Confidential Inference Systems

Design principles and security risks

Version 1.0, June 2025

About Pattern Labs

Pattern Labs is an AI security company working to assess and mitigate risks in advanced AI systems. Pattern Labs conducts research on security and policy for AI systems and collaborates with AI labs, academia, industry, and governments.

About Anthropic

Anthropic is an AI safety and research company that creates reliable, interpretable, and steerable AI systems. Anthropic's flagship product is Claude, a large language model trusted by millions of users worldwide.

Contents

Overview

This whitepaper describes principles for designing a "**confidential inference system**" – a system that securely uses confidential computing technologies to implement an AI model inference service – typically used by generative AI applications, such as chat assistants and image generators – that protects the confidentiality of user data (model inputs and outputs) and/or the AI model itself (its weights and architecture).

The structure of this whitepaper is as follows:

- This "*Overview*" section provides an overview of what the confidential computing use cases and guarantees are in the context of AI inference.
- The section "*The two aspects of confidential inference*" provides a high-level description of what confidential inference systems aim to achieve, both with confidential data and with confidential models.
- The section "*Service provider guarantees*" details what the service provider (typically a cloud service provider) guarantees the other parties in order to ensure confidentiality.
- The section "*Trusted execution environments with AI accelerators*" details approaches to including AI accelerators in the confidential inference system.
- The section "*Components and their design principles*" presents a sketch, or reference design, of a confidential inference system implementing both the confidential data and the confidential model aspects, and dives into some principles involved in the design of such a system.
- The section "*Threat model, risks and attack scenarios*", while not a comprehensive security review of a complete design or implementation, goes over some of the major security threats involved with confidential inference.

What is confidential computing, in the context of AI inference?

The Confidential Computing Consortium defines Confidential Computing ([1], section 2) as:

> *Confidential Computing is the protection of data in use by performing computation in a hardware-based, attested Trusted Execution Environment.*

It also highlights AI inference as a key use case for confidential computing technologies [3].

Typically, three parties are involved in AI inference computations:

1. The **model owner**. Usually, this is the organization that built/trained the model, and owns the model architecture and model weights. In some cases, such as when a


base model is fine-tuned on proprietary data, the model owner may be the organization that adapted the model.

2. The **data owner / end user**. The data owner owns data that they wish to provide as input to an AI inference service, and receive its output.
3. The **service provider**. This is either a cloud service provider, or an on-prem compute service. There are two options for what the service provider provides:
    a. Infrastructure-as-a-service (IaaS): Virtual machines, AI accelerators, private networks, storage, key management systems and other cloud services, on top of which the model owner builds their AI inference service.
    b. "Inference-as-a-service": A cloud service which provides AI inference APIs. Examples of cloud platforms with AI inference services include Amazon Bedrock on AWS, Vertex AI on GCP, and Azure AI services.

The confidentiality requirements

An inference computation typically consists of performing forward propagation through a deep neural network with many[1] parameters (model inputs and model weights), generating a result (model outputs). The parameters are used in mathematical operations, such as matrix multiplication. The precise order and layout of these mathematical operations are essentially the model architecture. The operations require direct access to the input/weight parameter values, and yield direct access to the output values.

Therefore, the computation requires full access[2] to the inputs, outputs, and weights. However, for various reasons, including business, security, privacy, and compliance, the parties wish to perform the computation confidentially:

1. The **model owner** wants to keep the model architecture and weights confidential, without needing to trust the other two parties.
2. The **data owner** wants to keep the model inputs and outputs confidential, without needing to trust the other two parties.
3. The **service provider** wants to host the AI service while assuring the other two parties that it cannot access their weights and data.

[1] In today's largest models, the amount of parameters ranges from hundreds of billions to a few trillions.

[2] Ideas such as Homomorphic Encryption – making computations without decrypting the parameters – have not yet matured enough to be usable in the scale and performance required for AI inference, leading to full decryption being a requirement for successfully performing an inference computation.

Meeting the requirements with confidential computing

Confidential computing technologies include the ability to set up Trusted Execution Environments (TEEs), having the following properties:

1. With hardware support, the memory of the TEE is inaccessible, even to the operating system kernel.
2. Similarly, debugging features like breakpoints, traces, counters, etc. are disabled inside the TEE, making it impossible to monitor or alter execution.
3. The TEE can produce a cryptographically-signed proof ("attestation document") proving that it is running the expected code (the image was not tampered with).

Confidential computing TEEs are the basis for confidential inference systems where the requirements of all parties can be met. The design principles for such a system, and the associated security risks, are detailed in this document.

For the definitions and technical analysis of confidential computing technologies used in this document, see [1] and [2].

Limitation of scope

- This document discusses the confidentiality of the model *only during the inference computation*. Other procedures in the model's lifetime are not discussed. For example, insecure backups of the model weights may defeat their confidentiality.
- This document focuses specifically on inference. Although confidential computing is also relevant for other AI use cases where confidential access to the model is desired (such as training, fine-tuning based on proprietary data, interpretability research, and others), these use cases are out of scope for this document.

The two aspects of confidential inference

This section includes an overview of the two aspects of confidential inference:

1. **The data aspect**: The inputs and outputs of the model are confidential.
2. **The model aspect**: The model weights and architecture are confidential.

The two aspects are independent of each other, but not mutually exclusive; a confidential inference system may implement either aspect, or both of them. While the confidentiality goals are similar, the characteristics and data paths are different, and require separate designs.

The confidential data aspect

In the confidential data aspect, the **user data** (inputs and outputs of the model) are kept confidential. This may consist of:

- Simple ephemeral input/output pairs (as used in chat assistants, image generators, and other AI applications).
- Input/output data that is retrieved from storage, cache or external sources (as used in generative AI features such as RAG, prompt caching, and MCP).

Confidential inference systems with confidential data may be used to ensure data privacy, and allow the use of AI applications to process PII, PHI, or other private data, even as the data leaves the customer's network to be processed in the service provider's network.

Confidentiality and integrity requirements

From the data owner's point of view, this aspect of confidential computing is similar to requiring an end-to-end-encrypted communication channel with the AI model, where nobody except the data owner and the AI model itself can access (read and/or manipulate) the data. The host which runs the model, including other applications running on this host, should have no way to access the data.

Optional authenticity and containment requirement

As part of the confidential data aspect, it is beneficial to let the data owner have a way to audit the workload accessing and processing their data:

- The data owner should be able to verify the workload's authenticity, confirming it is indeed the AI model inference system and not a deceptive, malicious workload.
- The data owner should be able to audit the flow of their private data within the workload. This auditing capability allows the data owner to ensure that their data is

contained within the AI inference system, processed solely for essential AI inference purposes, and is not saved, copied or logged in a way that allows subsequent access.

With this requirement implemented, the data owner can verify independently that the data is processed confidentially by the AI model (based on the service provider guarantees, detailed below). Without it, the data could still be processed confidentially - but the data owner must explicitly trust the model owner, not just the service provider.

Implementation examples

An example project implementing a confidential inference system with confidential data based on AWS Nitro Enclaves can be found in [7].

An exploratory work implementing a confidential inference system with confidential data based on NVIDIA H100 can be found in [8].

The confidential model aspect

In the confidential model aspect, the **model parameters** (that is, its weights and architecture) are kept confidential. This aspect is beneficial for AI model manufacturers/owners, and they can use the confidential inference system in order to:

- Deploy their AI models to cloud service providers without sharing the model with the service provider.
- Decrease network latency and enable real-time AI applications by deploying their AI models to edge service providers without sharing the model with the service provider.
- Deploy their AI models to customers (on-premises / private-cloud deployments) without sharing the model with the customer.

Confidentiality and integrity requirements

The AI model owner requires a way to share the model weights and architecture with the inference service in such a way that only the inference service can have access to them. Other processes running on the host, including the operating system kernel, should not be able to access these parameters.
The confidentiality requirement extends to requiring defenses against efforts to reverse-engineer and extract the model weights and architecture from the deployed model.
Furthermore, for models that demonstrate dangerous capabilities, requiring security level SL4 or above (detailed in the subsection below), the model owner may wish to isolate the

model weights and keep them confidential from their inception (initial pretraining), so they are not directly accessible even to the model owner itself.

Authenticity and containment requirement

It is required that the AI model owner has a way to confirm that the workload accessing the model weights and architecture is the workload that they have authorized to access the weights, and not a malicious workload. They must also be able to audit the data path of the weights in this workload. This way, the model owner can ensure that the weights and architecture are contained within the confidential boundary, and are not duplicated, logged or copied outside of this boundary.

Confidential computing for SL4 and SL5 security levels of model weights

The RAND research report "*Securing AI Model Weights*" [4] defines five security levels for model weights. The top two security levels, SL4 and SL5, are designed for systems that can withstand weight theft attempts by threat actors ranging from leading cyber-capable institutions and state-sponsored groups to the world's most capable nation-states. These security levels are recommended for models that demonstrate dangerous capabilities, such that the risks associated with the theft of their weights are high.

For models that require an SL4 or SL5 security level, the report strongly recommends that confidential computing technologies be used, when available, for securing the model weights. The report requires making the following assurances:

- The TEE includes protections against physical attacks.
- Model weights are encrypted by a key that is confidentially generated and stored within the TEE.
- The code running within the TEE is prespecified, audited, and signed. The audit process ensures that the TEE does not leak the weights, and only uses them directly as necessary in the AI workload.

Service provider guarantees

The service provider makes guarantees that enable the implementation of the confidential AI inference system. These guarantees are critical to ensure that nobody except the data owner (end user) can access the user data, and that nobody except the model owner can access the model weights and architecture. Service provider operators and unauthorized applications (and by extension, attackers who have compromised the hosts or infrastructure running the confidential AI inference system) are therefore restricted from accessing this confidential data.

Three major cloud service providers - AWS, GCP, and Azure - have implementations of confidential computing solutions: AWS Nitro System with Nitro Enclaves, GCP Confidential VMs with Confidential Space, and Azure Confidential VMs. These implementations make some of the guarantees described in this section.

Hardware-based compute (CPU) and memory isolation

At the heart of confidential computing technologies is the isolation of confidential workloads from the other workloads. This is done by isolating a process, a container (or Kubernetes pod), or an entire virtual machine (or Kubernetes node), so that the instructions it executes and the memory it uses are not accessible to other processes, containers, or virtual machines, even by the operating system kernel or hypervisor. This includes protections such as disabling monitoring or debugging facilities, encrypting memory, and protections against side-channel attacks.

Software is not enough for these protections - for example, a software-only kernel module/driver implementing process isolation could be disabled/bypassed via another kernel module. These protections are therefore implemented in hardware, using technologies such as Intel SGX, Intel TDX, AMD SEV/SEV-SNP, and AWS Nitro System.

Secure cryptographic attestation of the running workload

The hardware powering the confidential computing technology can be used to securely attest the code running in the confidential inference system is the expected (and authorized) code. This is done using TPM (Trusted Platform Module) hardware, producing *measurements* of the workload binaries, the underlying operating system, and the low-level boot process itself (known as *measured boot*). These measurements, presented as PCR (Platform Configuration Register) values, are included in an *attestation document* that is cryptographically signed by the hardware, using a hardware-based root of trust. This attestation document is used for validating the running workload.

The service provider must guarantee that the hardware-based root of trust is not compromised (e.g. its private key is not leaked), and that it correctly measures and signs the PCR values, so that the attestation document cannot be forged by an attacker.

Trusted Execution Environment (TEE)

The compute and memory isolation and encryption, together with secure attestation, are used in order to form a *trusted execution environment* (TEE), sometimes referred to as a *secure enclave* or a *confidential boundary*, in which data can be processed confidentially. The service provider may provide methods to decrypt confidential data only within the TEE. One way to do it is by introducing KMS policies, allowing decryption only when the request comes from within the TEE, which can be verified using the attestation procedure. Another way is to generate the encryption keys (possibly ephemeral keys) within the TEE.

The relationship between the TEE and AI accelerators

As part of AI inference, data which is considered confidential (user data, model weights, model architecture) needs to be transferred from the CPU to **AI accelerators** (GPUs/NPUs) and back. This is detailed further in the next section.

Restricted operator access

Both the model owner and the data owner benefit from reduced access by service provider operators to the decrypted confidential data and parameters, and the service provider can (optionally) make guarantees in this area.

One possibility for restricting operator access is avoiding installation of control programs ("agents") inside virtual machines and cloud-hosted services, and leaving the entire data plane to the service provider's customer (the model owner).

Typically with cloud service providers, even if there is restricted access by operators to the data plane (e.g. remotely connecting to virtual machines or other cloud-hosted services), there is still a control plane: A set of administration interfaces, where these services can be shut down, duplicated, or reconfigured, including security policy reconfiguration. One more possibility for restricting operator access is restricting operator access to this control plane as well.

An example implementation of restricted operator access can be found in AWS Nitro System, which is designed to prevent operator access entirely - see [9]. This guarantee is also included in the Service Terms for AWS Nitro System. It can be difficult for parties

outside the service provider to verify the implementation of restricted operator access controls; in the case of AWS Nitro System, this was verified by a third party - see [10].

Trusted execution environments with AI accelerators

AI accelerators (GPUs/NPUs) are essential, critical components of training and inference systems for many modern AI models. They are physically separate from the main CPU; therefore, care must be taken to ensure that the confidential boundary extends to the accelerator. The main issue is that, as part of inference, data which is considered confidential (user data, model weights, model architecture) needs to be transferred between the CPU and the accelerators. A second issue is debugging features (such as memory dumps) and measuring features (such as performance counters) which may be used to circumvent the confidential boundary on the accelerator.

Note that AI inference systems might make use of an array of multiple AI accelerators that share the workload, connected to each other via high-speed connections (such as NVLink, InfiniBand, or others). In this case, the security of these inter-accelerator connections is beyond the scope of this document; for the purpose of this document, the AI accelerator is treated as a discrete hardware device connected to the host.

There are two approaches to including AI accelerators in a confidential inference system:

First approach: Native support within the AI accelerator

In this approach, the accelerator hardware includes features that can be used to implement a TEE, as described above (isolation, encryption and attestation), within the accelerator itself. The accelerator receives encrypted confidential parameters (user data, model weights and architectural parameters), natively decrypts them as needed, and the unencrypted data resides only in the accelerator's private memory; the output is encrypted before it's sent out of the accelerator. Furthermore, the accelerator disables debugging and measurement features for the duration of the computation, in order to provide compute isolation, and includes secure attestation features, in order to allow verification of the computation it performs.

There are two possibilities for how the encrypted confidential parameters are delivered to/from the AI accelerator:

1. **End-to-end parameter transfer (single TEE):** The key used to decrypt/encrypt the confidential data is only accessible within the accelerator TEE: Either it is generated by the accelerator and does not leave it, or it is an outside key (e.g. on a KMS) that


can only be accessed from within the accelerator, based on a policy that validates the accelerator-generated attestation document. End-to-end parameter transfer means that only the accelerator TEE does any data processing; any non-accelerator (CPU) inference system application cannot access the data, and at most, can only function as a "router", blindly transferring encrypted, inaccessible data. Therefore, only the accelerator TEE is required for the confidential inference system, and relevant applications running on the host do not have to run in any TEE.

2. **Host-to-accelerator direct parameter transfer (two TEEs):** The accelerator supports confidential data sharing "locally" (with the CPU), e.g. via an encrypted shared memory buffer. This secure channel is established between the CPU and the accelerator, so the application running on the CPU needs access to the plain data in order to encrypt it in a way that the accelerator can access. Since the data is confidential, this requires an additional TEE on the CPU, in which the confidential data is decrypted; it is then re-encrypted for the accelerator, and sent out of the CPU-based TEE into the accelerator-based TEE for decryption and processing.

An example of AI accelerators with native support for confidential computing is Confidential Computing on NVIDIA H100 GPUs. An exploratory work using this can be found in [8].

Second approach: Bridging the CPU enclave to the accelerator

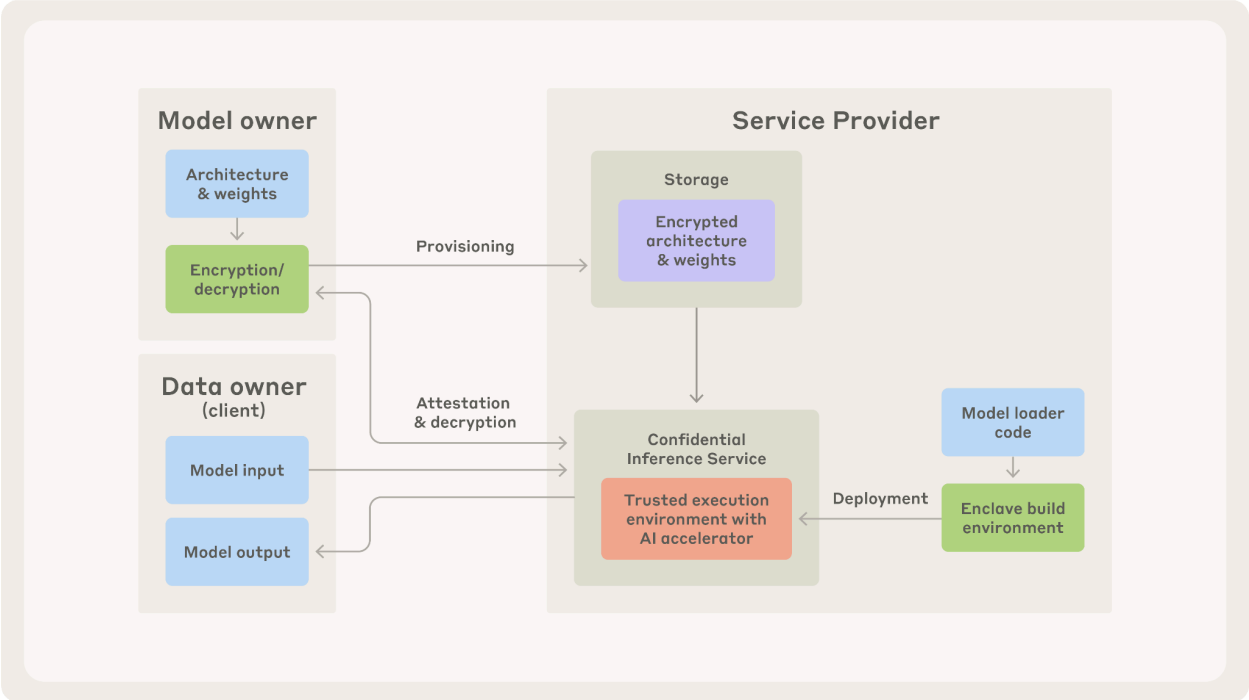
In this approach, the accelerator does not natively support confidential computing (or these features are not used). Instead, the accelerator is bridged or "attached" to the TEE on the CPU, so it becomes a hardware extension to this TEE. Note that this approach is not airtight, and may be susceptible to some attacks, such as certain side-channel attacks, as the delivery of the confidential data from the CPU to the GPU is not encrypted in this approach. Other protection features, like memory encryption, isolation within the accelerator, and disabling debugging/monitoring features on the accelerator are required to form a secure solution.

There are two ways to implement this approach:

1. **Hardware-enforced TEE-only access:** Hardware features in the CPU or hypervisor are used to ensure that only the TEE is able to access the accelerator (e.g. the PCIe bus). These features may include CPU core allocation (only the cores that are allocated to the TEE are allowed to run code which accesses the accelerator) and write-only memory (a memory range in the accelerator that can only be written, where input parameters are loaded, and only the accelerator may read them; and a memory range in the TEE that can only be written, where output parameters are returned, and only the TEE may read them).

2. **The entire VM is the TEE:** If the entire VM is treated as the TEE, then there is no need for special handling - the accelerator is, by definition, inside the confidential boundary. However, this option could introduce additional attack surface (namely, the operating system and the services running on it), which could be avoided with a leaner TEE.

Components and their design principles

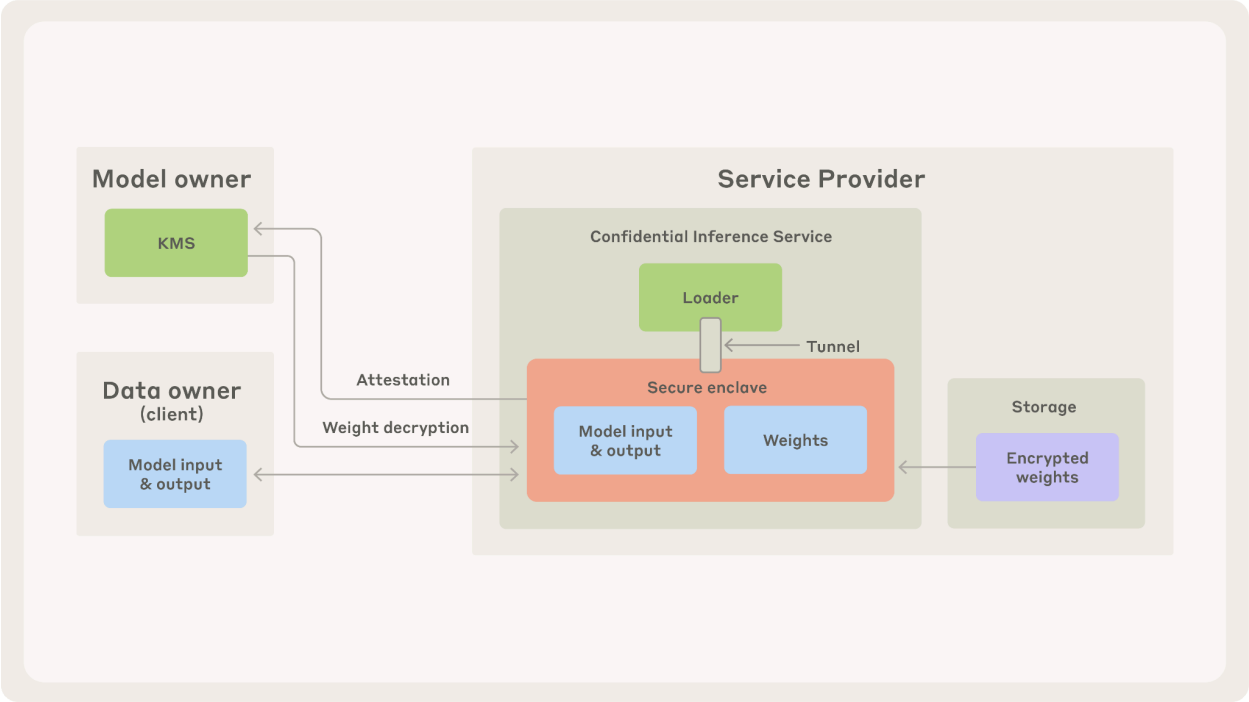


There are three major components of a confidential inference system, illustrated above:

1. Confidential inference service.
2. Model provisioning procedure.
3. Enclave dev & build environment.

Confidential inference service

The **confidential inference service** component is responsible for receiving the model input, weights and architecture, executing the model, and sending out the model output.



A reference design of the confidential inference service consists of a host, owned by the service/compute provider, running two sub-components:

1. **Secure enclave program.** A program running as a secure enclave / trusted execution environment (TEE) on this host, which makes up a "confidential boundary" in which plain (unencrypted) weights and data may be present. The enclave cannot communicate with the outside world apart from a specific predefined tunnel interface for communicating with the host. This program could run either on the CPU or on the AI accelerator; see also the section "Trusted execution environments with AI accelerators" above.
2. **Out-of-enclave programs & communication proxies.** Programs running on this host outside of the TEE, including a "loader" program which is responsible for loading the secure enclave and communicating with it over the specially-defined tunnel. This tunnel is meant for facilitating communication between the enclave and any external resources, such as storage buckets for obtaining the encrypted model weights, the model owner's KMS for weight decryption, and communication of model input & output with the data owner.

Out-of-enclave programs

The programs running directly on the host are owned and operated by the **service provider**. They are responsible for launching the enclave, ensuring that it is a valid enclave

(also known as an *authenticated launch*, implemented using cryptographic attestation) and communicating with it.

The secure enclave

The content (source code and files on the file system) of the enclave image is normally not *confidential* to any party (that is, it is not encrypted at all); rather, the data that it is designed to handle and run computations on is confidential. For the inference use case, the purpose of the image is to be a *model loader and invoker*. That is, it receives and decrypts model parameters (architecture and weights) from the model owner, and model inputs from the data owner; it *loads* the model, and *invokes* it to produce model outputs. The loader and invoker must be developed in a generic manner – agnostic to the model architecture and weights; particularly, it cannot contain any secrets owned by the model owner, because these may easily be revealed by reverse-engineering the image.

Furthermore, it is in the model owner's interest that the model architecture & weights are not leaked by the enclave; and it is in the service provider's and data owner's interests that the model inputs & outputs are not leaked by the enclave. Therefore, although the program running in the secure enclave is capable of decrypting weights and inputs (in fact, it must do this), it has the requirement that it must not do anything with them that is not absolutely required for inference. For example, it must not leak the decrypted weights through the available communication interface.

Note that since the enclave is a loader, the same enclave program may be used for different models. A well-designed enclave program may be usable for a wide range of models, and might not require frequent changes.

Decryption of the model weights and architecture

Outside of the confidential boundary, the model parameters (weights and architecture) are stored encrypted at rest (see also the "*Model provisioning*" section below). It can be assumed that the encrypted parameters can be copied into the secure enclave, where they are ready to be decrypted.

Under this assumption, the question is how to provide the secure enclave with the data decryption key (or, in the common case of envelope encryption, how to unwrap the data encryption key). A reference design of how to do this is as follows:

1. An external KMS holds the data decryption key. (For envelope encryption, it holds the key encryption key, known as "KEK", which can be used to "unwrap" the wrapped data encryption key, known as "DEK").

2. The enclave can communicate with the KMS over the communication channel facilitated by the programs running outside of the TEE. A secure communication channel is established (using a protocol such as TLS).
3. The enclave provides a signed attestation document to the KMS, proving its identity. The attestation document contains a signed ephemeral public key.
4. The KMS verifies the attestation document, and then encrypts the data encryption key with the ephemeral public key and sends it back to the enclave. (For envelope-encrypted weights, the enclave first provides the wrapped data encryption key, and the KMS unwraps it.)

Encryption of the model inputs and outputs

There are two approaches to encryption of the model inputs and outputs:

1. **Encryption termination at the service provider**: The client encrypts the model inputs, and they are decrypted by the service provider, which forwards them to the confidential inference service.
2. **End-to-end encryption (E2EE)**: The client encrypts the model inputs, and they are decrypted only inside the TEE.

(In both approaches, the outputs are handled in the same way, in reverse direction.)

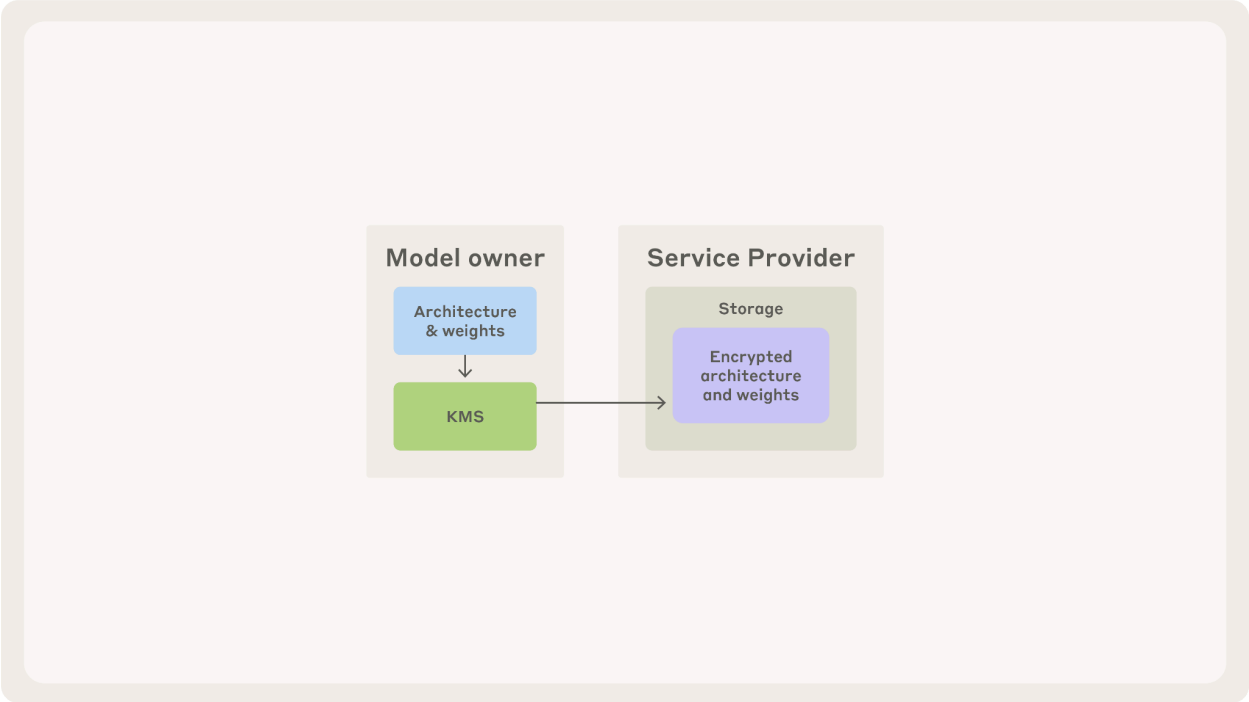
In the first approach, the data owner **shares their data** with the service provider (but not with the model owner), and *does not* implement the confidential data aspect of the confidential inference system. The client sends the model inputs to the service over a secure protocol such as TLS. The service decrypts the model inputs, and simply sends them directly to the TEE. (At this point, there is no point in re-encrypting the model inputs upon entering the enclave, as they are already accessible to the service provider.)

In the second approach, the data owner **does not share their data**, thus implementing the confidential data aspect of the confidential inference system. There are a few options for how to implement this:

1. The client receives an attestation document from the enclave, and validates it. Then, it uses the signed public key from the attestation document in order to send the encrypted data to the service, encrypted (e.g. envelope-encrypted) in a way that only the enclave is able to decrypt it.
2. The client uses a KMS in order to encrypt the data; the KMS is configured with a policy that will only allow the enclave (based on a signed attestation document) to decrypt the data. This method is used, for example, in [7].

Model provisioning procedure

The **model provisioning procedure** component consists of methods to prepare the model for use within the confidential inference system. That is, to encrypt the model so that it may be stored encrypted, and only decrypted within the confidential boundary of the confidential inference service.



In general, as part of the model provisioning procedure, model parameters (architecture and weights) should be encrypted with strong encryption, and the encrypted weights are assumed to be opaque and impossible to decrypt without access to the encryption key. Therefore, the encrypted parameters may be stored at rest with a lower security level than the confidential boundary (e.g. in cloud storage).

KMS-based model encryption

Some advantages of using a key management system (KMS) to encrypt model parameters are:

- Good KMS implementations never leak encryption keys, and strictly limit the operations that can be performed with the keys to a well-defined API (e.g. a “decrypt” API operation). Using a KMS therefore eliminates the inference system’s need to handle encryption keys by itself, and reduces security risks associated with the encryption key handling.


- Moreover, this changes the conditions required for being able to decrypt the model parameters. Instead of "knowing the key", the condition becomes "having authorization in the KMS API policy".
- Some cloud-based KMS implementations have native support for policies authorizing secure enclave access, and these can be used.

There are three possible places that this KMS may reside:

1. Cloud-native KMS, on a cloud account belonging to the model owner, on the same cloud service provider that hosts the inference service.
2. Custom (not cloud-native) KMS, on a machine that belongs to the model owner, possibly on a different cloud service provider (or on-prem host) than the one hosting the inference service.
3. Custom KMS, running as an additional service inside the secure enclave defined above (as part of the confidential inference service component).

The most convenient option is the first option: Cloud-native KMS. This may benefit from native support for authorizing secure enclave access. However, a security tradeoff is made with this choice. Within the threat model of the system, the model owner may choose not to trust the service provider, or its account on it; the potential compromise of both of them are in scope within the threat model. Such a compromise would allow a threat actor to decrypt the model parameters. As an example, consider that an AWS-based model owner's AWS admin account is compromised; then the threat actor can discover the encrypted model weights on an S3 storage bucket, and discover the AWS KMS CMK used to encrypt them. With a simple policy edit, the threat actor can give themselves decryption access on the KMS API, and use it to decrypt the weights.

The second option, a custom KMS running on a host owned by the model owner, eliminates the risk of policy edits by a threat actor. However, it introduces a couple of additional risks:

1. Communication with the KMS now needs to happen over an external network (e.g. the Internet) and this connection must be secured.
2. The KMS must be built securely, including adequate protection of its keys, which may be a challenging engineering task.
3. The KMS must be able to correctly validate secure enclave attestation documents generated by the confidential inference service, which may also be a challenging engineering task having potential security pitfalls.

The third option, a custom KMS running as an additional service inside the confidential inference TEE, eliminates once again the risk of external communication, but retains the challenges of engineering the KMS. The benefit of this is that the encryption key is generated in, and never leaves, the confidential boundary of the TEE. Note that:

- According to the RAND research report “Securing AI Model Weights” [4], models requiring security level SL4 or SL5 need to use TEE-isolated weight encryption keys, effectively requiring this option.
- In some of the current implementations of VM-based secure enclaves, it is not easy to implement a KMS inside a secure enclave if there is no persistent storage or a reproducible source of confidential random number generation available. It is possible to implement a KMS inside an enclave which is always up, using the enclave’s encrypted RAM as secure storage; this introduces further engineering complications.

KMS monitoring and controlled availability

In any of the above approaches, it is important to monitor (log) the use of the KMS to decrypt the model weights, in order to track down illicit uses. Furthermore, for particularly risky models with strictly controlled access, it may be considered to introduce “controlled availability” for the KMS: Keep the KMS offline (“cold”) by default, and connect it to the system (either automatically or manually, depending on the use case) only when the model weights need to actually be decrypted by the secure inference service. This reduces the risk of unexpected out-of-band KMS decryption access attempts by a threat actor.

Optional confidential model provisioning

Normally, a model can be provisioned for the confidential inference system once it is “ready” – that is, once its training / post-training / fine-tuning steps (colloquially, “model finalization steps”) are completed and its architecture and weights are finalized. Assuming these model finalization steps occur outside of a TEE, the model parameters exist in an unencrypted form outside of the TEE at this “pre-provisioning” stage (e.g. in storage owned by the model owner).

In some cases, model weights may need to be confidential even during or immediately after training/finalization, including the model owner themselves being restricted from having access to their own model’s weights. This could be suitable for models requiring SL4 or SL5 security levels (see [4]). This is beyond the scope of the confidential inference system; however, we still mention it here, because it is adjacent, and crucial for the protection of the model weights. There are generally two ways to achieve this:

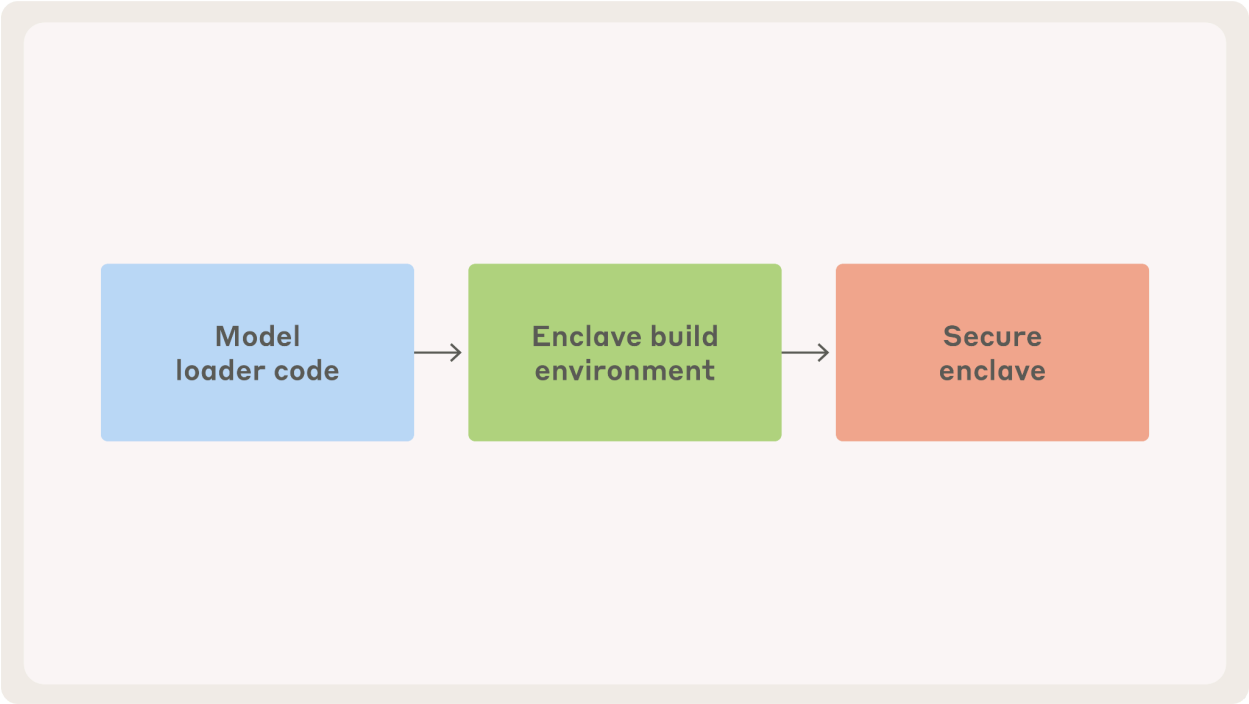
1. **Best-effort secure training and immediate destruction:** Harden the security controls on the servers / workloads that perform the model training, weight finalizations, and confidential inference provisioning. Upon provisioning, the unencrypted model parameters should be destroyed (the tradeoff being that if they

are not destroyed, the existence of their unencrypted copy may form an easier opportunity for their theft).

2. **Confidential training and provisioning:** Perform the model training and finalization steps inside a TEE (the design principles of which are beyond the scope of this document), and provision the model parameters for confidential inference from within this TEE.

Enclave dev & build environment

The **enclave dev & build environment** consists of the environment that needs to be set up to develop and build (and "package", in the terminology used by the Confidential Computing Consortium - see [2]) the secure enclave that is set up and used in the confidential inference service.



In VM-based TEE technologies, the secure enclave must be built, packaged and measured[3] in order to later validate that the expected enclave is running in the TEE. For example, in AWS Nitro Enclaves, this process consists of building a Docker image, and then packaging it to an "EIF" (Enclave Image File) format. In GCP Confidential Space, it consists of building a Docker image, and then signing its measurements.

[3] Measurement is the process of computing hashes over certain properties of the secure enclave. These measurements are included in the cryptographically-signed attestation document that the enclave produces.

Enclave program source code assurance

As explained above, it is in the interest of all parties that the enclave does not leak one party's data or parameters. In order to assure all parties of this, the enclave program source code must be made available for audit by all parties. In other words, program may have one of the following styles of code ownership:

- **Shared-source**, with the source code of the program being co-owned by the model owner and the service provider. This way, the model owner can audit the code and ensure that it does not leak or manipulate the model weights, while the service provider can ensure that the code does not leak or manipulate its clients' data.
- **Open-source**, with the source code of the program being made available for anyone to audit, and maintained by the model owner and the service provider (or possibly a separate, third-party open-source project maintainer). In addition to the above, this also allows the third party – the data owner – to audit the code and ensure that its data is not being leaked or manipulated.

Enclave program binary assurance

Even when all parties are assured, through source code audits, that the enclave program does not leak data or parameters, there is still the possibility that the *compiled binary* does not fully match the source code. This way, additional capabilities (or bugs / vulnerabilities) may be introduced to the image, which allow leak / theft of data or parameters.

One important case where this could happen is the case that the build environment is compromised; the build environment is an extremely fruitful target for executing a supply chain attack, allowing attackers to inject malicious code into the enclave where model weights and data are decrypted. Therefore, the build environment must be hardened against compromise.

Even if the build environment is not compromised, simply having one of the involved parties own and control the build is enough to *break the assurance*. For example, if the model owner builds the enclave, the service provider and data owner cannot be sure that the model owner has not added functionality to reveal the data; similarly, if the service provider builds the enclave, the model owner cannot be sure that there is no added functionality to reveal the model weights.

There are three approaches for achieving binary assurance:

Approach #1: Fully reproducible build process
The enclave program is built using a fully reproducible build process. This way, one party (it does not matter which one) builds the enclave program, and publishes the binary; the other parties also build the program, and compare their result to the published binary. The additional builds are discarded and not used, but the comparison achieves the required binary assurance.

Note that, in many cases, the build process has 3 major build steps:

1. Build the program binary with a compiler (such as gcc) - unless it is written in an interpreted language (such as Python).
2. Build a container image with a container image builder (such as Docker).
3. Package an enclave image with an enclave image packager (such as the AWS Nitro Enclaves builder), or sign its measurements (e.g. the GCP Confidential Space build process).

In a fully reproducible build process, **all steps** of the build process must be reproducible.

An example of an enclave image build system which supports reproducible builds is Bazel, which can be used for building Confidential Space images for GCP. See also [11].

Approach #2: Binary analysis
Lacking a fully reproducible build process, the enclave program binary may be compared with the source code by using binary analysis tools, such as BinDiff, to analyze the compiled binary.

Approach #3: Using a pre-built, pre-assured third-party or open-source binary
In the "third-party" / "open-source" style of ownership, the enclave program is owned by a separate, third-party / open-source "enclave owner". The enclave owner may build the enclave and publish the binary, and then they (or the open-source community) may publish their assurances of the image using methods such as the two described above (a fully reproducible build process, or binary analysis). Note that, especially if relying on the open-source community, care must be taken to ensure that the assurances are legitimate and verifiable.

Third-party content in the enclave image

The enclave image contains third-party content, such as a container base image (e.g. a Docker base image), a language interpreter (e.g. Python), a suite of precompiled code libraries, and others. This content poses risk to the integrity of the enclave program; bugs, vulnerabilities or upstream supply chain compromises of this content could cause the


enclave program to be compromised, or to make it possible to leverage the enclave program for stealing model parameters or data.

In order to reduce the attack surface, the usage of third-party content should be reduced to a minimum. For example, it is advisable to use an empty base image (Docker "`FROM scratch`"), only bringing in the required libraries for loading the enclave; to remove any unnecessary dependencies; and to audit any dependencies that are required and cannot be removed.

Third-party tooling in the build environment

The build environment is likely to contain third-party tooling, such as a compiler, a container image builder, and a suite of shell utilities. Care must be taken when setting up the build environment that these tools are vetted, as any bugs or upstream supply chain compromise of these tools could lead to a compromise of the built enclave image.

One example of a build environment that is not set up securely can be found in the Black Hat USA 2016 talk "SGX *Secure Enclaves in Practice: Security and Crypto Review*" [5], in which it was demonstrated that enclave build utilities were downloaded from the vendor website over an insecure plain-http connection, providing attackers having sufficient access to the network with the opportunity to inject malware into the downloaded build tools.

Threat model, risks and attack scenarios

This section includes some of the threats involved with the design of confidential inference systems. While not a comprehensive security review of a complete design or implementation, it goes over the major threats. Other, more specific security reviews exist, such as Google's overview of Confidential Space [12], and they complement this section.

The confidential inference threat model

The threats considered in confidential computing are threats to the integrity of the code and confidentiality of the data inside the TEE, where the threat actors are assumed to have control over everything outside of the TEE. This assumption can be realized through a cyber attack or insider threat; additionally, in the multi-party context described here (e.g. a model owner and a separate data owner), this assumption is realized through each party not trusting the other parties, and requiring assurances that the other parties cannot access their data.

The general threat model for confidential computing is described by the Confidential Computing Consortium ([1], section 5) as such:

> *Confidential Computing aims to reduce the ability for the owner/operator/pwner of a platform to access data and code inside TEEs sufficiently such that this path is not an economically or logically viable attack during execution.*

Therefore, within the threat model of confidential inference systems:

- The threat actor can be assumed to have complete control over any machine / VM associated with the inference system, including machines where a secure enclave is loaded (in the terms used in this document - the Confidential Inference Service out-of-enclave programs).
- The threat actor can be assumed to have complete control over any cloud or on-prem network resources associated with the inference system, including storage buckets where encrypted weights are stored.
- For the purpose of confidentiality assurance, it can be assumed that each party does not trust the other parties; in other words, each party assumes the other parties to be "threat actors", in the sense that having them gain access to confidential data is considered a threat within the threat model.

Regarding the three major components of confidential inference as described in this document:


- **The confidential inference service** is the backbone of the system. This is the component holding the TEE in which the model architecture, weights, inputs and outputs are decrypted and/or held unencrypted, and is a prime target for attack.
- **The model provisioning procedure** is where the model architecture and weights are encrypted, and stored/downloaded in their encrypted form. This is a potentially hot target for attacks, particularly cryptographic attacks.
- **The enclave build environment** is a lucrative target for supply-chain attacks, because controlling this environment gives a threat actor easier access to the TEE holding the unencrypted model parameters and data.

Systemic vs. introduced security risks

There are two different kinds of risks:

- **Systemic risks** - risks which are inherent in the confidential computing system components that are relied upon by the inference system. Some, but not all, of these risks may be considered to be out-of-scope in the confidential inference threat model.

  In order to mitigate these risks, the inference system implementers need to be on the alert for publicly disclosed vulnerabilities and/or publicly uncovered cyber operations, and install any patches in a timely manner. Moreover, according to the RAND research report “Securing AI Model Weights” [4], for models requiring the top security level (SL5), it is recommended that the model owners employ a team of experts who proactively search for zero-day vulnerabilities in components related to securing model weights; this includes all confidential computing system components, and is a way to further reduce these systemic risks.

- **Introduced risks** - risks which are introduced by the implementation of the confidential inference system. These risks are related to insufficiently secure design, implementation or configuration of the inference system itself, even assuming security of the individual confidential computing components that it comprises.

  To mitigate these risks, the inference system implementers need to ensure the system is implemented securely, practice defense in depth, and audit the security of the system through security reviews and red-team engagements.

Systemic risks

Flaw in the implementation of the secure enclave, including the hardware and software powering it. Confidential inference relies heavily on the security of the secure enclave.

- Flaws in the hardware implementation of VM-based CPU and memory isolation. This includes CPU cache / side channel attacks on the memory isolation technology, and also includes physical access attacks, such as cold boot attacks.
- Flaws in the kernel driver for the secure enclave (e.g. "nsm.ko" for the AWS Nitro Secure Module) which is responsible for communicating with the hypervisor from within the enclave and requesting an attestation document from it.
- Flaws in the base enclave software (e.g. the Nitro Secure Module software components, such as the "nitro-cli" enclave loader and "vsock-proxy").
- Flaws in the hypervisor, which allow circumventing isolation (e.g. with a guest-to-host escape vulnerability or a guest-to-guest lateral movement vulnerability) or forging cryptographic attestation signatures.

Flaw in the AI accelerator's confidential computing technologies. Inference requires AI accelerators, and they must properly be included within the confidential boundary.

- Flaws in the hardware, firmware, or driver implementation of NPU and memory isolation in the AI accelerator. This includes the ability to read confidential memory regions via the CPU, to enable debugging features (breakpoint, monitoring, counters, etc.) on the NPU, to use side-channel methods to discover the contents of isolated memory, or to sideload additional NPU programs which can access confidential memory from within the NPU.

Vendor-side supply-chain risks. The confidential inference system is full of components (both hardware and software) that originate from third parties, such as hardware secure modules and hypervisors. There is a risk that these components, as provided by the vendor, include malicious functionality due to a vendor-targeted supply-chain attack (such as a threat actor manipulating the source code in the vendor's source control, or replacing/tampering with hardware before its arrival and installation at a datacenter).

- All the examples described above (flaws in the implementation of the secure enclave, or the AI accelerator) also apply here, except in this case, the flaws are planted.

Cryptographic risks. Many components and subcomponents of the confidential inference system, and confidential computing in general, rely on cryptography for their security.

- Breakthroughs in the cryptanalysis of the algorithms used to encrypt the weights, to wrap the weights key, to encrypt the model inputs and outputs, to encrypt the memory, to sign the attestation document, etc. If algorithms such as AES-GCM, SHA-384, or others used in confidential inference systems are cryptographically broken and a feasible attack can be mounted against them, this can seriously impact the security of these systems. This includes mathematical or algorithmic breakthroughs, breakthroughs in the availability of computing power, and breakthroughs in the availability of quantum computers.

- The cryptographic algorithms and protocols are misused or have implementation flaws, which can lead to feasible attacks. For example, if the attestation signature is incorrectly validated by a KMS, an attacker could replace the enclave image with a malicious one and pass attestation. Similarly, if the attestation signature uses a hash function which is susceptible to collisions, an attacker could mount an attack against the signature scheme and forge signatures.
- The attestation PKI[4] is compromised (e.g. the hypervisor private key is stolen or broken) so that attestation document signatures can be forged by an attacker.

Introduced risks

Risks introduced in the confidential inference service

Insecure handling of the secure enclave. There are various ways in which an implementer can mishandle the secure enclave.

- When the enclave is loaded, it should be ensured that it is a valid enclave (this is known as an *authenticated launch*). Failure to do so could allow the attacker to replace the enclave with a malicious enclave, which may attempt to intercept confidential data.
- If the system is implemented in such a way that the enclave can be run in "debug mode" then the enclave running in this mode can be monitored or debugged by an attacker. Furthermore, if it is possible to decrypt parameters (such as model weights) inside a debug-mode enclave, then it is probably possible to decrypt them outside of the TEE as well.

Code flaws or security design flaws in the secure enclave program. The secure enclave program may have bugs or insecure design.

- Bugs in the parameter or data parsing in the enclave program could lead to enclave compromise (arbitrary code execution in the enclave) or confidential memory exfiltration.
- Data parsing bugs in any client running inside the enclave (such as a KMS client, TLS client, DNS client, and others) could be attempted to be exploited by an attacker running as a server on the host.
- Unnecessary open ports, software, or dependencies in the enclave image increase the attack surface and could also lead to enclave compromise.
- Any flaw in backend software installed and running in the enclave endangers the security of the enclave. This includes the Linux kernel, Docker itself, base Docker images ("FROM"), core libraries such as PyTorch or TensorFlow, and others.

[4] Public-key infrastructure.

Interception of traffic to/from the secure enclave. In the confidential inference service, an attacker is well-positioned to intercept and/or manipulate traffic to/from the secure enclave (known as a MITM[5] attack).

- If some of the communication protocols to/from the enclave are unencrypted, they are potential targets for MITM attacks; additionally, the unencrypted portions (such as handshakes) of encrypted protocols could be attacked. Particularly, plain DNS could be attacked, and various TLS attacks could be attempted.
- Combined with possible bugs in the enclave TLS client (such as failing to validate certificates) or a PKI compromise, TLS MITM could be attempted. This is a potentially devastating attack; it could be used, for example, to intercept and manipulate the communication of the enclave with an external KMS.

Overly permissive policies for access to/from the secure enclave. The principle of least privilege dictates that access to/from the secure enclave should be confined to the minimum necessary.

- For example, in AWS Nitro Enclaves, the policy file "vsock-proxy.yaml" defining an allow list of services that the enclave may access might be overly permissive, or not kept up-to-date as the enclave image is updated with new versions of the enclave program.

Mishandling of the AI accelerator data loading mechanism.

- If protection features like memory encryption/isolation, locking access to a specific CPU, disabling debugging/monitoring features on the accelerator, and so on, are not properly enabled by the secure enclave, then this could lead to compromise of confidential data while it is on the NPU.

Risks introduced in the model provisioning procedure

Insecure protection of KMS decryption access. Access to the decryption API of the KMS controlling the encryption of model architecture & weights must be securely protected to prevent threat actors decrypting the model.

- If the KMS decryption access policy is misconfigured, and there is a rule allowing a decryption request coming from somewhere other than the secure enclave, then the architecture & weights can be decrypted by that source.
- Even having a correctly-configured KMS policy, if the policy regarding who is allowed to edit the KMS decryption access policy is overly permissive, then whoever is allowed to edit the KMS policy can technically decrypt the model architecture & weights.

[5] Man-in-the-middle, or machine-in-the-middle.


- If any user allowed to access the KMS for decryption or edit the KMS decryption access policy is compromised (for example through a social engineering attack, workstation compromise, API token leak, or insider threat recruitment), then threat actors can leverage this access to decrypt the model architecture & weights.
- If the cloud account hosting the KMS is compromised, or the entire cloud service provider holding that account is compromised, then this could similarly lead to KMS decryption access.

Missing or insufficient KMS decryption access policy updates. From time to time, the KMS decryption access policy needs to be updated. Failure to do so can undermine security.

- When a secure enclave is decommissioned, its access to KMS decryption must be removed to avoid violating the principle of least privilege.
- In particular, if a secure enclave is decommissioned due to the discovery of a security flaw, extra care must be taken to remove its access to KMS decryption. Such cases could introduce devastating BYOVE – "Bring Your Own Vulnerable Enclave" attacks (see also [6]), where the threat actors leverage a decommissioned, vulnerable enclave in order to run arbitrary code which has access to the decryption of confidential model parameters.

Assurance-breaking KMS placement. The KMS controlling the encryption of model architecture & weights is a critical component in assuring the model owner of the confidentiality of the model.

- If the KMS is a cloud-native KMS, the service provider essentially has access to it – both physical access in the datacenter, and regular access through the network. They could bypass security policies and use the KMS to decrypt weights, breaking the confidential inference assurance that the service provider cannot access confidential data belonging to the model owner.
- Moreover, any placement of this KMS in the control of any party other than the model owner, protected only by software-controlled policies, breaks this assurance, as the KMS host is technically able to decrypt weights if they manage to obtain the encrypted weights.

Insufficient key rotation or excessive key reuse.

- If a KMS key is compromised, then any data it was used to encrypt may be compromised. To reduce the risk, the same KMS key should not be used to protect too many copies of the model.
- If the same key is used to encrypt too much data, it is susceptible to surpassing the usage limits of the particular cipher or cipher mode of operation. In particular, weights are very large, so this may be more likely to happen in the confidential

inference use case than in other common encryption use cases, especially if many different models' weights are encrypted with the same key (or, with many cipher modes, even if the same weights are re-encrypted).

Theft of encrypted model architecture & weights. Although they are encrypted, the theft of encrypted parameters (e.g. from a storage bucket) still has security implications.

- Long-term stored encrypted model parameters are susceptible to "harvest now, decrypt later" attacks, where the threat actor steals the encrypted parameters and stores them until a way is developed to decrypt them. Some such ways are future compromises of the KMS, or breakthroughs in feasible cryptanalysis, attainable compute power, or availability of quantum computers.

Risks introduced in the dev & build environment

Compromised enclave build environment.

- If the environment being used to build the enclave image is compromised, it can be used to slip malicious code into the enclave as it's being built.
- If the setup of the environment (including downloading of tools from third parties) can be tampered with, a threat actor can force building a compromised build environment.
- When working with build instance templates/snapshots, care must be taken to avoid malware existing in the environment while the template/snapshot is generated.

Compromised enclave dev environment.

- If the workstations being used to develop the enclave program are compromised, they can be used to slip malicious code into the enclave program as it's being written.

Assurance-breaking build procedures.

- If one of the confidential inference parties builds the enclave image, this may break the assurance to the other parties that the enclave is safe to use. This is discussed in detail above, in the section about the dev & build environment.

References

1. Confidential Computing Consortium. (2022, November). *A Technical Analysis of Confidential Computing v1.3*. https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.3_unlocked.pdf
2. Confidential Computing Consortium. (2022, December). *Common Terminology for Confidential Computing*. https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/Common-Terminology-for-Confidential-Computing.pdf
3. Confidential Computing Consortium. (2025, January). *Confidential Computing Messaging Guide*. https://drive.google.com/file/d/13Pdz2x8mvkjkhxT6nyk8iZdovcN8KNyo/view
4. Nevo, S., Lahav, D., Karpur, A., Bar-On, Y., Bradley, H. A., & Alstott, J. (2024). *Securing AI Model Weights: Preventing Theft and Misuse of Frontier Models*. Research Reports, RAND. https://www.rand.org/pubs/research_reports/RRA2849-1.html
5. Aumasson, J. P. & Merino, L. (2016). *SGX Secure Enclaves in Practice: Security and Crypto Review*. Black Hat USA 2016. https://www.youtube.com/watch?v=0ZVFy4Qsryc
6. David, O. (2025). *Abusing VBS Enclaves to Create Evasive Malware*. Akamai Security Research Blog. https://www.akamai.com/blog/security-research/2025-february-abusing-vbs-enclaves-evasive-malware
7. AWS Samples (2024). *AWS Nitro Enclave Large Language Models*. https://github.com/aws-samples/aws-nitro-enclaves-llm/
8. Biderman, D., Narayan, A., & Ré, C. (2025). *Mind the Trust Gap: Fast, Private Local-to-Cloud LLM Chat*. Hazy Research Blog. https://hazyresearch.stanford.edu/blog/2025-05-12-security
9. Amazon Web Services (2024). *The Security Design of the AWS Nitro System: No AWS operator access*. AWS Whitepapers. https://docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/no-aws-operator-access.html
10. NCC Group (2023). *Public Report - AWS Nitro System API & Security Claims*. https://www.nccgroup.com/us/research-blog/public-report-aws-nitro-system-api-security-claims/
11. Google Cloud (2025). *Create and customize workloads - Confidential Space*. Google Cloud Documentation. https://cloud.google.com/confidential-computing/confidential-space/docs/create-customize-workloads
12. Google Cloud. *Confidential Space security overview*. Google Cloud Documentation. https://cloud.google.com/docs/security/confidential-space