded and instantiated only after verifying against their transparency proofs. This verification process ensures that the artifacts have not been tampered with and can be proven genuine and valid.

After successful verification of the artifacts, their measurements and transparency proofs are incorporated into a comprehensive attestation bundle. The attestation report, public-private keypair, and the self-signed X.509 certificate is generated upon client request for RA-TLS session freshness. By re-establishing the CVM's trustworthiness with each session, this process ensures remote clients can maintain trust on the CVMs at all times.

To further enhance trust on the CVMs, revocation lists for all artifacts are refreshed daily, and the attestation agent service running inside the CVM ensures that it always uses the latest revocation list in the generated attestation bundle.

Meta's infrastructure for updating services arranges for smooth transfers between older and newer versions of the software by launching new CVMs with the new software, redirecting new traffic to the new CVMs, and then shutting down the old CVMs. We regularly reboot all CVMs in Private Processing to ensure freshness.

Artifact transparency

Meta has worked with a third-party transparency log provider – Cloudflare – to build the infrastructure to ensure that security-critical artifacts are recorded on a third-party, append-only log[6]. We are committed to releasing binaries corresponding to entries on these append-only logs. This will be done on an ongoing basis.

The third-party provider ensures that artifacts submitted are unique, timestamped, and use incremental epochs. In addition, the provider creates a signature over the digest to prove to any interested party that they logged this binary. Interested parties can connect to Cloudflare's Key Transparency Auditor APIs directly to fetch the latest epochs and their submission times.

Artifact types

Meta utilizes both Auditable and Private artifacts. Auditable artifacts will be released to eligible researchers. Researchers will be able to access artifacts, inspect them, and reproduce their role in close-to-production environments.

A minimal set of remaining artifacts are kept private and are maintained internally. However, this does not include any code, binaries, or executables.

Five types of artifacts are used:

1. Binaries (auditable)
2. Revocation Lists (auditable)
3. Models (private)
4. System Prompts (private)
5. Authorized host identities (auditable)

Meta will make auditable artifacts accessible as part of our release process when updating the Private Processing service. In some cases (like third-party security fixes bundled within these artifacts and pre-disclosed to us under NDA and coordinated disclosure agreements), Meta will make artifacts available in a coordinated disclosure with relevant vendors, typically within a 90 day timeframe.

Each artifact is published to its own namespace. This allows independent component releases, so researchers can focus on what has changed if they so choose. As of this publication, these are the namespaces (subject to change):

- prod.pc.orchestrator.cvm → base CVM image of orchestrator
- prod.pc.orchestrator.container → orchestrator container
- prod.pc.revocation_list → revocation list
- prod.pc.orchestrator.prompt → prompts
- prod.pc.vcek_cert → VCEK certificate transparency
- prod.pc.predictor.cvm → base CVM image for predictor
- prod.pc.predictor.container → predictor container
- prod.pc.predictor.llm → LLM model
- prod.pc.predictor.cache → LLM runtime kernel cache
- prod.pc.orchestrator.integrity_rules → thresholds for integrity checks
- prod.pc.orchestrator.soma_blobs → integrity database
- prod.pc.orchestrator.trending_topics → list of trending topics

Release process

When pushing a new artifact to production, Meta's release process includes these steps:

1. Archive the artifact for researchers.
2. Submit the artifact digest to a third-party transparency log provider for addition to the relevant transparency log.
3. Receive a signature indicating the log provider has added the given digest to the transparency log.
4. Store that digest alongside the artifact.
5. Release the artifact to researchers, after it is no longer subject to any coordinated disclosure requirements. See the Vulnerability management section for more information.

[6] **Note:** this is not a Merkle-tree based transparency *ledger* but a simple log maintained by a third-party.

Artifact expiration and revocation

To ensure the system does not keep old artifacts in memory, an expiration policy is in place for each artifact type. The expiration is checked at the time of attestation bundle verification against the signature timestamp.

Meta has implemented policies for different namespaces that are tied to the expected release cadence for that artifact. For example, the machine learning models will likely be updated on an ad-hoc cadence, so expiration is on the order of months. For the CVM base image, releases are on a weekly cadence to pick up the latest kernel updates, so expiration is on the order of weeks.

There is also a revocation mechanism, in case of an accidental release. A single revocation list is maintained for all artifacts. The revocation list is also submitted to the provider for signature periodically. The revocation list and its signature is included as part of the attestation bundle, where the clients verify the exclusion of the artifacts from the list. The list is published every 3 hours, and each signature expires every 24 hours.

Artifact validation

In conclusion, an artifact is considered valid if all of the following criteria are met:

1. It belongs to a trusted namespace.
2. Its timestamp shows it is not expired, as defined for the namespace in question, compared against a trusted time source.
3. It has a valid transparency proof, meaning it is signed with a trusted key as defined in the Transparency Policy, and the signature is valid.
4. The verifier has access to a valid revocation list as verified using the Revocation Policy.
5. The artifact has not been revoked.

Secure software

Securing software is an important part of providing meaningful privacy guarantees in Private Processing. This section focuses on how we have hardened the software running within the TEE in order to ensure security of Private Processing. In a Trusted Execution Environment (TEE), the hardware security controls act as a key security barrier in combination with limited software running on the CVM, and we incorporated additional security practices to minimize the impact of or reduce the likelihood of compromise of the CVM itself.

Software layers can be hardened by addressing known vulnerabilities, reducing the number of attack surfaces, using secure coding practices, and implementing access controls. We have done work from the CVM boot process through to the Application(s) powering the service themselves to provide these additional defenses.

Virtual machine hardening

The Guest Virtual Machine consists of three major components: the boot flow, the guest kernel, and the guest OS, all of which support the workload application itself. Each component has gone through its own unique steps to limit their attack surface and harden their implementation. Hardening of the workload application will be covered in a later section.

On the virtual machine boot process, our system today leverages a minimized stage0 from Google Oak with minor modifications to allow kernel, initrd, cmdline launch digest ingestion from qemu, and to add checks against ACPI table corruption.

The guest kernel is an enlightened (patched to support SEV-SNP which has been part of the mainline since kernel version 5.19[7]) standard Linux OS version 6.12+ kernel[8] from the stable tree, which is updated on an ongoing basis. The kernel configuration has been modified in accordance with the recommendations from the Linux Kernel Self-Protection Project[9], and additional loaded kernel modules are minimized to only the GPU drivers.

The kernel is compiled with best-practice security flags, such as -fstack-protector, enabled to further improve kernel security and resilience.

The guest OS is based on Linux (CentOS 10 as of writing), where the base OS and the application code have been separated into two distinct parts, with the application code running within a container. While both parts are measured, it allows for separated updates between the base OS and the application code. In the base OS image, all non-essential services have been removed, including network interfaces. Instead, network traffic enters and leaves the guest VM through a small number of controlled input and output paths via a VSOCK interface.

This results in a minimal image with only the Linux kernel, the attestation agent, VSOCK bridge, and select metric collection (see the Debuggability and operations section for more details).

Upon boot, the Guest Virtual Machine establishes two communication channels with hypervisor to receive a minimally required set of files:

- **Boot stage only channel:** Used to transfer files that do not require update throughout the lifetime of the Virtual Machine and closed before any client's request is received (e.g. transparency proofs for Virtual Machine itself and included BT artifacts).
- **TEE Secure Analysis (TSA):** This channel stays open during the life of the Virtual Machine and used to transfer any files that require a "refresh" (e.g. revocation list with transparency proof). See Remote diagnostics section.

Each file is validated by either consumer of the file (e.g. Binary Transparency artifact verification service would verify and copy corresponding transparency proofs), or the TSA, before being copied into the Virtual Machine memory.

[7] https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.19
[8] https://www.kernel.org/
[9] https://kspp.github.io/Recommended_Settings.html

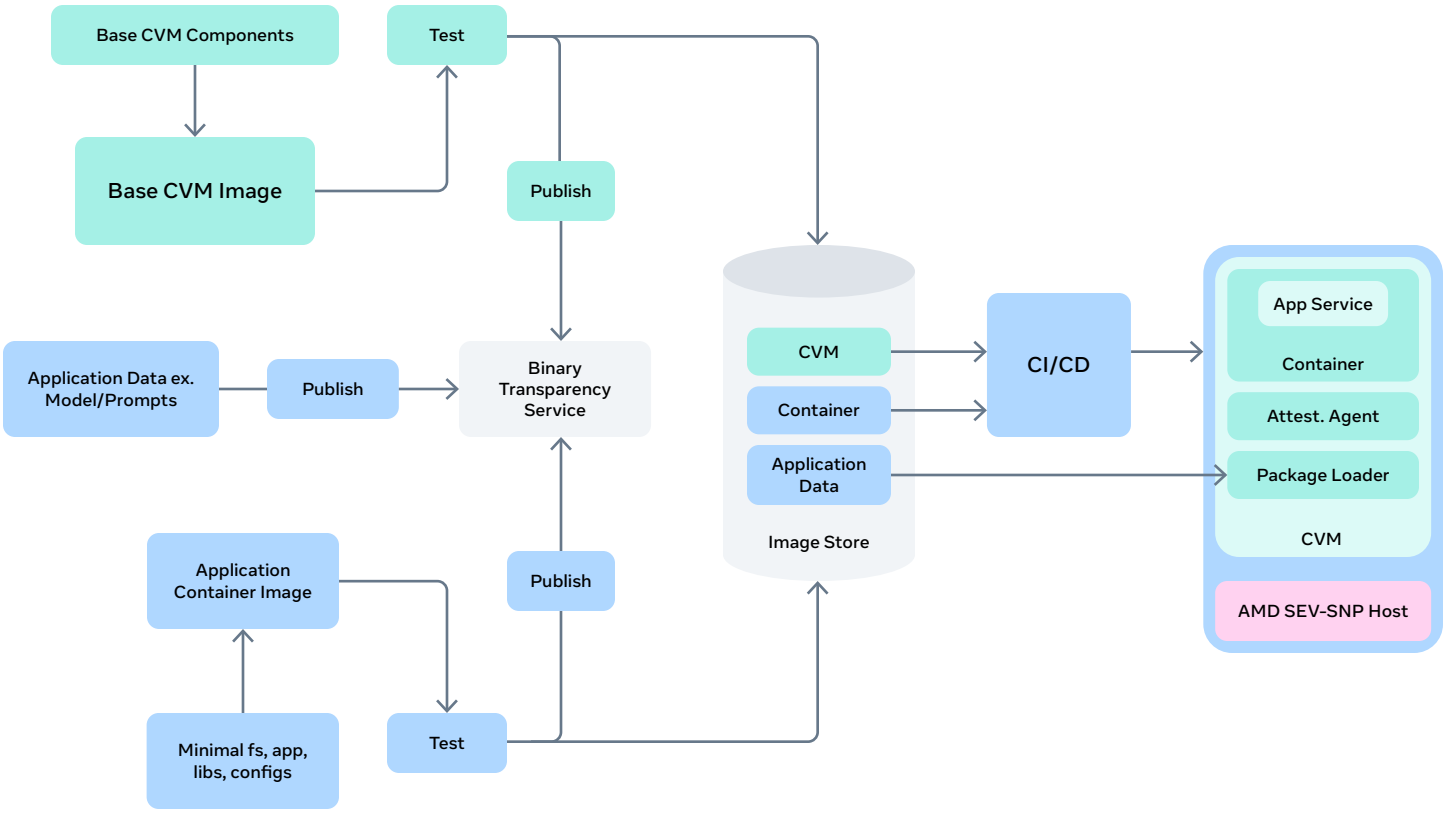
Application containerization

The services running on Private Processing are built as small container units and provisioned separately inside the base CVM image in a LXC-based isolated environment.

LXC is a framework in Linux operating systems that leverages the process isolation capabilities of the Linux kernel to establish and execute application containers. These containers function as independent, isolated and minimal Linux OS environments, all managed by a single kernel. LXC utilizes Linux kernel security features such as namespaces, apparmor, and cgroups to achieve isolation.

Containerizing the TEE application services provides tighter restrictions and control over potential data exfiltration paths and minimizes the impact of security events caused by exploiting application vulnerabilities, helping prevent access to sensitive data in the CVM.

It is important that the TEE applications and the base CVM image can be built, tested, published and audited independently and still be bound in a single AMD SEV-SNP attestation report. This means that transparency proofs can be verified in a single flow to gain trust on the entire stack.



The independent build and deployment flow is very similar to the CVM image and both the CVM image and application container image are appropriately tagged, measured, and their digest published in an external binary transparency log for artifact inclusion and attestation purposes.

Tagged images are then picked up by CI/CD systems and deployed as generic base CVMs on AMD SEV-SNP Hosts. The containers are injected at runtime via shared folders for the CVM's trusted package loader to copy, measure, verify and instantiate the containers containing the services.

While different versions of a container image may run on the same CVM image version, we do not allow changes during the CVM's lifetime. During runtime, the app container image is loaded and instantiated just once, at boot. To load a different version of the app container image, a CVM reset would be necessary. This non-configurability avoids time-of-check-to-time-of-use concerns in the workload's attestation measurement.

Application hardening

Due to the application execution within a container, any unknown application security issues will be limited in impact by the limitations on communication beyond the VM. To provide defense in depth, we have taken additional steps to harden the application itself to further minimize the impact of any application-level security issue.

For inference, the application relies on vLLM[10] together with the safetensor format, which avoids code execution risks when loading models and weights, e.g. by avoiding insecure Python pickles. Additional work has been done to remove unnecessary dependencies from vLLM to keep the attack surface minimal. As a last step, it has been compiled with best-practice security flags enabled to further improve binary security and resilience. The Private Processing front-end application relies on similar security hardening as vLLM, but is implemented using Llama-Stack[11].

[10] https://github.com/vllm-project/vllm
[11] https://github.com/meta-llama/llama-stack/tree/main

Anonymous routing

In order to achieve the non-targetability principle, we implemented anonymous routing, where flexible routing and load balancing decisions are made without user identifiable information such that any attempt to compromise a single user's data requires a scalable and repeatable attack to compromise the entire system.

We accomplish this by leveraging the Anonymous Credentials protocol in order to authenticate the user request while de-identifying the end user. Additionally, user requests are routed through a third-party Oblivious HTTP relay proxy to hide the user's IP Address from Meta infrastructure.

Anonymous Credentials Service (ACS)

In order to authenticate a user without identifying them, while ensuring that they are authorized to access this system and rate limit access to our infrastructure, we leverage Anonymous Credentials.

Meta relies on the open source Anonymous Credential Service (ACS) for the user to obtain a credential (ACS token) that allows them to communicate with the CVM service without being identified. This ensures that we route the user request without knowing who the user is, helping achieve non-targetability. Using credentials also ensures that the Private Processing service only accepts traffic from authorized users.

Oblivious HTTP (OHTTP)

The request is proxied through an Oblivious HTTP provider to Meta's OHTTP Gateway. We use the draft, chunked Oblivious HTTP protocol RFC to help decrease the number of round trips needed to establish a connection, send a request, and receive a response. We have worked with Fastly to provide this infrastructure.

We establish an RA-TLS connection over the OHTTP chunked messages, which is securely terminated only inside the CVM. As part of the inner HTTP request, we pass an ACS token to validate and authenticate the client.

The connection process has five steps:

1. The client creates a TLS session wrapped in an HTTP request (Inner HTTP) with an ACS token in the headers to connect to Meta infrastructure. In the host header field, we set it to indicate this is a request to Private Processing.
2. The client encrypts the InnerHTTP using OHTTP HPKE and creates an Oblivious HTTP request (OuterHTTP) and sends the request to a third-party OHTTP relay. We use HTTP/2.
3. The third-party OHTTP relay strips out IP information and then relays the request to Meta's endpoint, the OHTTP gateway.
4. Meta's OHTTP gateway decrypts the HPKE requests, validates the ACS token, and selects a CVM node to which it relays the body of the request.
5. The CVM accepts the TLS message and continues with the handshake.

With the chunked protocol, this process continues in such a way that the client and CVM both send and receive body chunks, where each chunk is a message from client or CVM.

Key configurations

ACS relies on a key configuration which allows authentication and verification of anonymous credentials. Similarly, OHTTP relies on a key configuration to enable the client and server to encapsulate or decapsulate the inner HTTP request. We rely on a third-party (currently Fastly) to distribute these configurations to prevent Meta from being able to target specific users in such a way that would allow de-anonymization of the requests.

We deploy an ACS and OHTTP Key Config to a third-party CDN service (currently Fastly). Clients connect regularly to the relay proxy in order to fetch the latest config. This allows ACS and OHTTP to rotate private keys without disrupting the client requests. This key config is signed with a Meta-owned key, and that signing key is hardcoded on the client to verify that the key config was distributed by Meta.

Specifically for ACS, after fetching the key config, the client connects to Meta endpoints to fetch a new ACS credential which can be used a finite number of times.

Additional security operations

Vulnerability management

We implement a robust vulnerability management strategy for all software running within our confidential virtual machines (VMs). Our approach ensures that all software within the VMs maintains the highest security standards, safeguarding sensitive data and operations from potential threats.

This involves continuously detecting, monitoring and assessing third-party software dependencies against multiple feeds to identify and allow us to quickly patch critical vulnerabilities. For vulnerabilities without available patches, we conduct detailed evaluations to understand their impact on our privacy and security guarantees. If they weaken our guarantees, and fixes are either not possible or not imminently available (e.g., by third party hardware vendors), we will evaluate disabling Private Processing and associated features it supports.

In addition to our automated vulnerability management tooling, our relationships with our hardware and software vendors ensure that we are informed about newly detected vulnerabilities so we can quickly address them. If Meta, as part of a coordinated disclosure process, receives security fixes under an embargo, we will apply the fixes as soon as possible and will follow the responsible disclosure terms under the embargo for releasing the fixes in source or binary form, as part of our verifiable transparency efforts. Commonly across our industry, these terms are typically up to 90 days (but can extend longer).

Host monitoring

We monitor hosts for indications of potential compromise through the collection and monitoring of system metrics and information. It allows us to maintain continual awareness of potential attempts to tamper with CVMs, without the need to collect additional telemetry from them. It minimizes the risk that CVMs accidentally export any sensitive information. Monitoring generates telemetry including:

- **Process Activity:** Execution of new processes
- **Kernel Activity:** Module Loads and Unloads
- **Hardware Modification:** PCI device activity
- **Network Activity:** Connection open, DNS requests
- **SSH:** SSH Logins/Logouts

Debuggability & operations

Debuggability and operations are crucial for running reliable services at scale. Given its nature, Private Processing requires a unique approach, since we have removed standard system administration capabilities to maximize privacy and security.

In Private Processing environments, general-purpose access to systems is restricted. Interactive access, such as SSH and console access to Confidential Virtual Machines (CVMs), is disabled in production systems. This necessitates alternative methods for observing application and service behaviors, diagnosing issues, and collecting performance and reliability metrics.

Logging

All logging solutions adhere to the following principles:

- Logs must only exit the CVM through explicit, controlled channels.
- Logs must not contain private data. Only logs that have been evaluated and approved within the context of Private Processing are permitted to be exported from the CVM.
- Any log filtering must be attested as part of the CVM image.
- Changes to filtering rules require code modifications and an associated review process.
- No operating system logs are allowed to be exported from the CVM. Necessary OS-level debug information is provided through remote diagnostics tooling.

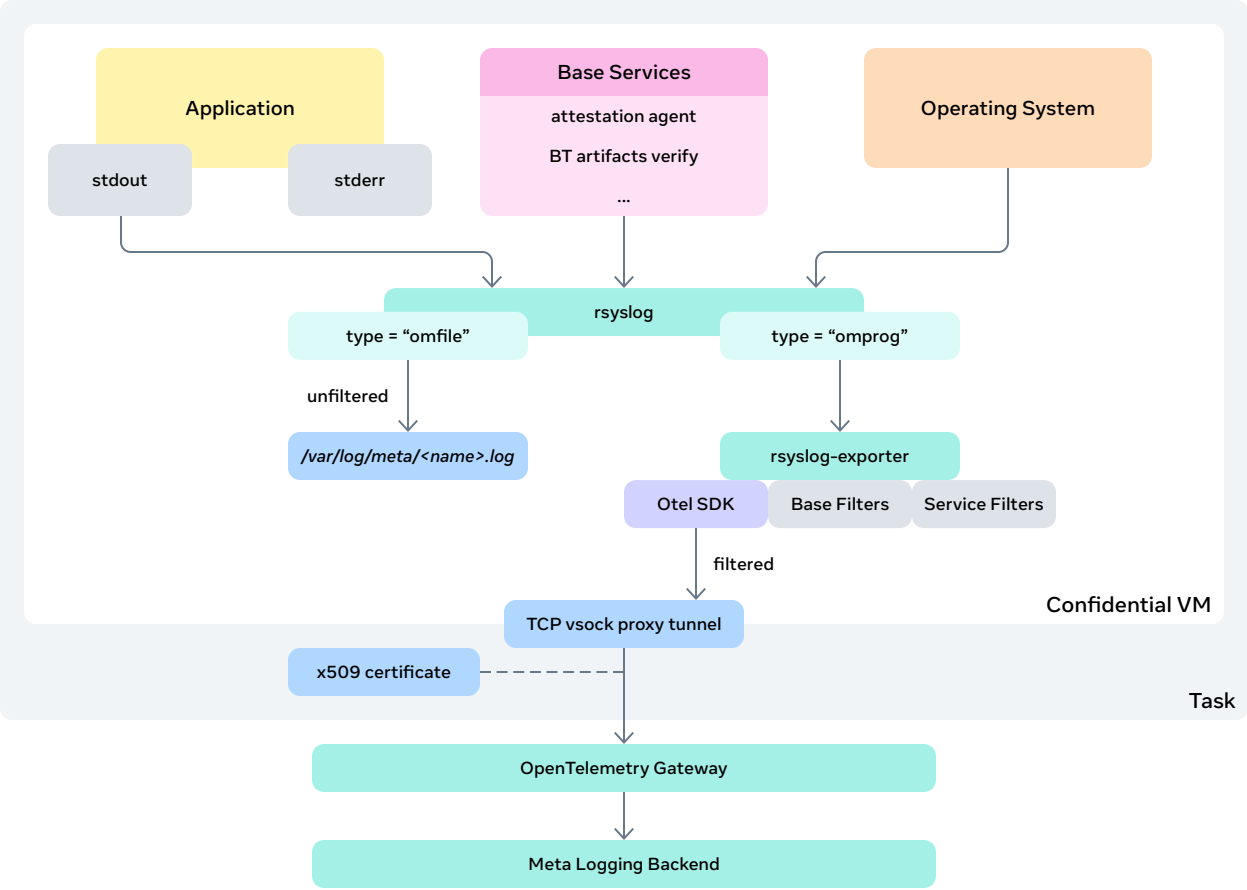
We facilitate the export of three types of logs, each requiring distinct solutions:

- Unstructured Logs
- Structured Logs
- Numerical Metrics

Unstructured logs

Unstructured logs are text-based logs generated by applications and various operating system components without a predefined structure. A common example includes application logs directed to stdout/stderr or log statements written to a file.

This section illustrates the process of handling unstructured logs within the Trusted Execution Environment (TEE) and their export to an external Meta Logging backend.



Setup phase:

The log design incorporates a mechanism for filtering logs during normal CVM operation to maintain the TEE's privacy properties. However, during the “setup phase” – from CVM launch until the TLS keypair is generated and linked to the attestation report – logs can be exported without restriction. This phase is critical to prevent key material leakage, and once completed, the CVM cannot revert to this phase.

Service management and loggin configuration:

Each service or application within a Confidential Virtual Machine (CVM) is managed by `systemd` and configured to log using `rsyslog`. A unique `SyslogIdentifier` is specified in the service configuration to ensure distinct identification.

Log processing and filtering:

For each `SyslogIdentifier`, an `rsyslog` configuration file defines the log processing and filtering rules. Two actions are configured:

- omfile: This standard module logs to a file without applying any filters. The log files remain within the TEE and are not accessible externally.
- omprog: This module allows the execution of an arbitrary binary (`rsyslog-exporter`) during logging. It serves as the integration point for processing log statements, applying filtering rules, and exporting them to an external system.

Filtering rules:

- **Base filters:** Applied to services provided by TEE Infrastructure, these rules are consistent across all TEE use cases and managed centrally.
- **Service filters:** TEE application owners must analyze their service logs to compile a list of allowed message patterns that do not contain user data and are useful for troubleshooting.
- Owners are responsible for reviewing these rules regularly.
- Filters operate on an “allowlist” basis, meaning only explicitly allowed messages are exported. Filtering rules are embedded into the CVM image during build time and are part of the attestation process.

Log exportation:

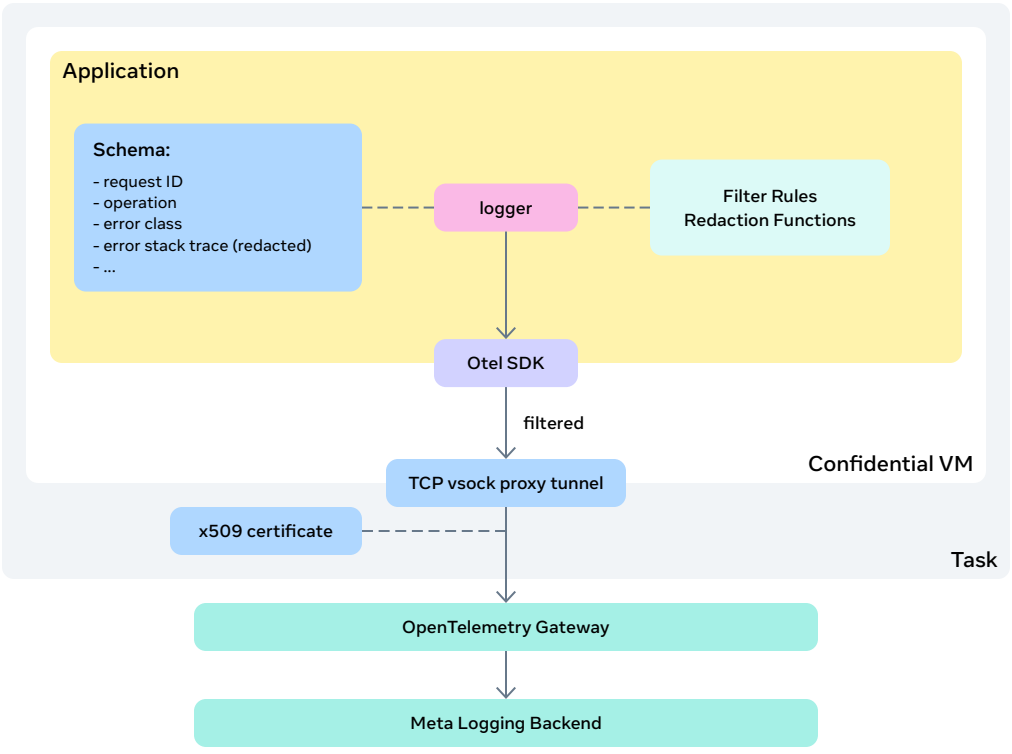
- The `rsyslog-exporter` utilizes the OpenTelemetry SDK to export processed log statements to an OpenTelemetry gateway via a TCP↔VSOCK channel.
- Exported logs are mapped to a service instance (Task) and Meta Logging backend category, with this information included in the OpenTelemetry payload sent to the gateway.
- The OpenTelemetry gateway integrates with the Meta telemetry backend, translating log statements into the appropriate format for the backend system.

Structured logs

Structured logs are logs that adhere to a predefined schema, tailored to meet specific application requirements. At Meta, structured logs are typically associated with Scuba and are implemented using an internal logging framework. Since this framework is not open-sourced, an alternative approach is employed for Trusted Execution Environments (TEEs).

This diagram provides a high-level overview of how structured logs are exported from the CVM to the Meta logging backend using the OpenTelemetry framework. The specific log structure and filtering rules are determined on a per-application basis.

- The application code uses OpenTelemetry SDK to configure Scuba logging via OpenTelemetry gateway.
- The schema is defined based on the application needs and owners have full control over it. The log attributes are transformed into a Scuba column during injection.
- OpenTelemetry logger requires information about the destination table (set via *"fb.scuba.table"* resource attribute). Attributes that convey information about Task are optional, but help to isolate the instance of the service and can be useful during troubleshooting.
- The communication channel between OpenTelemetry SDK and OpenTelemetry gateway is identical to the one used for unstructured logs.
- The filtering is applied within the application code on a per log attribute (or field) basis before OpenTelemetry payload is constructed. Since Meta has full control over logging schema and the logging process, the main goal of the filtering is to correctly redact error traces to ensure no sensitive data is revealed.



Remote diagnostics

Remote diagnostics enable operators to gather detailed information from the Confidential Virtual Machine (CVM) without requiring interactive access, such as SSH or console access.

Core design principles

The remote diagnostics tooling is developed with these principles:

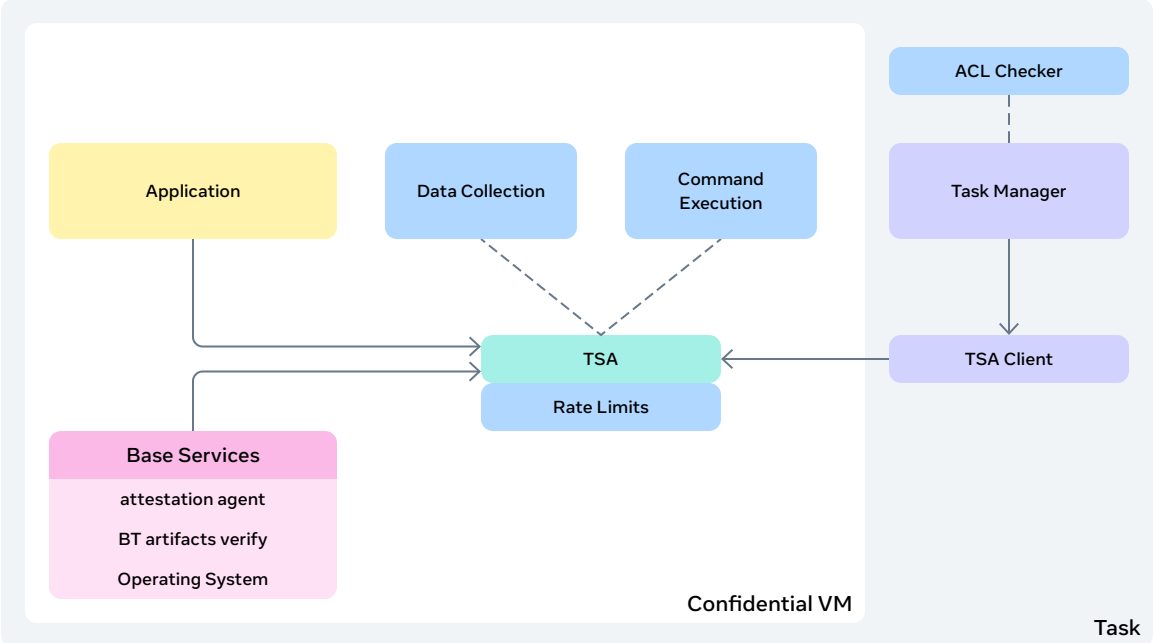
- **Verifiable interface:** Provides a deterministic interface with access limited to a small, restricted set of operational metrics and data. This access can be transparently verified by researchers to ensure integrity and security.
- **Rate limiting:** Implements rate limiting on certain operations to minimize the risk of user data exposure and potential side-channel attacks.
- **Privacy-preserving crash reporting:** Ensures that crash reporting is conducted with privacy-preserving principles. For example, unresolvable stack frames are excluded from reports to prevent exposure of user data, such as stack pointers from buffer overruns.
- **Transparency:** Designed without dependencies on Meta infrastructure to enhance transparency and accountability.

Remote diagnostics are facilitated by our “TEE Secure Analysis” (TSA) service, which provides minimal, safe, and deterministic debugging capabilities.

The TSA service operates within the CVM, and its binary is attested as part of the CVM image. This attestation ensures public transparency and holds Meta accountable for any commands executed within the TEE environment.

The TSA client is deployed outside the TEE and communicates with the TSA service over a VSOCK channel.

The TSA interface is not restricted to local access within the Task boundaries. Instead, broader access is controlled by the Task Manager, which manages access control lists (ACLs) to expose the diagnostic interface to remote operators.



Metrics

"Metrics" refer to information provided in the form of counters emitted by applications or collectors. FB303 counters are a typical example of metrics. The CVM exports metrics to the Meta backend, where they can be used for further analysis.

Export for service counters (FB303)

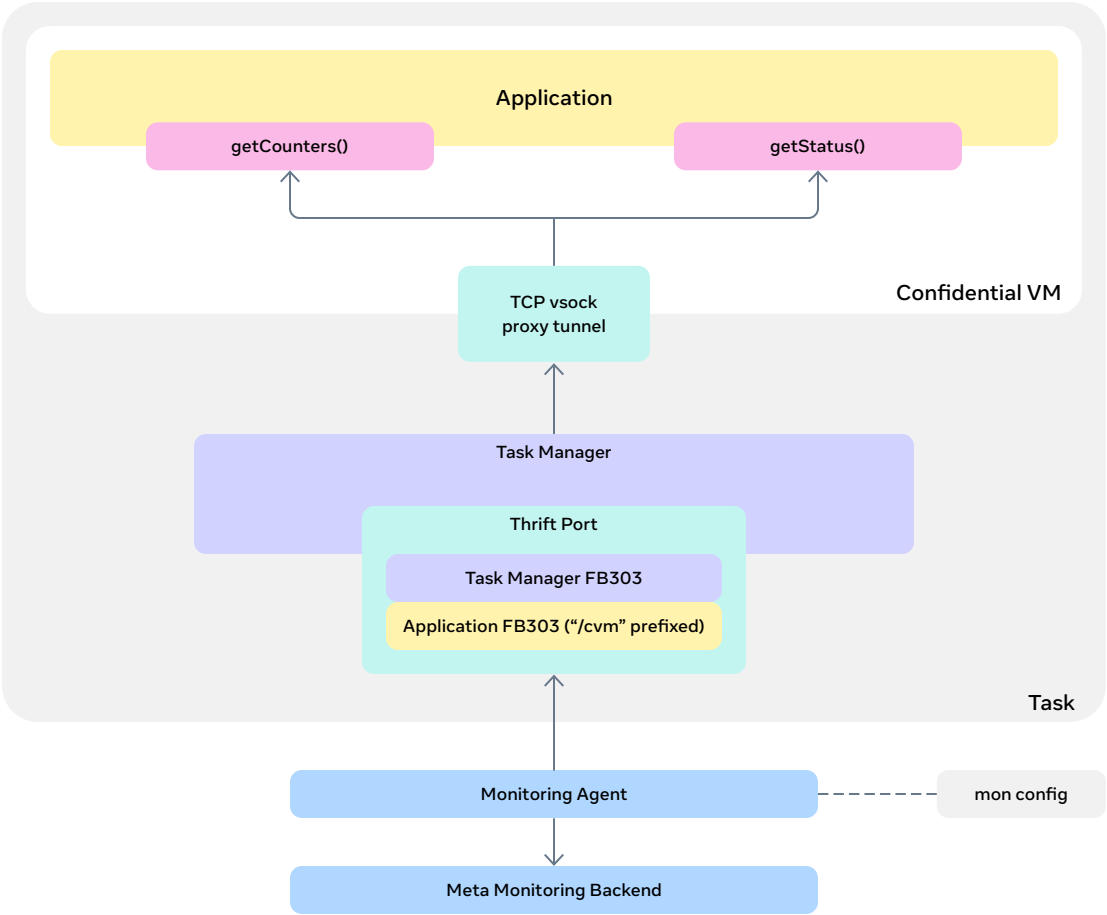
Service-specific counters are explicitly defined by the application owner and contain only numeric values. We consider it safe to export all available FB303 counters without further filtering inside TEE as they don't contain user data.

This diagram represents how application FB303 counters are exported to the Meta monitoring backend:

The application provides FB303-compatible `getCounters()` interface, which is periodically polled by Task Manager running outside of TEE via the TCP-VSOCK proxy tunnel. Received application counters are extended as Task Manager counters, prefixed by `/cvm`, and further polled by the Monitoring agent running on the host. Afterwards, the service owner can create a monitoring config to configure which of the counters should be exported to the Meta backend.

For services that communicate directly with the "external" clients (e.g. mobile devices), the counters are exposed on the interface different from the dataplane. Otherwise, using the same port is acceptable.

The service owners go through a rigorous review of all exported counters to ensure these uphold Private Processing's privacy commitments.



Export for operating system (OS) counters

Information about system resource utilization is critical for understanding service hardware requirements. To minimize the security risks associated with exporting such information, only a limited set of system metrics are available outside of TEE and with timescale resolution that is deemed to be safe.

The mechanism to export TEE system counters to Meta backend is similar to FB303 counters, but instead of application they are emitted by a custom collector.

These operating system metrics are supported:

- System CPU utilization (no per-process granularity)
- System memory utilization (no per-process granularity)
- Filesystem utilization (disk space used)
- Limited GPU metrics (such as current utilization, memory usage, temperature, and power draw)

More detailed system metrics could be possible to obtain via Remote diagnostics when required, but with rate limiting applied to ensure privacy preserving properties of the TEE.

Export for GPU metrics

For effective GPU status monitoring and failure detection, it is important to collect health and status metrics from GPUs.

We collect a limited set of these non-privacy-sensitive GPU health metrics from GPUs in Private Processing environments, initially focusing on device information like firmware version. These metrics are published on an out-of-band monitoring interface. In confidential GPU or Protected PCIe (PPCIe) mode, that interface only exposes non-sensitive metrics. We expect to collect more out-of-band GPU health metrics over time.

Threat model

In order to assess the design of Private Processing against the principles and guarantees we set out to uphold, we have developed a threat model to identify the potential attack vectors and vulnerabilities which could compromise the promises we make to users. We have analyzed the key assets, actors, and threats which are critical to those guarantees. Across each of these areas, we have mitigations in place to limit the viability of an attack, and welcome feedback on ways to improve mitigations.

As we shared in our Security Engineering blog, our threat model for Private Processing includes three key components:

- **Assets:** The sensitive data and systems that we need to protect.
- **Threat actors:** The individuals or groups that may attempt to compromise our assets.
- **Threat scenarios:** The ways in which our assets could be compromised, including the tactics, techniques, and procedures (TTPs) that threat actors might use.

Assets

Private Processing is meant to protect a primary asset or assets which may vary by feature. In the context of WhatsApp's Writing Help or Summarization features, it is meant to protect messages, whether they've been received by the user, or are still in draft form.

In addition to the primary assets, e.g. messages, we also protect secondary assets that support the goals of Private Processing and may interact with the primary assets. These include:

- The Trusted Computing Base (TCB) of the Confidential Virtual Machine (CVM)
- The underlying host and its hardware
- The cryptographic keys used to protect data in transit, provide transparency and attest the platform

Threat actors

We have identified three threat actor types that could potentially attack our system to attempt to recover assets.

1. Malicious or compromised insiders with access to our infrastructure.
2. A third party or supply chain vendor with access to components of the infrastructure.
3. Malicious end users targeting other users on the platform.

Because of our dependency on third-party vendors, we consider threat actors (1) and (2) individually. Defeating this attestation method would require collusion with our hardware vendors and is therefore out of scope.

Threat categories and mitigations

There are a number of threat categories by which threat actors may attempt to compromise the system, each with a number of scenarios we have identified. The tables below, one for each category, outline the scenarios we have considered and associated mitigation strategies we have developed in Private Processing. These scenarios are not exhaustive and we will continue to evolve our defenses in response to emerging threats.

Platform Threats

Platform threats encompass attack scenarios that target any exposed surface where the system processes untrusted data.

Threat scenario	Threat details	Mitigations
Service exploitation	External users or malicious insiders could attempt to target network-reachable vulnerabilities in the CVM service. Given each CVM serves multiple users, a compromise of any service running within a CVM risks access to other users' messages. Furthermore, malicious insiders could more directly target downstream CVM nodes.	Meta designed Private Processing to reduce such an attack surface by limiting the exposed entry points to a small set of thoroughly reviewed components, which are subject to regular assurance testing. The service binaries are hardened and run in a containerized environment to mitigate the risks of code execution, and limit a compromised binary's ability to exfiltrate data from within the CVM to an external party. See the Secure software section for more details.
Prompt injection	External users could attempt prompt injection attacks against specific users if the users's context window contains untrusted data. While dependent on use case, prompt injection could allow the attacker to exfiltrate a user's messages.	Features built on Private Processing do not share any data to anyone else, so Prompt Injection does not expose any data to outside parties.

System software threats

Threats to the system software encompass internal and external actors targeting software run by the CVM to attempt to exfiltrate messages or disrupt the platform through denial of service (DoS).

Threat scenario	Threat details	Mitigations
Logging data disclosure	The CVM or services running in it could directly log primary assets or expose side channels to reconstruct them. Direct logging or side-channels could be incorporated by accidental inclusion in by malicious changes to the service.	Private Processing implements a log-filtering system to limit export of only explicitly allow-listed log lines, outlined in more detail in the Debugability and operations section.
Source control and build system attacks	A malicious insider could attempt to compromise Meta's source control or build system to place a backdoor or vulnerability enabling external persistence into the CVM environment.	Meta's builds the CVM and associated services in restricted environments that maintain provenance and require multi-party review.
Software supply-chain compromise	The CVM contains third party dependencies in both the operating system and application service that an attacker could attempt to compromise upstream.	Meta actively monitors known package compromises and has restricted the CVM runtime through containerization and network controls to limit a compromised binary's ability to exfiltrate data externally. See the Vulnerability management section for more details.

Privileged access threats

Privileged Access threats arise from the potential for misuse, compromise, or unauthorized access to the host machines supporting the platform or to the hardware itself.

Threat scenario	Threat details	Mitigations
Malicious time sourcing	Since the host controls the time sources for the CVM today, any timestamps relied on by the CVM can not be trusted for freshness or expiration checks.	A client-source timestamp is used in cases where time needs to be securely sourced.
Host-level attacks	Any interface between the hypervisor and CVM could potentially provide input mechanisms for an attacker to influence the CVM behavior, e.g. a TOCTOU vulnerability could modify artifacts being loaded from the host.	To address these unknown risks, we built Private Processing on the principle of defense-in-depth by: • Actively tracking novel vulnerabilities in this space. • Minimizing and sanitizing untrusted inputs to the TEE, • Minimizing attack surface through CVM hardening and enabling abuse detection through enhanced host monitoring. Because we know that defending against physical access introduces significant complexity and attack surface even with industry-leading controls, we continuously pursue further attack surface hardening: • Ensuring all memory used by the CVM is encrypted such that DRAM cannot be inspected. • Deploying industry standard physical security controls to protect our datacenters from bad actors. • Further diffusing targeted attacks by operating traffic to Private Processing hosts through a third-party OHTTP relay proxy. • Leveraging Anonymous Credentials to provide non-targetability.
TEE software exploitation	This is a growing area of security research, and vulnerability researchers have demonstrated the ability to bypass TEE guarantees under certain circumstances directly from a host.	
Physical host attacks	Physical host attacks may attempt to defeat TEE guarantees or present prior-compromised hosts as legitimate to an end user.	
Hardware supply-chain compromise	A vendor compromise may result in Meta receiving compromised hardware components that subvert TEE guarantees. Malicious insiders at vendors could also attempt to compromise PKI to allow them to hotpatch malicious updates.	

System-level threats

System-level threats encompass malicious activities targeting the core security properties of the system.

Threat scenario	Threat details	Mitigations
Attestation measurement attacks	An attacker could attempt to make the CVMload additional code or malicious configurations in such a way that they remain unmeasured and non-transparent.	All security-critical, externally loaded artifacts are measured and have their own transparency log entry. No code is ever loaded privately, all models are loaded in a weights only format, preventing deserialization attacks. See the Artifact transparency and Attestation and the boot process sections for more details.
Attestation certificate extraction	An attacker may attempt to compromise a CVM to exfiltrate an RA-TLS certificate private keys to trick clients to connect to non-TEE hardware.	All connections initiated from the client leverage a freshness nonce to prevent replaying prior attestations. See the Remote attestation and connection section for more details
Transparency attacks	An attacker may attempt to run a CVM without verifiable transparency by running the service without publishing a corresponding entry in the transparency log.	An inclusion proof of the attestation measurement is validated by the client before sending any data over the connections. See the Attested, encrypted communications section for more details.
Image roll back attacks	An attacker could attempt to run a previous CVM version which now has known vulnerabilities.	An explicit revocation list is passed by the server for any image determined to be externally exploitable. All artifacts also contain expiration dates to limit their validity. See the Artifact expiration and revocation section for more details.
Targeted routing attacks	The third party relay proxy may attempt to redirect all traffic from a particular user to another cluster of malicious TEEs that validate to the client.	The use of Anonymous Credentials and OHTTP prevents this, and also requires compromising Meta's private key used by client's encapsulation of the request with HPKE. See the Anonymous routing section for more details.
User de-anonymization side-channel attacks	Meta insiders may attempt to de-anonymize requests based on side-channels from the client to identify individual users.	Client-side logging is limited to not reveal identifying information prior to selection.
ACS De-anonymization attacks	Meta insiders may attempt to provide especially crafted ACS tokens to users to identify them.	The client validates ACS tokens using keys distributed by a third-party to ensure they are not uniquely identifiable.

In-app transparency reports

To ease auditability for end-users and external security researchers, WhatsApp offers users the ability to record and export their interactions with the Private Processing system. These reports include security and transparency information in addition to the data that was processed by the system, with some caveats (e.g. to account for deleted data). A high-level overview of the data included in the reports are:

1. Type of requests processed by the Private Processing system (e.g. summarize, writing suggestions), timestamps, and additional request details.
2. Information processed by the Private Processing system and the result of the request and the response.
3. Attestation information for TEEs that processed the data along with transparency information of the CVM that runs atop.

Private Processing reports are off by default. Once a user turns on Private Processing reports, reports are generated on-device at the user's request, at which point the user can decide where to save that information locally on the device. The security and transparency information includes all TEEs that processed the data.

The goal of these reports is twofold:

1. Demonstrate what data has been shared to Private Processing to verify only intended data is sent from the client to Private Processing.
2. Log details of the encrypted, attested connection to allow verification.

For details on how to set up and use this feature, see this WhatsApp Help Center article.

Key fields

Field	Purpose
Type of request	Summarize, writing suggestions.
Request UUID	Request identifier.
Request timestamp	Time and date of the request.
App version	App version.
Locale	Language and region setting of the client.
Client surface	Identifies the WhatsApp client surface initiating the request.
Timezone offset	Client timezone offset from UTC, in hours.
Phone number country code	Country code associated with the user's phone number.
Message content	Messages sent to the Private Processing system. The report reflects the latest available content. Messages that are no longer available are redacted. Chat locked messages are not included.
Conversation type	Type of conversation being processed by Private Processing.
Conversation name	Name of the conversation being processed.
Web search consent state	User's consent state for web search functionality in Private Processing.
Private Processing response	Response for the request.
Attestation artifact digests	Digests of non-user-specific artifacts loaded into the CVM, such as integrity models and trending topics. Allows verification of what common data was available in the processing environment.
Attestation report	Attestation of the code running on the Private Processing system.
Attestation certificate chain	Used to verify the origin of the TEE hardware from AMD.
Signature	Cloudflare signature on the verified data (measurement hash and image Id).
Version	Indicates the ciphersuite used.
Transparency namespace	Field to verify the software packages included in the Confidential Virtual Machines (CVM) where TEE is being run.
Signature timestamp	Time for the Cloudflare signature on the transparency proof.
Epoch	Ever-increasing value incremented by 1 for each Cloudflare signature.

Enabling web search in Private Processing

Because the knowledge of Large Language Models (LLMs) is time-bound to the information available up to a specific point in time when they were trained, LLMs may need to perform web searches to be useful in response to particular contemporary user requests. That is why some Private Processing features may require information that is not available locally in the TEE.

For example, when chatting with AI, users may ask for information beyond its local knowledge, like the latest news or a real-time sports score. When this happens, the AI may initiate a web search request. This is an optional capability and users can disable web search at any time.

If the model determines a user's prompt requires real-time information, it generates a search query based on the user's prompt and sends it to Meta infrastructure outside of the TEE which directs it to selected web search engine providers.

The web search results are returned to Private Processing to form an AI response. To provide auditability and transparency, the user can see the exact web search query directed to the web search engine and the sources that the AI used in generating its response in the UI and in-app transparency report.

Users can control whether to use web search and turn it off or on through an in-app setting. When web search is turned off, the AI can only use its local knowledge to respond. Here is how it works:

Generating a search query:

- A model in the TEE forms a search query from the user's prompt and relevant conversation context.
- The query is sent from the TEE to Meta Infrastructure, which then directs the request to an external web search provider.

Data security safeguards applied:

- **Unlinked searches:** Search queries are not associated with the user making a prompt.
- **Search transparency:** For each response that uses web search, search terms can be viewed in the UI and our in-app transparency report to ensure users can understand what data formed a search sent to Meta and web search engines.
- **Data minimization:** The model in the TEE and prompts are built to minimize what is used to form a search so that only the minimum necessary data leaves the Private Processing system.
- **Limiting query length:** Search query length is limited to 100 characters to further minimize data used to form a search query.
- **Limiting number of queries:** Only up to 5 search queries per User Prompt are allowed in order to reduce risk of prompt injection attacks or unexpected behaviors of the search planning model. Results from the web search are returned to the TEE, where they are integrated into the AI response.

Conclusion

Throughout this paper, we have outlined the work we are doing today to bring Private Processing to the market in a safe and secure manner.

We expect the Private Processing hardware and software requirements to continue to evolve in response to new threats and the evolving product capabilities, which will necessitate future hardening and updates to this system. As these requirements evolve so will our threat model and mitigations, and we'll continue updating this paper to share our learnings and architecture.

We welcome feedback from our users, researchers, and the broader security community through our security research program:

- **More details:** https://bugbounty.meta.com
- **Contact us:** bugbounty@meta.com

Glossary of terms

Term / Acronym	Definition
Abstract Syntax Notation One (ASN.1)	A standard Interface Description Language (IDL) that defines data structure for serialization and deserialization. Widely used in telecommunications, computer networking, and cryptography to ensure data can be exchanged and interpreted consistently across different platforms and systems.
Advanced Micro Devices (AMD)	A multinational semiconductor company known for designing and developing computer processors (CPUs) and graphics technologies (GPUs).
Advanced Encryption Standard (AES)	A widely used symmetric encryption algorithm established by the National Institute of Standards and Technology (NIST) in 2001 for protecting sensitive data.
AES Key	The cryptographic key used within the AES algorithm. The same key is used for both encrypting and decrypting data (symmetric).
AMD Secure Encrypted Virtualization Secure Nested Paging (AMD SEV-SNP)	Enhances virtual machine security by encrypting memory and implementing memory integrity protections, creating an isolated and secure execution environment for virtual machines.
Amplitude Shifting Key (ASK)	A digital modulation technique used to transmit data by varying the amplitude of a carrier wave. This form of amplitude modulation represents binary data (0 and 1) by different voltage levels.
Anonymous Credential Service (ACS)	A system that enables users to authenticate and prove possession of certain attributes without revealing their personal information. This crucial tool for privacy-focused authentication allows users to interact with services without compromising their identity.
Anonymous Credentials Service (ACS) token	A credential based on the Anonymous Credentials protocol and issued by the Anonymous Credential Service. This is used to authenticate users without de-identifying them.
Archival Resource Key (ARK)	A persistent URL identifier used to support long-term access to digital resources, helping ensure that information can be found and accessed reliably over time, even if the location of the resource changes.
Artifact	A byproduct of software development that helps describe the architecture, design, and function of software. Artifact types can be tangible or intangible, including documents, code files, diagrams, executable programs, or any other output that contributes to the creation, understanding, or maintenance of a software system.

Term / Acronym	Definition
Attestation	A process of verifying the integrity, authenticity, and compliance of a system, identity, or workload, often through cryptographic means, to establish trust in otherwise opaque or automated entities.
Attestation Agent (AA)	A component that provides proof or evidence of a system's integrity, authenticity, and compliance; a key player in Confidential Computing and secure software development, helping to establish trust in digital environments.
Attestation bundle	A collection of multiple related attestations packaged together in a single file; they typically provide verifiable information about software artifacts or events, such as build provenance, code review results, or vulnerability scan reports.
Attestation policy for AMD SEV-SNP	Defines the requirements for AMD SEV-SNP TEEs including guest policy, which security features are expected to be enabled/disabled, and minimum TCB policies that are recommended by the hardware manufacturer.
Attestation report	A formal document, typically used in conjunction with financial statements, often prepared by a third-party expert, that verifies and validates the security posture or compliance of an organization.
Attestation verification	The process of confirming the authenticity, integrity, and compliance of a system, device, or software; a formal way of proving that something is as it claims to be using cryptographic methods, this process helps build trust in digital environments by ensuring data is not tampered with and systems are running as intended.
Base filters	A foundational or default set of criteria used to initially narrow down a dataset or research result. Usually defined at the system or view level and available for everyone to use.
Basic Encoding Rules (BER)	A set o f guidelines that define a specific way to encode information, especially within the context of ASN.1. It uses Type-Length-Value (TLV) structure, where each data element is encoded with its type, length , and actual value.
Certificate Revocation List (CRL)	A document maintained by a Certificate Authority (CA) that lists digital certificates that have been revoked before their expiration date. Informs users and systems when a specific certificate is no longer valid and should not be trusted for secure communications or authentication.
Common Vulnerabilities and Exposures (CVE)	A publicly listed catalog of known security vulnerabilities and exposures maintained by the MITRE Corporation, a non-profit organization.
Confidential Computing Consortium (CCC)	A community focused on advancing Confidential Computing capabilities and adoption through open collaboration, particularly on projects securing data in use. The CCC brings together hardware vendors, cloud providers, and software developers to accelerate the adoption of Trusted Execution Environment (TEE) technologies and standards.

Term / Acronym	Definition
Container Vulnerability Management (CVM)	A security practice focused on identifying, assessing, and mitigating vulnerabilities within container images and containerized applications.
Cryptographic hash	A one-way function that transforms data of any size into a fixed-size output, known as a hash value or digest. You can't get the original input from the hash function and are crucial for verifying data integrity, authentication, and secure storage of passwords.
Cryptographic materials	All data, devices, equipment and software containing or used for cryptographic information, including things like encryption keys, HSMs, cryptographic algorithms, and software libraries that implement these algorithms. Cryptographic material aims to secure and protect information and communications.
Custom collector	A specialized tool or component designed to gather and process data in a way that standard collectors cannot;; used when data collection needs to be tailored to specific requirements, such as collecting data from a particular device, application, or system, or transforming the data in a unique way.
Distinguished Encoding Rules (DER)	A set of specifications that provide a unique and canonical way to encode data according to Abstract Syntax Notation One (ASN.1). As a subset of the Basic Encoding Rules (BER), it restricts the ways data can be encoded, ensuring a consistent and unambiguous representation for each ASN.1 value.
Distributed Denial-of-Service (DDoS)	A cyberattack where multiple systems flood a target server or network with traffic, aiming to overwhelm it and make it unavailable to legitimate users.
Elliptic Curve Digital Signature Algorithm (ECSDA)	A widely used cryptographic algorithm for generating and verifying digital signatures; an efficient and secure method for authenticating messages or data and resistant to various cryptographic attacks.
ECDSA Key	A digital signature algorithm (DSA) which uses keys (a private key and a public key) derived from elliptic curve cryptography (ECC), enabling secure authentication and date integrity.
Elliptic Curve Cryptography (ECC)	A public-key cryptosystem based on elliptic curves over finite fields providing strong security guarantees with smaller keys used for multiple cryptographic operations including key agreement, digital signatures and (indirectly) encryption.
Ephemeral data processing	The handling of data that is temporary, short-lived, and designed to be discarded after a specific purpose or period, often used in applications like ephemeral messaging, ephemeral storage, or ephemeral environments.
Ephemeral environments	A temporary, isolated and on-demand deployment of an application system, created for specific purposes like testing, previewing features, or collaboration, and destroyed when no longer needed.
Ephemeral messaging	Messaging applications or features that automatically delete messages after a set period or upon viewing, leaving no permanent record of the communication.

Term / Acronym	Definition
Epoch	The data and time relevant to which a computer's clock and timestamp values are determined.
Extended Berkeley Packet Filter (eBPF) program	A Linux kernel technology enabling engineers to build programs that run securely in kernel space.
FB303 Counters	A core set of thrift functions that provide a common mechanism for querying stats and other information from a service.
Function Level Reset (FLR)	A targeted reset operation that specifically affects a single function or virtual function (VF) within a PCIe device, rather than the entire device, useful in Single Root I/O Virtualization (SR-IOV) scenarios.
Galois/Counter Mode (GCM)	A mode of operation that provides data confidentiality (encryption) and authenticity (integrity), as defined in the NIST Special Publication 800-38D.
Host Hardening	The process of securing a computer system or server by reducing its vulnerabilities and potential attack surfaces.
Host Monitoring	The process of tracking the performance, availability, and overall health of individual servers (physical or virtual) within an IT infrastructure. It involves collecting and analyzing data from the host to detect issues, ensure optimal performance, and facilitate proactive maintenance.
Hybrid Public Key Encryption (HPKE)	A cryptographic protocol (defined in RFC 9180) that combines public key encryption with symmetric encryption, which allows a sender to: • encrypt a symmetric key using a recipient's public key • encrypt/authenticate a message with that symmetric key • Send the two ciphertexts to the recipient This ensures that ciphertext is not tampered with after its creation.
Hypervisor (Virtual Machine Monitor [VMM])	Software that allows multiple VNS to run on a single physical machine by abstracting the hardware resources and allocating them to each VM.
Interface Description Language (IDL)	A language used to describe the interfaces of software components, allowing for communication and interaction between different programming languages or systems.
Joint Test Action Group (JTAG)	A standardized hardware interface, based on the IEEE 1149.1 standard, primarily used for debugging, testing, and programming electronic devices, especially complex chips and boards.
Linux Containers (LXC)	A virtualization technology that allows you to run multiple isolated Linux environments (containers) on a single host system.

Term / Acronym	Definition
LXC Image	A snapshot or template used to create and deploy LXCs, which contain the necessary files and configurations for a container to run, providing a lightweight and isolated environment for applications.
Linux Kernel Self-Protection Project (KSPP)	The design and implementation of systems and structures within the Linux kernel to protect against security flaws in the kernel itself, which focuses on enhancing the kernel's resilience against attacks by addressing vulnerabilities within the kernel, rather than the userspace from exploitation.
Namespace	A grouping of related names, often used to avoid naming conflicts and organize resources; essentially like a container that holds unique identifiers (names) for things like variables, functions, classes, tables, or even entire web domains.
Natural Language Processing (NLP)	A branch of artificial intelligence (AI) that focuses on enabling computers to understand, interpret, and generate human language, allowing for more natural and effective human-computer interaction.
Non-Volatile Memory Express (NVMe)	A storage access and transport protocol that allows flash and solid-state drives (SSDs) to communicate directly with a computer via a high-speed PCIe bus, enabling faster data transfer and lower latency compared to older standards.
NVIDIA's Hopper Confidential Computing Platform	A hardware-base solution for securely processing data and code that is in use, preventing unauthorized access and modification. Hopper GPUs are ideal for demanding tasks, such as AI inference, deep learning training, scientific simulations, data analytics, and generative AI, accelerating these workloads for faster and more efficient results.
NVIDIA Remote Attestation Service (NRAS)	A robust security mechanism designed to enable the verification and validation of the integrity and authenticity of NVIDIA devices and platforms.
NVLink	A high-speed point-to-point interconnect technology developed by NVIDIA, primarily used for enabling fast data transfer and communication between GPUs (and between GPUs and CPUs), offering a significantly higher bandwidth and lower latency than traditional PCIe interconnects.
Object Identifier (OID)	A unique, persistent, and unambiguous name used to identify objects, concepts, or things, typically represented as a sequence of numbers separated by periods. They are used in various protocols and applications to uniquely identify data elements, objects in databases, and even network devices.
Oblivious HTTP (OHTTP)	A protocol that helps make online requests more private by separating who is making a request from the content of the request itself.
Oblivious HTTP Application Intermediation (OHAI) Proxy	A protocol that aims to provide secure, low-latency HTTP exchanges between clients and servers while protecting the identities of both parties. It allows for HTTP requests and responses to be handled in a way that prevents the intermediary proxy from learning about the content of the requests or responses.

Term / Acronym	Definition
On-Die	Components or functionalities integrated directly onto the silicon die. A die is a small silicon square containing an integrated circuit, while a wafer is a thin circular slice of semiconductor material from which multiple dies are fabricated. On-die examples include transistors, diodes, resistors, and other components that form the integrated circuit.
Open Telemetry (OTel) SDK	A Software Development Kit that provides a working implementation of the Open Telemetry API, enabling developers to instrument their applications, collect telemetry data (metrics, logs, and traces), and export it to various backends for analysis.
Orchestrator	A tool or system that manages and coordinates complex workflows, tasks, and processes across various IT systems and environments by automating the coordination of multiple tasks, ensuring they work together effectively to achieve a specific outcome.
Package manifest	A file (often XML or YAMML) that provides metadata about a software package or application. It describes the package contents, dependencies, and other relevant information needed for installation, deployment, and management.
Page Table Entry (PTE)	An entry in a page table, which is a data structure used by operating systems to map virtual memory addresses to physical memory addresses.
Payment Service Provider (PSP)	A company that facilitates electronic payment transactions between parties, such as customers, businesses, and financial institutions. They provide the infrastructure and services for businesses to accept a wide range of payment methods (credit cards, debit cards, digital wallets, and bank transfers).
PSP-generated signature	A digital signature created by a Payment Service Provider (PSP) to authenticate and secure payment transactions; used to ensure the integrity and authenticity of payment data, guaranteeing the transaction's validity and protecting against fraud.
Peripheral Component Interconnect Express (PCIe)	A high-speed serial interface standard used for connecting various computer components, including graphics cards, storage devices, and other peripherals, offering significantly faster data transfer rates than its predecessor, PCI.
Prime and probe	A type of side-channel attack that exploits shared cache memory to leak secrets from cryptographic computations or other processes; An attacker "primes" the cache by filling it with data they can control, then "probes" the cache by accessing specific memory locations to see if the victim's activity has evicted the primed data, revealing information about the victim's memory access patterns.
Prometheus	An open-source systems monitoring toolkit that collects and stores metrics data as a time series; designed to monitor cloud-native applications and infrastructure.
Prompt (command) injection	A type of code injection attack that leverages adversarial prompt engineering to manipulate AI models.

Term / Acronym	Definition
Proxygen	A C++ HTTP library developed by Meta that provides tools for building high performance HTTP servers, proxies, and clients.
Quick Emulator (QEMU)	A widely used, free, open-source software that serves as a machine emulator and virtualizer that allows users to run operating system and applications designed for one hardware platform (like ARM) on a different platform (like an x86 PC) by emulating the target architecture. QEMU can also function as a hypervisor, enabling the creating of VMs with near-native performance.
Red Hat Enterprise Linux (RHEL)	A commercial open-source Linux distribution designed for enterprise use, known for its stability, reliability, and security, offering certified software, hardware, and support (including long-term support), along with tools for managing large-scale deployments.
Remote Attestation TLS (RA-TLS) Certificate	A TLS certificate that incorporates Intel Software Guard Extensions (SGX) remote attestation information, ensuring the integrity and confidentiality of the TLS connection by verifying the enclave's trustworthiness. Defines lifetime of the certificate.
Remote diagnostics tooling	Software and hardware used to diagnose and troubleshoot issues with devices, systems, or networks from a remote location, enabling IT professionals to access, monitor, and rectify problems on devices without being physically present.
Reverse map table	(Reverse mapping) A data structure or process that maps backwards from a destination (like a physical memory page) to its source (like a virtual address or a list of PTEs). Used for tasks like memory management and security, where knowing which virtual addresses or PTEs point to a specific memory location is crucial.
Revocation mechanism	A process that invalidates a digital certificate or key before its official expiration date; crucial for security because it allows organizations to quickly disable compromised or untrusted credentials, preventing unauthorized access or data breaches.
Revocation policy	Similar to Transparency Policy, specifically for revocation of different types of artifacts.
rsyslog	An open-source software utility for high-performance log processing on UNIX and Unix-like systems. It collects, transforms, filters, and routes log messages within an IP network.
rsyslog exporter	A tool that makes rsyslog's internal metrics (log processing speed and memory usage) available for monitoring and alerting by exporting them to a Prometheus-compatible endpoint, allowing system admins to track the health and performance of their rsyslog servers and detect potential issues.

Term / Acronym	Definition
Root of Trust (RoT)	A security foundation upon which all secure operations of a computing system depend, ensuring the integrity and authenticity of critical functions like secure boot and authentication, often implemented in hardware.
Secure Cloud Business Applications (SCuBA)	A program developed by the Cybersecurity and Infrastructure Security Agency (CISA) aiming to enhance security configurations for cloud environments and helping agencies adopt necessary security and resilience practices.
Security Protocol and Data Model (SPDM)	A standard developed by the Distributed Management Task Force (DMTF) that defines messages, data objects, and sequences for secure communication and authentication between hardware components, like PCIe cards and NVMe.
Self-signed X.509 certificate	A digital certificate that is created and signed by the entity using it, rather than by a trusted Certified Authority (CA); typically used for internal purposes (development, testing or private networks) where strict validation against a CA hierarchy is not required.
Service filters	Mechanisms that control and potentially modify traffic related to specific services (like web, email, or network protocols). They act as gateways to analyze, filter, or transform data exchanged between clients and services, which can occur before, during, or after, service processing, and can involve tasks like authentication, authorization, security enforcement, and content inspection.
Single Root I/O Virtualization (SR-IOV)	A hardware technology enabling a single PCIe device to appear as multiple virtual devices, allowing multiple VMs to access it directly, improving I/O performance and resource utilization.
Software hardening (aka application hardening)	A cybersecurity strategy that involves strengthening applications and systems by reducing their vulnerability to attacks through measures like updating software, patching vulnerabilities, and implementing security measures. This process includes removing any unnecessary software, disabling unused services, applying security patches and configuring settings to enhance protection.
Syslog identifier (facility code)	A field within a Syslog message that identifies the source or the application that generated the message, helping categorize and prioritize logs in order to filter and analyze them easier.
System-On-Chip (SoC)	An integrated circuit that combines multiple components, such as CPUs and GPUs, memory, and I/O interfaces, onto a single chip, enabling smaller, more efficient, and faster devices.
SoC packaging	The physical enclosure and interconnects that protect and connect a SoC chip to other components in a system, enabling smaller, more efficient, and cost-effective devices.
Task manager	A utility program that helps users monitor and manage processes and applications running on a computer (often included in an OS, such as Windows).

Term / Acronym	Definition
Task manager counters	Specific performance metrics, like CPU usage or memory consumption, that are displayed within a Task Manager utility, providing real-time data on how the system is utilizing its resources.
Trusted Computing Base (TCB)	The upgradable firmware components relevant to AMD SEV-SNP. Includes SP API versions, CPU microcode, and more.
Threat actors	Any individual, group, or entity that intentionally poses a risk to digital systems, infrastructure, or data, ranging from cybercriminals motivated by financial gain or hacktivists driven by political agendas, and even insiders within an organization who may be negligent or malicious.
Threat scenarios	A specific, detailed description of a potential cybersecurity incident or attack, outlining the threat actor, the vulnerability, the potential impact, and possible mitigation strategies. These scenarios help organizations understand the nature of threats and prioritize risk mitigation efforts.
Thrift	An open-source framework that facilitates interoperable and scalable services across different programming languages.
Thrift functions	Define the API of a service, allowing different programming languages to communicate and interact with each other.
Transparency policy	Defines which keys are trusted for transparency proofs as well as their freshness requirements. Freshness requirements are based on the artifact type and each artifact has their own namespace to distinguish the requirements between them. Similarly, namespaces are also used to separate services (e.g., Orchestrator has a namespace as well as the Predictor). This separation allows defining policies on which TEEs are allowed to connect which other TEE services.
Transparency proof	A measurement that provides verifiable evidence about the state of a TEE at a specific point in time; this mechanism shows (often with cryptographic proofs) that the TEE was in a known secure state before it performed sensitive operations, and that it remained in a known secure state afterwards, which builds confidence in the TEE's integrity and security.
Transparency report	A regularly released document that discloses key information about how a company handles data, enforces policies, and responds to legal requests. These reports aim to build user trust and demonstrate commitment to protecting user rights and privacy.
Transport Layer Security (TLS)	A cryptographic protocol that provides secure communication over a computer network, particularly for web browsing. It's the successor to Secure Sockets Layer (SSL).

Term / Acronym	Definition
TLS 1.3 Handshake	The process by which a client and a server establish a secure, encrypted connection using the TLS 1.3 protocol, which involves exchanging information, including cryptographic parameters and certificates, agreement on encryption methods, and authentication of each other. This creates a secure channel for data transmission, preventing eavesdropping and ensuring data integrity.
TLS 1.3 Protocol	The latest version of the Transport Layer Security (TLS) protocol that ensures secure communication between clients and servers over a network. Building predecessor protocols (like Secure Sockets Layer [SSL]), it focuses on stronger security, faster performance, and simplified implementation.
Trusted Execution Environment (TEE)	A secure, isolated, hardware-base boundary within a processor that protects sensitive data and code from unauthorized access, ensuring the data and code integrity and confidentiality.
TEE Secure Analysis (TSA)	A service that provides minimal, safe and deterministic debug capabilities through the analysis and assessment of the TEE to ensure its security and integrity. TSA services help organizations identify potential vulnerabilities, assess compliance, and ensure the TEE is functioning as intended.
Trusted time	Since TEEs do not have a trusted time source, policies can include a current time field that can be used to ensure the freshness checks discussed above are accurate.
Unified Extensible Firmware Interface (UEFI) Specification	Defines a software interface between an operating system and platform firmware (a modern replacement for the legacy BIOS), offering enhanced features like faster booting, improved security, and support for larger storage devices.
Unstructured logs	Log files that lack a predefined format or schema, typically containing a mix of text, numbers, and potentially other data types, which makes them challenging to parse and analyze automatically.
User data exfiltration	The unauthorized transfer of data from a computer or network to an external location, often involving malicious intent. This type of data theft occurs when someone copies, retrieves, or transmits data from a secure system without proper authorization (either manually or through malware), posing a significant risk to organizations and potentially leading to financial losses, reputational damage, and legal consequences.
Versioned Chip Endorsement Key (VCEK)	(From the context of CC and AMD SEV-SNP) A private ECDSA key derived from a chip-unique secret and specific TCB version, used to sign attestation reports and verify the integrity of the system's firmware.
VCEK certificate	A digital certificate associated with a specific version of a TPM chip, containing information about the chip's unique EK and its manufacturer's verification; a key component in establishing trust and security when using TPMs for various applications like secure boot, encryption, and data protection.

Term / Acronym	Definition
Virtual Large Language Model (vLLM)	An active open-source library that supports LLMs in inference and model serving efficiently.
Virtual Machine Socket (VSOCK) interface	A communication channel that allows VMs to communicate with their host system or other VMs using a socket-based protocol, offering a streamlined, zero-configuration approach.