BlindLlama: Confidential and transparent AI using TPMs

Mithril Security

Authors: Corentin Lauverjat, Laura Yie, Shannon Dsouza, Daniel Huynh, Yassine Bargach

23.08.2023

Table of contents

Introduction

Our goal

> The goal of BlindLlama is to make AI confidential, and to be able to prove it. BlindLlama is a confidential and transparent API serving popular open-source LLMs.

We achieve this with two key steps:

Pre-deployment: We take various security measures to create hardened, isolated environments in which we deploy our APIs. We design the system so that we cannot access this environment and therefore cannot view data sent to our service. Our code is open-source and auditable, to enable transparency and scrutiny over our privacy measures.

On deployment: We then deploy AI models inside those hardened environments, and leverage TPMs to generate a file containing cryptographic proofs that attest that we are serving the expected open-source code and that our admins cannot see user data.

End users can verify that this proof is valid using our client-side SDK. Once the client has verified it is communicating with a genuine hardened verified environment, it sends data for AI inference using attested TLS[1].

Problem statement

While the usage of APIs for Large Language Models (LLMs) such as ChatGPT has skyrocketed in the past year, serious concerns have been raised concerning their security and lack of transparency.

One of these concerns relates to the inadvertent leakage of confidential information, such as trade secrets or sensitive personal information, where models use input data as training data, which users may or may not be aware of. This concern has led to concrete policy changes in the industry. For example, Samsung opted to ban staff usage of ChatGPT following multiple incidents of accidental proprietary data breaches. Concerns over the safety of LLMs have equally been documented in recent research efforts.

A related concern is code integrity. Even where APIs and models are open-source, users cannot verify that they are communicating with a server that hosts the code and model they expect. For instance, malicious backdoors could have been inserted into code by the operator, or a server which is hosting the API might already be compromised. Users have no way of knowing if the API has been compromised.

By creating secure confidential AI APIs and using TPM to measure and attest them, we can address these issues and offer end users significant privacy and security guarantees.

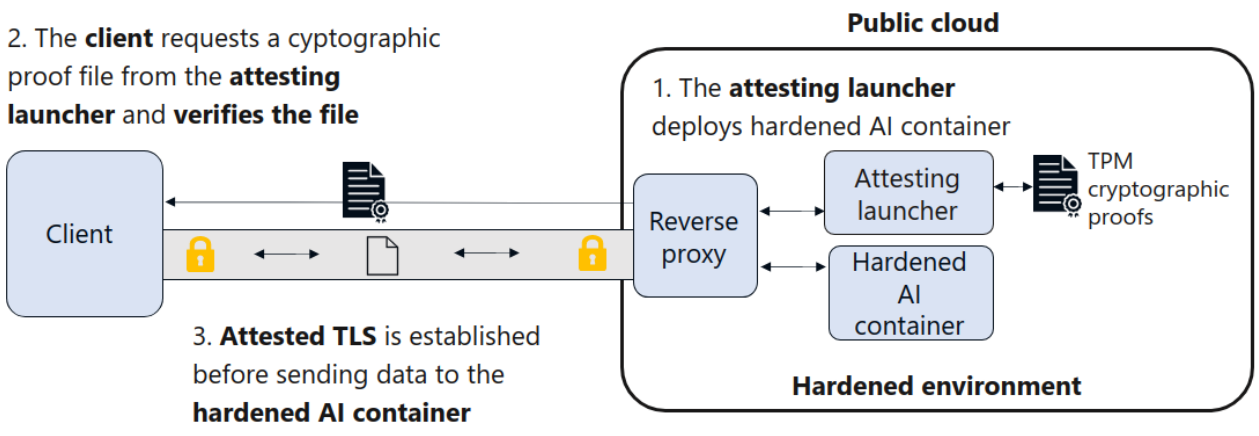
BlindLlama whitepaper

In the following sections, we will introduce the technologies we leverage to create verifiable confidential AI APIs, as well as the challenges we face in doing so and our future goals.

Architecture

BlindLlama is composed of a client-side Python SDK that verifies the remotely hosted confidential API and a server-side architecture made up of:

- A reverse proxy
- An attesting launcher
- A hardened AI container



Public cloud
2. The client requests a cyptographic proof file from the attesting launcher and verifies the file
1. The attesting launcher deploys hardened AI container
Client
Reverse proxy
Attesting launcher
Hardened AI container
TPM cryptographic proofs
3. Attested TLS is established before sending data to the hardened AI container
Hardened environment

Figure 1. Architecture of BlindLlama

Client

The client performs two main tasks:

- **Verifying** that it is communicating with **a genuine BlindLlama hardened environment** serving the expected code
- **Securely sending data** to be analyzed by a remote AI model using attested TLS

Server-side architecture

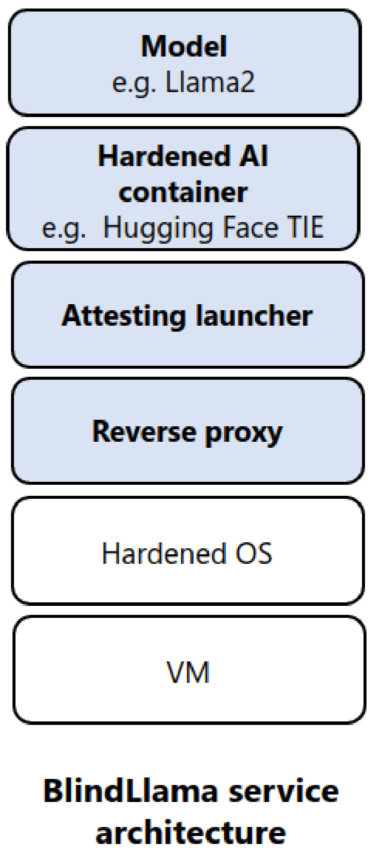


Figure 2. Server-side BlindLlama architecture

Let’s take a quick look at the roles of the three key elements within the server-side architecture.

The **attesting launcher** is responsible for:

- **Loading the hardened AI container** which serves the model
- Creating the **cryptographic proof file** used to attest that we are communicating with a genuine hardened environment

The **hardened AI container** is responsible for:

- Serving the AI model with strict isolation and security measures, which prevent our admins from having access to end user data

The **reverse proxy**:

- **Forwards communications** between the client and the attesting launcher or hardened AI container using **attested TLS**

Trust model

We will explain more precisely which parties have to be trusted when using BlindLlama.

To understand better which components and parties are trusted with BlindLlama, let's start by examining who or what is trusted when dealing with a regular AI service.

Trusted Parties with regular AI providers

In the case of an AI provider serving an AI API to end users on a cloud infrastructure, the parties to be trusted are:

- **the AI provider**: they provide the software application, including the AI model, that is in charge of applying AI models to users' data. Examples in the industry include Hugging Face, OpenAI, Cohere, etc.

- **the cloud provider**: they provide the infrastructure, hypervisor, VMs, and OS, to the AI provider. Examples in the industry include Azure, GCP, AWS, etc.

- **the hardware providers**: they provide the physical components, CPU, GPU, TPMs, etc. to the cloud provider. Examples in the industry include Intel, AMD, Nvidia, etc.

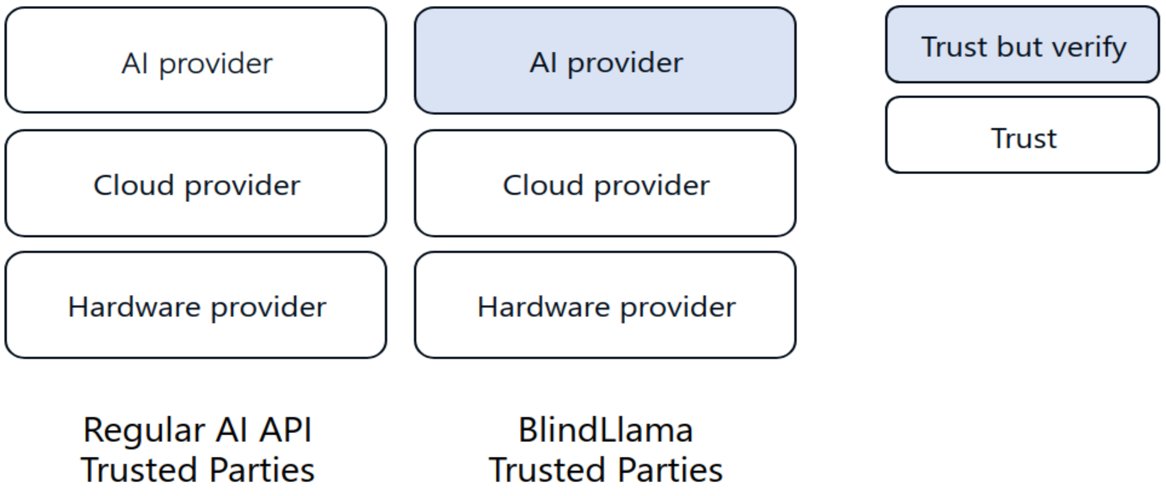


Figure 3. Trusted parties of regular AI API vs BlindLlama API

The higher the party in the stack, the closer they are to the data. Thus the AI provider if malicious or negligent represents the biggest security risk for the user of the API.

In most scenarios today, there is often blind trust in the AI provider, i.e. **we send data to them without any technical guarantees regarding how they will handle or use our data.** For instance, the AI provider could say they just do inference on data, while they could train their models on users' data. And even if most AI providers are honest, there is no way to know if their security practices are strong enough to protect your data.

For privacy-demanding users, this is not enough and they require technical guarantees, so they often elect not to use AI APIs.

Trusted parties with BlindLlama

With BlindLlama APIs, Mithril Security, the AI provider, is removed from the list of trusted parties and cannot see or use end users' data. We can prove such controls are in place using TPM-based attestation.

How we make APIs confidential and transparent

We will present an overview of how we ensure that data sent to our AI APIs remains confidential and how we prove that even our admins cannot see users' data

The key aspect to achieving zero trust is our **secure and transparent systems by design:** our privacy measures by removing all admin access (logs, network, etc.) to our server-side environment (where the hardened container, reverse proxy and attesting launcher reside) can be technically verified using TPMs, providing robust proof that we cannot expose users' data.

We provide a client library to securely consume the AI API using attested TLS: Confidential data is only sent to the AI model once we have verified the use of our hardened AI systems and verified the identity of the hardened environment. Upon successful verification, a secure TLS channel is established between the hardened environment and the client. The hardened environment's private TLS key is only present inside this environment and is not accessible to our admins thanks to the isolation mechanism we have enforced and which can be verified.

Also, the system is optimized for auditability, so that external auditors can completely audit the solution. At the end of this process, users will know that when using our Python SDK, they will be able to verify that they are talking to a privacy-friendly AI service that will not expose their data.

Server side: secure toolchain building

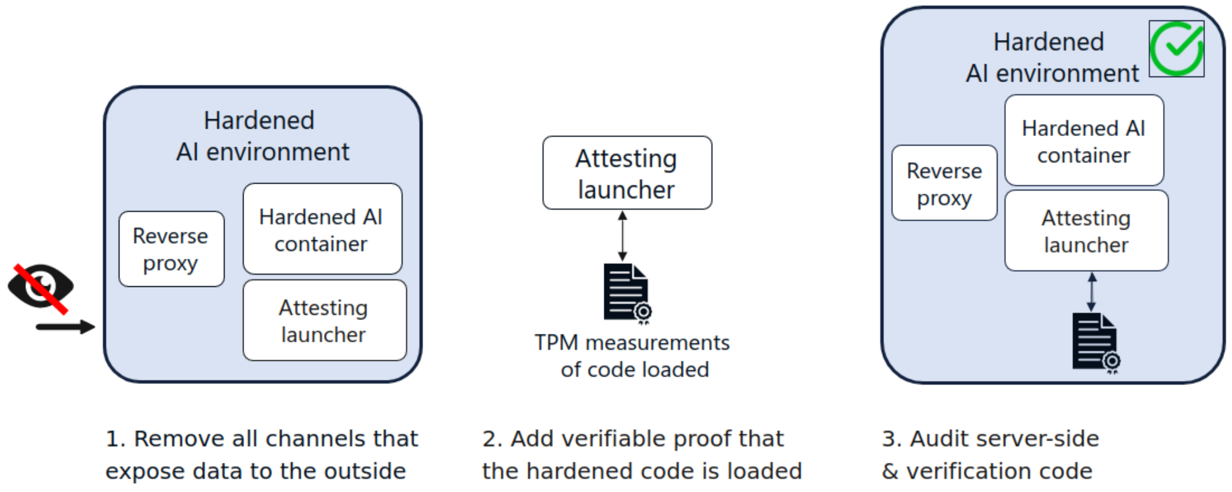

Hardened
AI environment
Reverse proxy
Hardened AI container
Attesting launcher
Attesting launcher
TPM measurements of code loaded
Hardened
AI environment
Reverse proxy
Hardened AI container
Attesting launcher
1. Remove all channels that expose data to the outside
2. Add verifiable proof that the hardened code is loaded
3. Audit server-side & verification code

Figure 4. Three steps of BlindLlama's secure toolchain building

1. Hardening the server-side environment to ensure privacy

The hardened AI container, attesting launcher and reverse proxy are deployed on the Cloud provider's infrastructure - this means the Cloud provider has to be trusted but further isolates the AI provider, Mithril Security, from users' data, since we have no physical access to the infrastructure.

We remove all admin/operator access from the hardened environment, such as SSH access to the VM. This means that once the service is deployed, we are *blind* to all data sent to it. Because we have no access to data, we can neither see it nor use it for any other purposes. This ensures that no insider threat can impact our production services.

We provide more details about **hardened environments** in our concepts guide.

2. Proving privacy controls are applied

While other solutions may also claim to put similar controls in place, they usually provide no technical evidence that AI providers will do what they say they will do. Even where a code base is open-source, users cannot check the server they connect to hosts the application they expect and nothing else.

This point is critical for security and privacy reasons:

- **Privacy:** if users cannot distinguish whether or not an AI provider actually implements proper privacy controls, malicious or negligent AI providers could expose data, and honest AI providers have no way to technically convince hesitant users that their data is actually safe.
- **Security:** integrity of AI systems is key for data traceability and ensures data is only shared securely with trustworthy and untampered systems.

With BlindLlama, we use the Trusted Platform Modules (TPMs), to create a cryptographic proof file which contains measurements relating to the server-side hardened environment's code and stack. Before sending any data, end users verify this evidence using our client-side library to make sure they are talking to a privacy-preserving AI infrastructure.

We provide more information on TPMs in our concepts section.

3. Auditing the whole stack

The security and privacy properties are derived from the system integrity and the system design. Therefore, the trustworthiness of our service is derived from the trustworthiness of our code.

Our code is open-source and we encourage the community to review our codebase. We also have already had one of our similar AI deployment solutions, BlindAI, audited by Quarkslab. This is to provide a high level of confidence that our hardened environment for privacy-friendly AI consumption implements all the security checks we say it does.

Client-side: Secure AI consumption

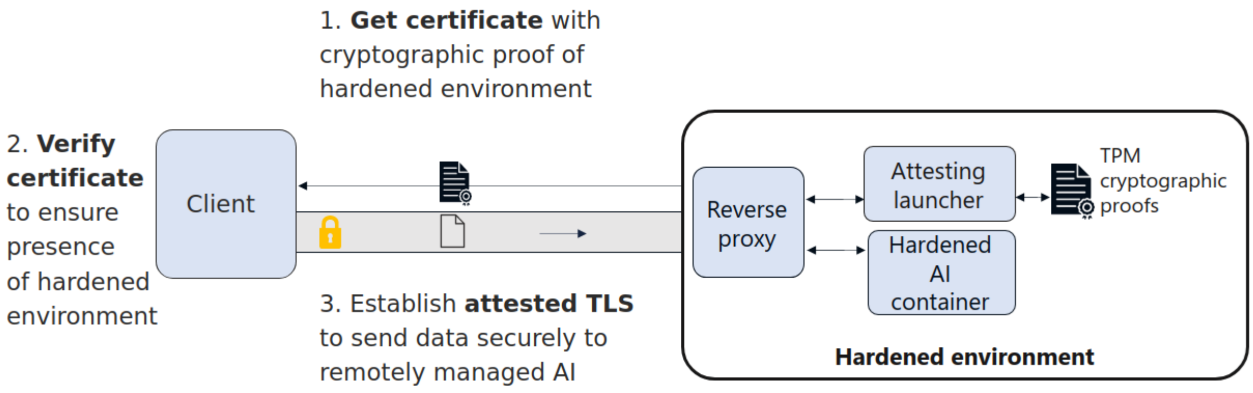

1. **Get certificate** with cryptographic proof of hardened environment
2. **Verify certificate** to ensure presence of hardened environment
Client
3. Establish **attested TLS** to send data securely to remotely managed AI
Reverse proxy
Attesting launcher
Hardened AI container
TPM cryptographic proofs
Hardened environment

Figure 5. The three steps of BlindLlama's secure AI consumption

1. Attestation

The client verifies that they are communicating with a genuine BlindLlama hardened environment where the expected code is loaded.

This is done by verifying a quote that contains cryptographic proof derived from the secure hardware that attests that a specific code base is launched.

We can verify that the code base launched corresponds to the audited/verified codebase of the previous step.

2. Attested TLS

Once the user knows they are talking to a BlindLlama hardened environment, they use a public TLS key bound to the quote. Note that the associated private TLS key lives inside the hardened environment and never leaves it. The TLS certificate is used to initiate a TLS connection which ends at the reverse proxy, which exists within the hardened environment.

Concepts

In this concepts section, we will describe the key concepts that are at the core of BlindLlama. Each section contains an introduction to the concept itself followed by a discussion of how we use these technologies in BlindLlama.

The concepts we will cover are:

- **Hardened systems:** We describe how we harden the BlindLlama server-side environment to eliminate data exposure, even to our admins
- **Trusted Computing Base (TCB):** We detail which elements of our computing base are trusted or attested
- **Trusted Platform Modules (TPMs):** We explain how we use TPMs to provide cryptographic proof that our privacy and security measures are in place

- **Attested TLS:** We describe how we combine secure communications with attested hardened environments to provide robust end-to-end data protection and prove we are communicating with our hardened environment

Hardened systems

What are hardened systems?

Hardened systems operate without special administrative privileges or backdoors. The underlying principle is that once a system comes into contact with confidential data, it should not be possible even for administrators to get privileged access to the system and the system should operate predictably. This is essential for achieving high-security standards, as the security of a system is only as strong as its weakest link, and it is often the humans that are the weakest link in otherwise reasonably secure systems.

Confidential Computing's Trusted Execution Environments (TEE) are ideal for safeguarding data from interference and unauthorized access.

Creating Hardened Environments in BlindLlama:

To ensure data security, we:

1. Modify the OS image and VM configurations to eliminate all system administrator's access. For instance, there is no SSH access to the hardened environment.
2. Limit or remove telemetry, and most logging that are sent to external monitoring services. Note that local-only logs do not pose a threat to confidentiality as long as they never leave the hardened system, so there is no need to disable them.

Note that for enhanced security, we plan to deploy a minimal OS which runs entirely in RAM, rather than using the disk. This is to eliminate the risks associated with disk usage such as confidential data leakage via the filesystem and system integrity corruption.

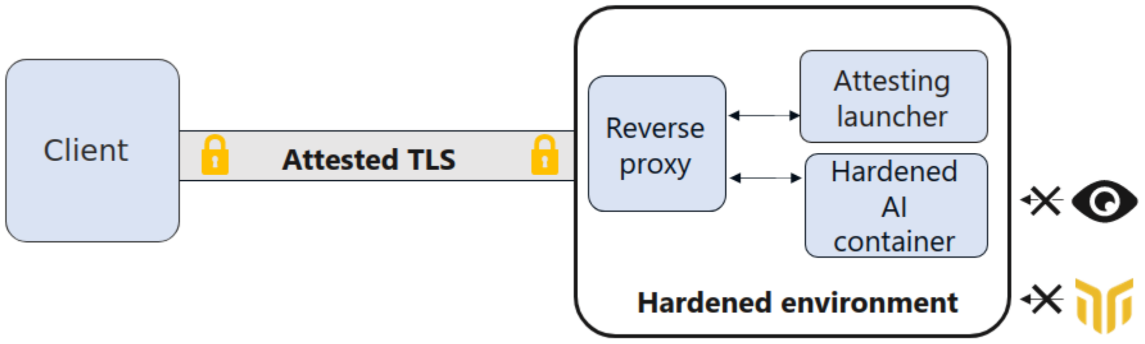


Figure 6. End-to-end data protection with BlindLlama

Security measures and process

In the case of text inference generation, a full hardening and audit of the application layer is done. A pen testing of the text generation inference server is done to:

- Have an overview of the software and its attack surface
- Remove all the possible accesses or controls that represent a threat

Following this, a remediation of any issues found is done.

While hardening the image, memory management is a crucial part of securing the runtime software. Strict isolation of the image is done through configuration by disabling or restricting multiple accesses (such as Inter-Process Communications, shared memory, syscalls, sockets, and so on…). On the network side, only the required ports will be open and used, and all the data in transit will be encrypted through the aTLS (attested TLS) encryption mechanism to ensure the integrity and confidentiality of the data. Furthermore, a thorough review of the web-based APIs included is taken into account.

Access control is also reviewed, as it is a common attack surface. If needed, we limit and restrict users and privileges.

Every verification that goes into delivering the hardened environment will be detailed in a deliverable that will also contain a full auditing report resulting from the pen testing and remediation actions.

Trusted Computing Base (TCB)

What is a TCB?

The Trusted Computing Base (TCB) is the set of all hardware, firmware, and software components that are critical to a system's security and upon which the system's trustworthiness depends.

Since the components of your TCB need to be trusted, they should be chosen so that they are trustworthy.

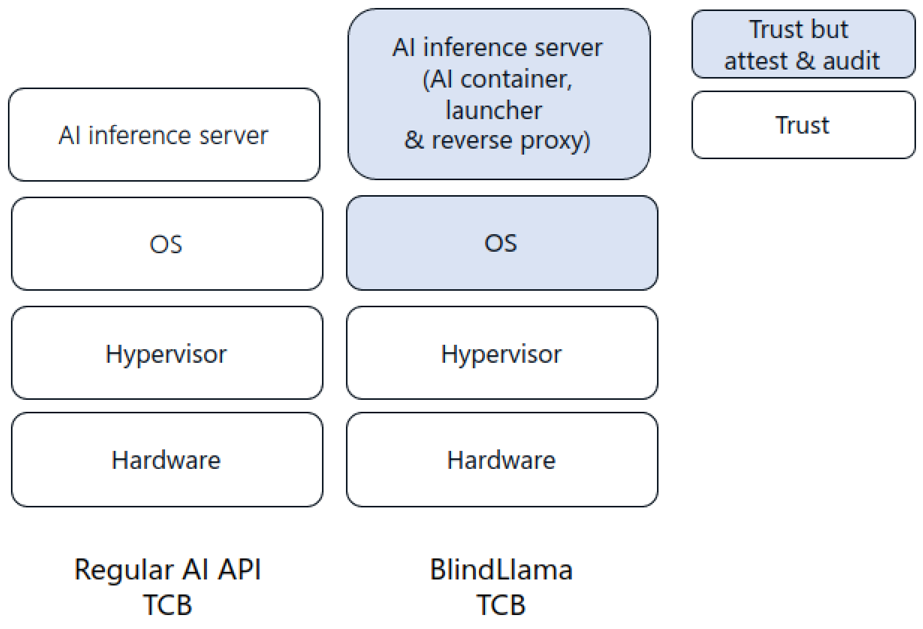


Figure 7. Trusted Computing Base of regular AI API vs BlindLlama

In the context of an AI API deployed in the cloud, the TCB is made of:

- **The hardware:** the physical infrastructure on which everything runs. This includes the actual servers, the networking equipment, storage devices, etc. If there's a hardware vulnerability (like the well-known Meltdown or Spectre vulnerabilities that affected many modern processors), it can undermine the security of everything running on that hardware.
- **The hypervisor** is responsible for creating, managing, and isolating virtual machines (VMs) on a single physical host. If there's a vulnerability in the hypervisor, malicious entities might escape their VM and affect other VMs on the same host. In the cloud, the hypervisor is the responsibility of the cloud provider.
- **The guest operating system** which provides the environment in which the applications run. Vulnerabilities at this level can allow unauthorized access, privilege escalation, or other malicious activities.
- **The application code** and model weights of the AI are the final layers. If there's a vulnerability in the server code or if someone can tamper with the model's weights, they might get undesired outputs, exfiltrate data, or crash the system.

What is the TCB of BlindLlama?

With BlindLlama, like with a typical API set-up, we still have to trust the cloud provider's hardware and its hypervisor. However, we deploy an auditable OS and server-side code, and the client verifies their integrity using vTPM's remote attestation, which we will learn about in the next section.

Trusted Platform Modules (TPMs)

What is a TPM?

TPMs are secure hardware components (usually in the form of a small chip), with built-in cryptographic capabilities and secure storage in the form of Platform Configuration Registers (PCRs). They are generally used to protect secrets such as encryption keys with enhanced security since they cannot be directly accessed or tampered with by the OS. In our case, we make use of the TPM to ensure the integrity of a whole software stack by measuring each component, from the UEFI to the OS, which can then be verified (or *attested*). This process is known as *measured boot*. In addition, we also use TPMs to measure and attest additional data that is needed to reflect security-sensitive events or configurations that can impact the state of the service.

TPMs are provided by all the major cloud providers in the form of virtual TPMs (vTPMs):

- Azure as part of their Trusted Launch VMs [2]
- AWS with NitroTPM & Secure Boot [3]
- Google Cloud Platform as part of their shielded VMs [4]

A virtual Trusted Platform Module (vTPM) is a software-based implementation of a Trusted Platform Module (TPM) chip which is provided by the hypervisor to the guest OS.

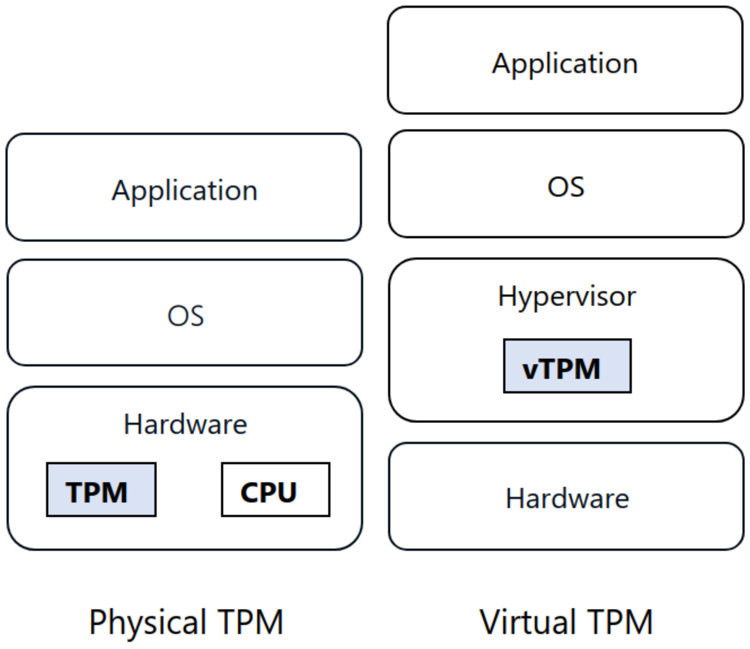


Figure 8. Simple hardware/software stack architecture with physical TPM vs. virtual TPM

How do we use TPMs in BlindLlama?

BlindLlama achieves confidentiality of data by first hardening the environment in which we deploy our API i.e. having it reviewed to ensure it is not leaking data, and then proving such a hardened environment is actually used by verifying the expected server-side code is loaded.

Server side

Measuring the software stack

When the TPM-enabled machine used for service deployment is booted, various default measurements are taken, such as hashes of firmware, boot loaders, and critical system files. These hashes are stored in the TPM's PCRs (Platform Configuration Registers), a set of registers, or locations in memory, within the TPM itself.

The security of those mechanisms relies on the fact that PCRs values cannot be set (or forged) but only extended.

PCR index	Allocation
0-3	UEFI
4-5	Bootloader
6	Sleep mode
7	Secure boot
8	Kernel
9	Initramfs (OS)
10-13	Unused

Figure 9. PCR values measured at boot-time

The BlindLlama attesting launcher uses the remaining registers to measure its application layer along with its configuration, such as:

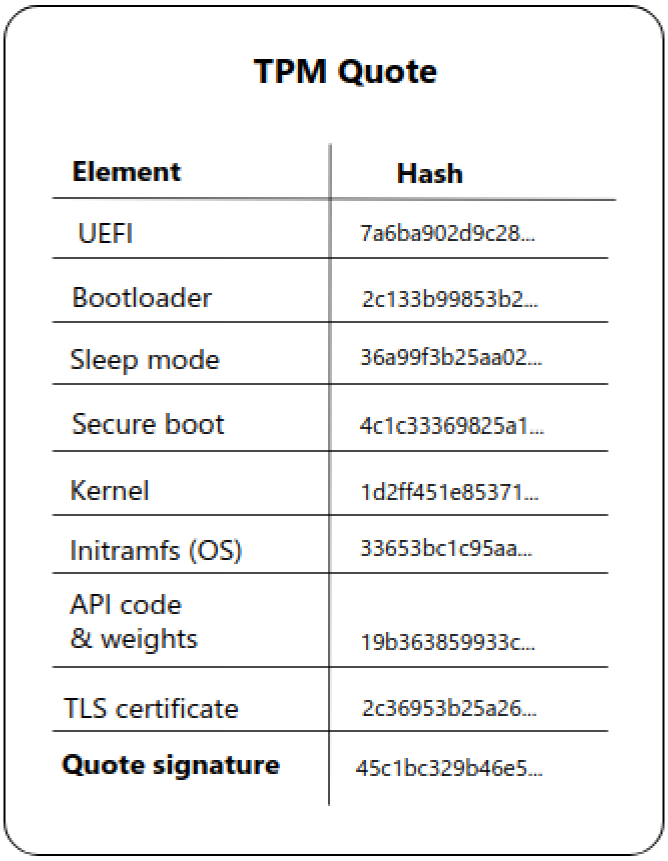
- the hardened container/ API code
- the model weights we serve
- the TLS certificate used for secure communications

PCR index	Allocation
14	API code & weights
15	TLS certificate

Figure 10. Additional PCR values recorded with BlindLlama

Collecting PCR values

The BlindLlama attested launcher requests a signed quote from the TPM which contains these PCR values and is signed by the TPM's Attestation Key (AK). The AK never leaves the TPM and therefore cannot be accessed, even by our administrators.



TPM Quote

Element	Hash
UEFI	7a6ba902d9c28...
Bootloader	2c133b99853b2...
Sleep mode	36a99f3b25aa02...
Secure boot	4c1c33369825a1...
Kernel	1d2ff451e85371...
Initramfs (OS)	33653bc1c95aa...
API code & weights	19b363859933c...
TLS certificate	2c36953b25a26...
Quote signature	45c1bc329b46e5...

Figure 11. Representation of TPM quote values and signature

Creating BlindLlama proof file

The BlindLlama attested launcher includes this TPM's quote in a cryptographic proof file, which additionally contains:

- **Certificate chain:** The certificate chain is an endorsement of the TPM's AK which signs the quote.
- **Event log:** A record of security-sensitive operations.
- Internal CA certificate: This **CA certificate signs the TLS certificate, and is used to verify the attested TLS connection.**

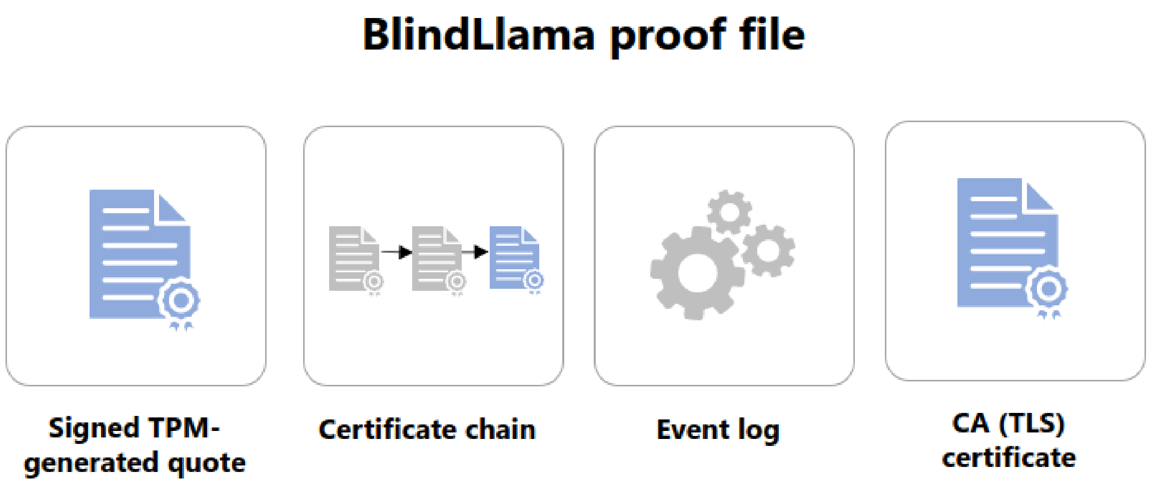


Figure 12. Elements included in BlindLlama proof file

Client side

Verifying the proof file

When the client queries our BlindLlama API before a secure connection can be established the client will receive and verify the cryptographic proof file. The client uses the certificate chain, a chain of certificates that goes up to the cloud provider's root CA. The leaf certificate associated key is the attestation key, which is then to verify the TPM quote’s signature.

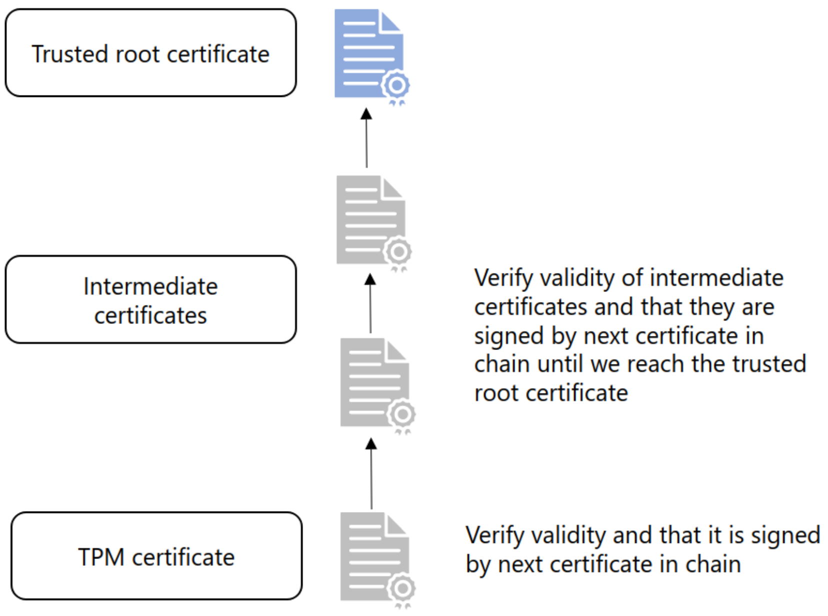

Trusted root certificate
Intermediate certificates
TPM certificate
Verify validity of intermediate certificates and that they are signed by next certificate in chain until we reach the trusted root certificate
Verify validity and that it is signed by next certificate in chain

Figure 13. Verification of the Attestation Key using a certificate chain

The verification is done in three steps:

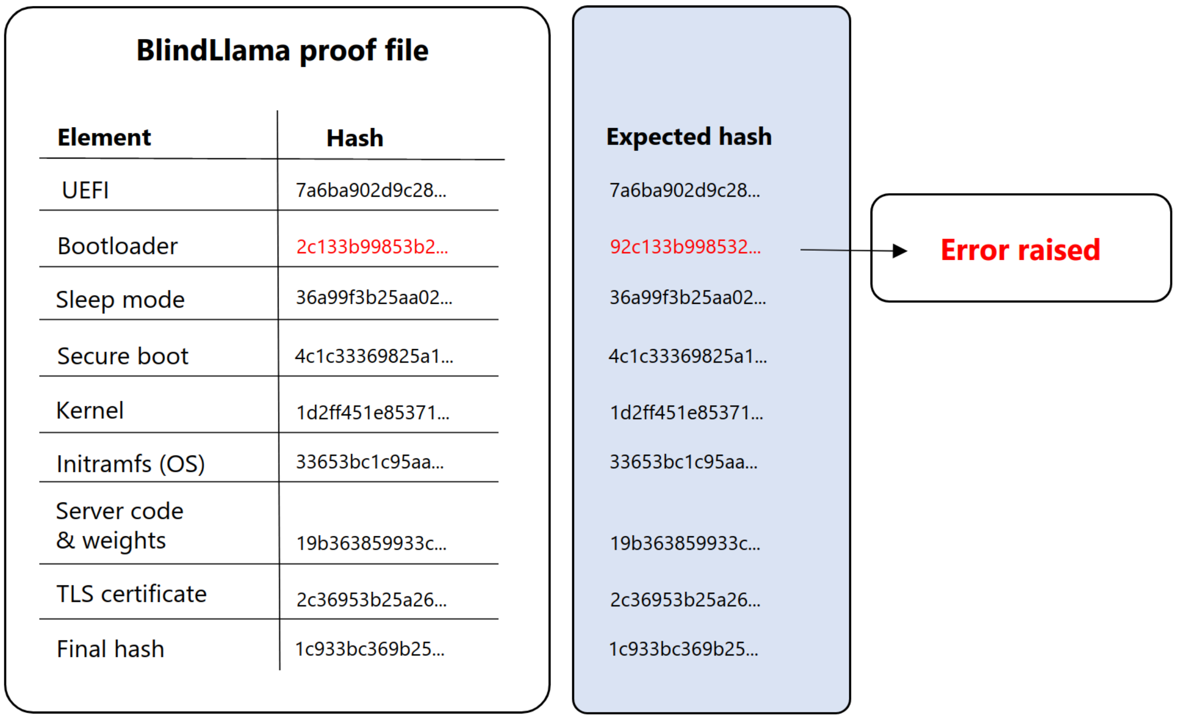
1. Validation of the Attestation Key (AK) certificate:
   - ➔ The client validates the AK using the certificate chain. This certificate chain is issued by the Cloud provider*, and is evidence that the key is really the Attestation Key of one of their TPMs.
2. Quote Verification:
   - ➔ Using the public part of the AK, the client checks the quote authenticity
3. Evaluate if the server-side environment is trustworthy based on the PCR values:
   - ➔ The client recreates the hash values with the help of the event log and verifies these values against the values in the TPM-generated quote (details provided in the next section).

* ⚠️ **Limitation:** Due to current limitations with Azure vTPMs, there is no signed endorsement from Azure regarding the TPM's attestation key. See the "Current Obstacles" section for more details.

Understanding PCR measurements

Once the quote's validity has been verified, the verifier needs to make a trust decision based on the value of the PCRs. This last step is more complex, since PCR values are used to reflect the security state of the system.

PCR values can either be deterministic or non-deterministic.



Element	Hash	Expected hash
UEFI	7a6ba902d9c28...	7a6ba902d9c28...
Bootloader	2c133b99853b2...	92c133b998532...
Sleep mode	36a99f3b25aa02...	36a99f3b25aa02...
Secure boot	4c1c33369825a1...	4c1c33369825a1...
Kernel	1d2ff451e85371...	1d2ff451e85371...
Initramfs (OS)	33653bc1c95aa...	33653bc1c95aa...
Server code & weights	19b363859933c...	19b363859933c...
TLS certificate	2c36953b25a26...	2c36953b25a26...
Final hash	1c933bc369b25...	1c933bc369b25...

Figure 14. Example of verification of BlindLlama proof file with unexpected bootloader value

For deterministic PCR values, every detail of what gets measured is known in advance. Therefore, it's possible to precompute the expected value. Such PCR registers typically measure static components that don't change across boots, such as firmware or certain boot components. To check them, we compare their values against the expected measurement (called a golden measurement). This is straightforward but rigid. The measured component can't be changed without also changing the golden measurement. So, if the PCR value matches the golden measurement, it means that the component is as expected and has not been tampered with.

On the other hand, some PCRs measure dynamic or variable data, like configuration files or the TLS CA certificate of the hardened server-side environment. These registers are typically associated with an event log. Every time an entry is added to the event log, a corresponding measurement is made on a PCR. When attesting the state of a system, the verifier will request both the quote with the PCR values and the event log. By replaying the events from the log and simulating the TPM's Extend operations, the verifier can reconstruct the measurement. The reconstructed measurement is compared to the one from the quote, if they match it ensures the integrity of the event log. Then the client checks if the events align with a security policy. Compared to the deterministic PCR approach, this approach is more flexible as it can accommodate changes in system configuration, updates, or software as long as the event log correctly reflects these changes. It is used for the application layer of BlindLlama.

Attested TLS

What is attested TLS?

Attested TLS combines the security of data in transit with TLS and a proof of the identity of the server-side hardened environment.

In comparison, regular TLS only prevents outside parties from knowing users' information when it is sent to an AI provider, but does not protect from the AI provider itself. Regular TLS leaves the data exposed and accessible once it arrives at the endpoint.

Attested TLS is often deployed in the context of Confidential Computing where a secure TLS connection directs communications to an attested isolated environment, or Trusted Execution Environments.

By binding a TLS certificate to an attested secure environment we protect ourselves against man-in-the-middle (MITM) attacks[5], as we have proof that we are communicating with our attested secure environment and not one that is merely forwarding a quote/attestation report from a valid machine.

How does attested TLS work in BlindLlama?

Let's take a look at how attested TLS works in BlindLlama step-by-step:

Server side

We deploy the BlindLlama service, available through our Mithril Cloud.

On deployment, a TLS-terminating reverse proxy using Caddy is created. The reverse proxy takes care of generating the TLS certificate required for a TLS tunnel. Since this is done within our hardened isolated environment, it remains protected and is not accessible even to our admins. The client connects to this reverse proxy server, which will relay the inbound/outbound communications firstly to/from the attesting launcher to complete the initial attestation process and then to the hardened AI container which serves the AI model.

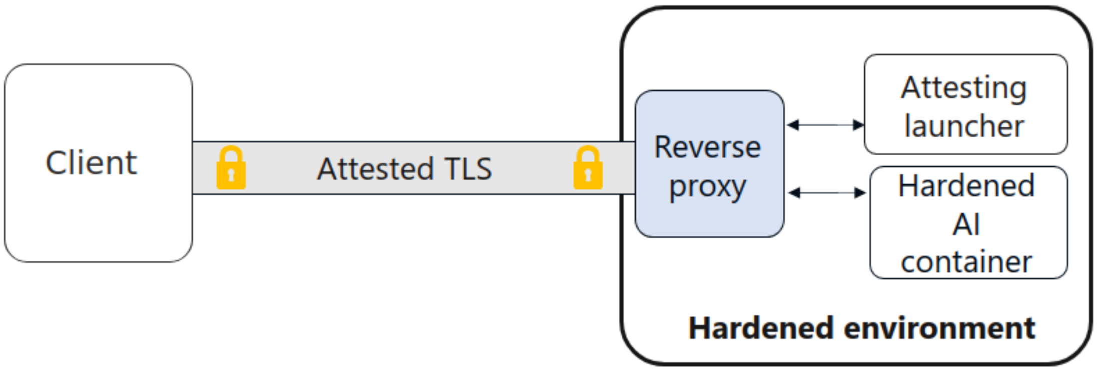


Figure 15. The reverse proxy forwards inbound/outbound communications to/from the client/server-side launcher and AI container

On startup, the caddy-generated TLS certificate is hashed by the BlindLlama attesting launcher and measured in the TPM platform register PCR[15]. For more details about TPMs and PCRs, refer to our section on TPMs.

The attesting launcher generates a proof file that includes a TPM-generated quote with all the hashed values stored in the TPM's PCRs. The hash of the reverse proxy's CA certificate, which is generated with and bound to its TLS certificate, is included in the proof file, which is then shared with clients and is used to set up a TLS session between the reverse proxy and the client.

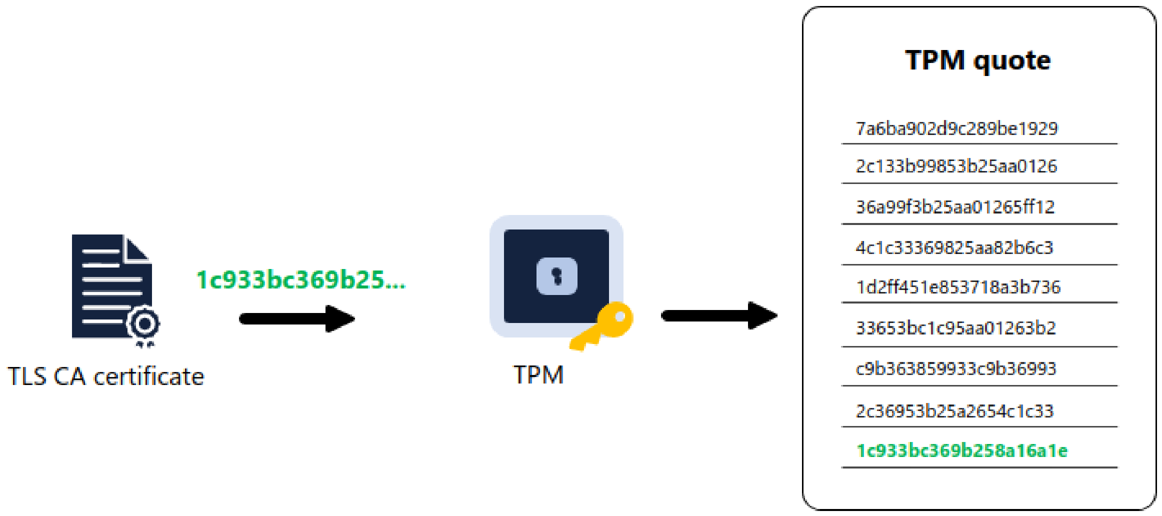


Figure 16. The reverse proxy's CA certificate (bound to its TLS certificate) hash is stored in a PCR register and then included in the BlindLlama proof file

Client side

When the end user first connects to the hardened server-side environment, the client will receive the following assets within the cryptographic proof file:

- **The reverse proxy's CA certificate** from the current connection/session
- **The TPM-generated quote file,** which includes a hash of the reverse proxy's TLS CA certificate in PCR 15
- **A certificate chain** for the TPM AK certificate.
- **An event log** of security-sensitive events

Firstly, the client verifies the TPM quote. The client comes with a built-in root certificate authority from a trusted party such as Azure. The client uses this certificate to verify the TPM certificate chain, which starts with the public TPM AK and ends with the cloud provider's root certificate.

Once the public AK is verified, it can be used to verify the TPM quote that is signed with the TPM's private AK.

Once the quote is validated as having been generated by a genuine TPM, the TLS CA certificate from the current connection is hashed and compared against the reverse proxy's TLS CA certificate in PCR 15 of the quote.

If the TLS CA certificate hash in the quote does not match the hash of the TLS CA certificate of the current connection, the connection will fail and an error is raised.

This prevents **man-in-the-middle attacks**. If a malicious server were to intercept and forward the expected proof file, we would still be alerted to the fact that the server we are communicating with does not have the expected TLS CA certificate hash, and communications with this server would end there.

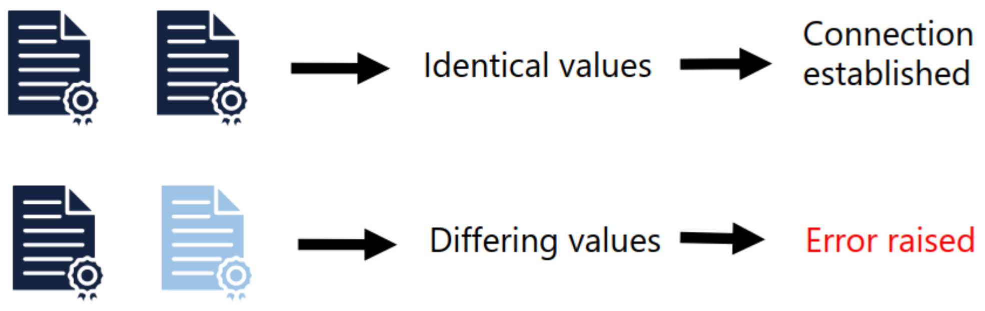


Figure 17. Checking TLS certificates against values in the proof file

As detailed in the previous section, the proof file also contains hashes relating to the stack of the machine the API is deployed on, the API's code, and the model's weights. This means not only are we sure we are connecting to the correct server using TLS but we know that the hardened AI container is serving the expected code and model!

Open source code and build transparency

Open source code means that auditors can inspect the code to ensure that it doesn't contain any malicious backdoors or vulnerabilities. This reinforces trust as users don't have to rely on trusting a "black box".

Build transparency

Build transparency ensures the code you see is the code that runs. To achieve this transparency, we have adopted the Supply-chain Levels for Software Artifacts (SLSA) [6] security framework. SLSA helps ensure the integrity of software artifacts by adding provenance statements to software artifacts. Our goal is to achieve SLSA build level 3, where a trusted builder attests to the build process. This gives the following properties:

- Traceability: With provenance statements, users can trace back to the origins of every software artifact. This ensures that the software being executed is the same as the software that was written, without any modifications in between.
- Authenticity: Non-forgeable provenance statements ensure that the build process and its outputs are genuine. Any tampering or unauthorized changes can be easily detected.
- Separation of Duties using a trusted third party: The trusted builder is a platform like GitHub Actions or Google Cloud Build, not us.

Reproducible Builds

A reproducible build [7] is a process that, given the same source code and build environment, always produces identical binary or output artifacts.

When possible we will also try to achieve reproducible builds. This will provide an even higher level of transparency. This removes the need to trust the builder (GitHub Actions or Google Cloud Build), since anyone can reproduce the software artifacts from the code. Reproducible builds are also mentioned in the SLSA framework as part of build level 4.

Current obstacles

When it comes to TPM (remote) attestation, the process involves:

- The TPM creates a quote, which is a set of measurements (values from Platform Configuration Registers or PCRs) that reflect the state and configuration of the system.
- The quote being signed with a TPM attestation key

The issue: Azure doesn't provide endorsement of the TPM's attestation key.
Without such endorsement from Azure, there's no authoritative way to verify the origin and authenticity of the TPM quotes provided by the service. When using a TPM, the chain of trust is paramount. Without endorsement, this chain remains incomplete in the Azure cloud.

We are in contact with Azure Confidential Computing and Trusted Launch teams, to remedy the situation and we expect the issue to be resolved soon.

Despite TPMs being a standard since 2009, we have found that the issue is not limited to Azure. Other cloud providers providing vTPM on their VMs have similar limitations:

- AWS does not provide endorsement of the TPM (like Azure)
- GCP provides an API endpoint to get the public key of an instance, but they do not provide a certificate. That being said, this limitation can be circumvented. Also, GCP is aware of this limitation and they plan on adding endorsement certificates in the future.

Next steps

Attest a cluster of GPUs

A cluster of GPUs could be either several GPUs on a single VM or several VMs (in the latter case we would need some sort of orchestration platform like Kubernetes)

In the future, we will add the ability to attest a cluster of GPUs.
This may take two approaches depending on the use case.

1. If the GPUs are all attached to the same machine (in the case of a very GPU-intensive computation), we can attest the GPUs (drivers included), by simply adding this information to a PCR register and validating it on the client.
2. If the GPUs are on separate machines, we will need to attest the entire cluster of machines. This would require an orchestration platform such as Kubernetes, we could then have one trusted node that attests every new node and its components by adding their quotes as input to its PCR register. Kubernetes distribution can be developed to take advantage of the TPM as a root of trust.

Bibliography

[1] R. Walther, C. Weinhold, and M. Roitzsch, “RATLS: Integrating Transport Layer Security with Remote Attestation,” in *Applied Cryptography and Network Security Workshops*, J. Zhou, S. Adepu, C. Alcaraz, L. Batina, E. Casalicchio, S. Chattopadhyay, C. Jin, J. Lin, E. Losiouk, S. Majumdar, W. Meng, S. Picek, J. Shao, C. Su, C. Wang, Y. Zhauniarovich, and S. Zonouz, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 361–379. doi: 10.1007/978-3-031-16815-4_20.

[2] “Trusted launch for Azure VMs - Azure Virtual Machines,” May 26, 2023. https://learn.microsoft.com/en-us/azure/virtual-machines/trusted-launch (accessed Aug. 18, 2023).

[3] “Amazon EC2 Now Supports NitroTPM and UEFI Secure Boot | AWS News Blog,” May 11, 2022. https://aws.amazon.com/blogs/aws/amazon-ec2-now-supports-nitrotpm-and-uefi-secure-boot/ (accessed Aug. 18, 2023).

[4] “Shielded VMs,” *Google Cloud*. https://cloud.google.com/shielded-vm (accessed Aug. 18, 2023).

[5] F. Stumpf, O. Tafreschi, P. Roder, and C. Eckert, “A Robust Integrity Reporting Protocol for Remote Attestation”.

[6] “Supply-chain Levels for Software Artifacts,” *SLSA*. https://slsa.dev/ (accessed Aug. 18, 2023).

[7] C. Lamb and S. Zacchiroli, “Reproducible Builds: Increasing the Integrity of Software Supply Chains,” *IEEE Softw.*, vol. 39, no. 2, pp. 62–70, Mar. 2022, doi: 10.1109/MS.2021.3073045.