Private AI Compute in the Cloud – Assessment Overview

Google

Version 1.4 – November 8, 2025

Prepared By
Evan Anderson
Elena Bakos Lang
David Brauchler
Thomas Cannon
Gérald Doussot
Aaron Kondziela
Giacomo Pope
Thomas Pornin
Domen Puncer Kugler
Javed Samuel
Eric Schorn

Prepared For
Google LLC

1 Executive Summary

Synopsis

Starting in the spring of 2025, Google engaged NCC Group to conduct a series of reviews involving selected aspects of their system: Private AI Compute in the cloud. The purpose of the Private AI Compute system is to extend the AI capabilities of a mobile device with more powerful cloud computing resources while aiming for the user data to still have the same privacy guarantees as with local-only computations.

This program of work consisted of several phases:

- **Phase 1**: 21 person-days focused on an architecture review of the Private AI Compute system.
- **Phase 2**: detailed review of selected components of the Private AI Compute system, further broken down into two stages:
  - **Stage 1**: 20 person-days on a cryptography security assessment of the Oak Session Library and the implementation of attestation and encryption between front-end services and Model Serving Component.
  - **Stage 2**: 44 person-days on the security analysis of IP-blinding relay, a cryptography security assessment of the T-Log system, a configuration review of Outbound RPC Enforcement, and a source code review of Private AI Compute frontend server.
- **Phase 3**: 15 person-days dedicated to the creation and review of a public facing summary report.

Phase 1 took place in April-May, while Phase 2 took place in June-September. The program was delivered remotely by ten consultants, with a total effort of 100 person-days.

Scope

NCC Group's evaluation included:

- **Overall system architecture of Private AI Compute**
  - Specifically including: Private AI Compute Frontend server, Encrypted Channels (Oak/Noise and ALTS), Inference Tasks Orchestration Pipeline, AI Safety Modules, Model Serving component, Hardened Tensor Processing Unit (TPU) Platform.
  - Excluded from the scope were: Confidential Computing Platform — Platform based on AMD SEV-SNP, phone apps using Private AI Compute.
- **Attestation and encryption**
  - Project Oak Session Library implementation: This open-source library provides an implementation (git commit `e172cbb6`) of Oak end-to-end encrypted attested sessions between communicating parties.
  - Attestation and encryption between front-end services and Model Serving component implementation: This included communications between Private AI Compute Frontend client and server, supported by the Oak Session protocol, and between the Private AI Compute Frontend server and Model Serving component supported by the ALTS protocol.
- **Selected Private AI Compute components**
  - Private AI Compute Frontend server: source code review.
  - IP-blinding relay: analysis of privacy-preserving IP obfuscation technique used to decouple users submitting queries from the data the service receives.
  - T-Log binary matching/verification: cryptographic implementation review and protocol review of Transparency Log.
  - Crash dump export and admin access: evaluation of system configuration to ensure that no private data is exposed in the event of application crash.

- Outbound RPC Enforcement configuration: validation of access control lists governing outbound system data.

The Google team provided documentation and code pointers, and offered timely support via a dedicated chat.

Limitations

NCC Group noted several limitations in the coverage of a few Phase 2 components in sections Attestation and Encryption Between Frontend Services and Model Serving Component and Private AI Compute Frontend Server, IP-blinding Relay, T-Log, Crash Dump, and Outbound RPC Enforcement. Nevertheless, NCC Group provided reasonable coverage of all components in scope for this program of work, and documented a number of findings.

Key Findings

The assessments uncovered a number of issues across each of the individual projects. The most notable findings included:

- **Timing-Based Side Channels** in the IP-blinding relay component that may be able to unmask target Private AI Compute users when particular conditions are met. Exploitability was marked 'Low'.
- **Denial of Service** in the IP-blinding relay component for legitimate users as a result of malicious applications exhausting victims' certificate quota, and in the `oak_session` library component where an attacker could exhaust the resources of a client/server application that uses the `oak_session` library. The overall risk was rated 'Low' and the concern does not impact privacy.
- **Lack of Full Protocol Transcript** in the `oak_session` library component, which may result in various protocol attacks. The overall risk was rated 'Low' and has undetermined exploitability.

NCC Group reported all findings to Google (see the comprehensive Table of Findings). NCC Group also communicated a number of notes and observations during the projects. Google filed bugs for some of these on internal systems and reported working on fixing them and/or having fixed them to NCC Group.

Following this section, NCC Group includes a description of the Private AI Compute system including its privacy model in the System Architecture Overview section. The Phase 2 security assessments of "Attestation and encryption" and of "Selected Private AI Compute components" used by the system are summarized in the Attestation and Encryption Between Frontend Services and Model Serving Component and Private AI Compute Frontend Server, IP-blinding Relay, T-Log, Crash Dump, and Outbound RPC Enforcement sections respectively.

Although the overall system relies upon proprietary hardware and is centralized on Borg Prime, NCC Group considers that Google has robustly limited the risk of user data being exposed to unexpected processing or outsiders, unless Google, as a whole organization, decides to do so. As explained in the architectural overview below, it is unavoidable that Google ultimately has this power and NCC Group is unaware of any commercially reasonable mitigations not undertaken. Users will benefit from a high level of protection from malicious insiders.

Additional Context from Google

Mitigation of Findings

We acknowledge the findings identified by NCC Group and are actively working on addressing them:

- **Unbounded Session Message Queues & No Limit on Attestation Data Request and Response:** These two findings, which could potentially lead to denial-of-service situations, are being actively addressed.
- **Timing-Based Side-Channel Attacks Can Unmask Users:** We acknowledge the theoretical risk of timing-based side-channel attacks. The system, as it is currently implemented, resists this type of attack due to its multi-user nature, which adds a significant amount of noise and makes it difficult to correlate queries to specific users.

Transparency and Verifiability Roadmap

We believe transparency and verifiability are key pillars of trust. To this end, we have made the underlying binaries for platforms and features like Android, Private Compute Core, confidential federated analytics, and our ML suite available for public inspection and review. As a first step with Private AI Compute, we offer transparency for our deployed binaries by publishing cryptographic digests (e.g. SHA2-256) of those binaries in a published ledger. Digests of application binaries used by Private AI Compute servers are published in the ledger before serving traffic. Within the protected execution environment, system identity is verified as authorized to handle private workloads before any user data is processed. Additionally, users can see when Private AI Compute is being used on their Pixel devices – Private AI Compute requests are visible in their Settings Network Logs.

Building on this foundation, future releases of Private AI Compute plan to include:

- **External inspection of remote attestation verification.** To facilitate verification and mitigate the potential threat of undiscoverable modification of the system, we will be enabling experts to inspect remote attestation evidence from the client and server, and confirm that it only includes binaries that were specifically endorsed by Google for the purpose of Private AI Compute and are inspectable by third parties as described above.
- Continued **third-party audits** and expanded support for code and binary inspectability.
- Expanded **security research programs**. Our Vulnerability Rewards Program will be expanded to specifically include Private AI Compute. We will invite privacy and security experts to investigate our system and reward them for discovering potential vulnerabilities.

Note that additionally, Google is contributing to community efforts on industry standards in this space, starting with an early-stage proposal at the IETF.

2 System Architecture Overview

A simplified representation of the overall Private AI Compute system is shown below:

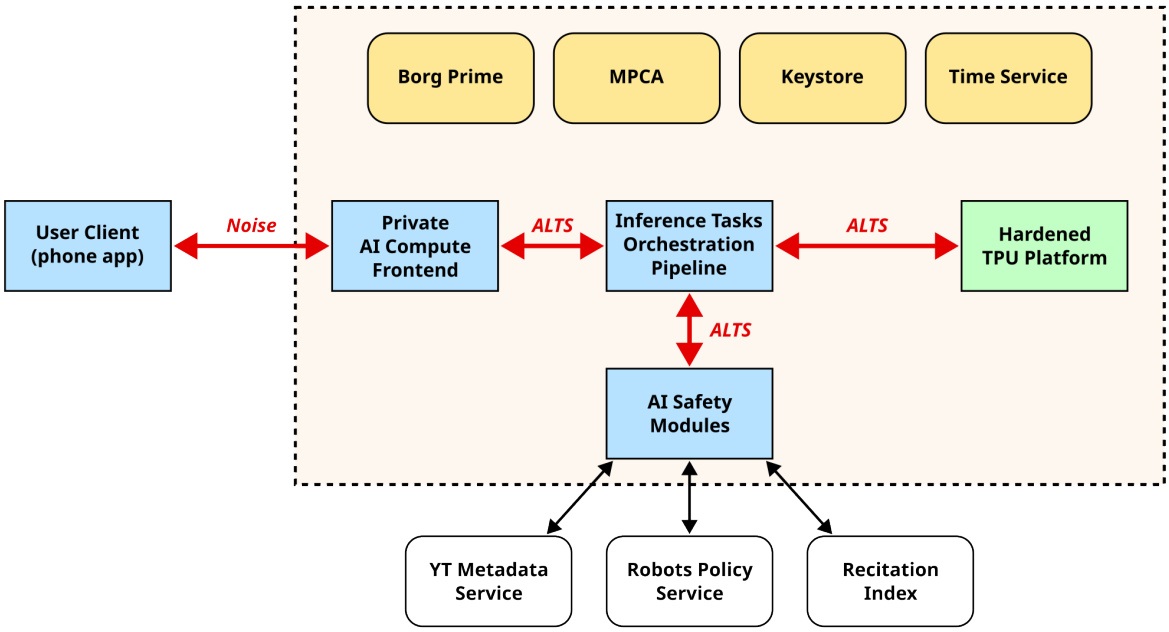

Figure 1: Private AI Compute in the cloud architecture overview (security-related elements)

The functionality of the system is to apply AI-powered inference on data which originates from the user's device; the *privacy goal* is to ensure that this user data is processed exclusively for the AI-powered inference requested by the user, and cannot be extracted from any of the involved systems. The inference is performed on the Hardened TPU Platform, which uses proprietary and dedicated chips especially designed for that task. Each inference process is driven by a request from the client system, which contains some private user data; the request is conveyed to the Hardened TPU Platform, where an appropriate response is generated, and conveyed back to the client device.

System Components

The **user client** is the system which is under exclusive control of the user, and where private data is stored. Typically, the client is a device such as a smartphone, and the user data consists of private text messages, email contents, photos, and other similar data. The client system has legitimate access to that private data.

The user client connects to Google's system through the Oak Session protocol, an open-source derivative of the Noise protocol framework. The Google-side entry point is the **Private AI Compute Frontend** server. This server recognizes the request as being relevant for AI processing, and then forwards it to the **Inference Tasks Orchestration Pipeline** server, which chooses which **Hardened TPU Platform** is going to process it. Once the TPU has finished processing the request, the response goes back to the Inference Tasks Orchestration Pipeline server, which then sends it to the **AI Safety Modules**, another server that applies some post-filtering on the response, in particular verifying that the response is compatible with Google's policies and does not include copyrighted contents.

Each of the Private AI Compute Frontend, Inference Tasks Orchestration Pipeline and AI Safety Modules is in fact a collection of servers that use hardware-backed enclaves within general-purpose CPUs. An enclave is an isolated area (roughly similar to a virtual machine) within a host system that allows performing computations on private data that is kept inaccessible from the host system, even from its hypervisor. The methods through which such an isolation is achieved depend on the hardware vendor (e.g. Intel TDX, AMD SEV-SNP,

Arm TrustZone...) but they are conceptually similar to each other; the CPU hardware itself applies access controls such as cryptographic encryption of data in RAM. Enclave-based security in network computing contexts hinges on the concept of *remote attestation*: a software image executing inside of an enclave can obtain a signed attestation that it runs inside a genuine hardware CPU with a specific software image; the signed attestation is verifiable by third parties. The communications between the Private AI Compute Frontend, Inference Tasks Orchestration Pipeline and AI Safety Modules instances leverage the ALTS protocol, which, from a security point of view, offers characteristics similar to those of SSL/TLS and Noise; each instance creates and maintains a public/private key pair, on which ALTS builds its cryptographic security, and the public part of that key is integrated in the attestation report. In this way, the Private AI Compute Frontend server can, for instance, make sure that when it establishes a connection with Inference Tasks Orchestration Pipeline, the data it sends within the encrypted tunnel can be decrypted only by a genuine hardware enclave running exactly the expected Inference Tasks Orchestration Pipeline software image.

Several additional components support this system; four are most relevant for the security analysis of the Private AI Compute system:

- **Borg Prime** is a central orchestrator for all tasks running within the Google network.
- **MPCA** is a certification authority that is involved in distributing root public keys to systems that are being deployed.
- **Keystore** can store cryptographic keys, and use them on behalf of requesters, subject to strict access control.
- **Time Service** enables synchronization of systems across the network, ensuring that they all work with the same current time and date. This service is similar in concept to the standard Network Time Protocol but with some added cryptographic protection to prevent active attackers from forging fake packets and thus trying to make systems operate on an incorrect time.

These systems are especially involved in how the communication protocols (Oak Session, ALTS) and the remote attestations are articulated together.

Server Authentication and Remote Attestation

When the client device sends its initial request, it must make sure that it has connected to a genuine Private AI Compute Frontend server. The Private AI Compute Frontend server owns a long-lived public/private key pair, which is really stored and used in Keystore. The public half of the key is inherently known to the client software (it is meant to be hardcoded in the client code or at least its configuration). The Oak Session protocol ensures that all data sent by the client will be decryptable only by a system that can use the corresponding private key. Keystore will perform these decryption operations on behalf of the Private AI Compute Frontend server only after making sure that it is indeed talking to a genuine Private AI Compute Frontend server, which naturally involves a remote attestation report verification. The signature on the report is not verified directly by Keystore, though; the Private AI Compute Frontend server dynamically obtains a signed authorization token from Borg Prime; Borg Prime will deliver that token only after validating the remote attestation from the Private AI Compute Frontend server.

The chain of validation is thus the following: the client trusts the public key for sending the user's private data, because it knows that the private counterpart of that key is held by Keystore, that will allow decryption only if requested by a system bearing a proper authorization token; the client furthermore trusts that such a token will be emitted by Borg Prime only after Borg Prime has verified the remote attestation report that demonstrates

that the Private AI Compute Frontend server runs the correct software image inside a genuine hardware enclave.

Further server-to-server communications follow a similar model, though without involving Keystore. Each server running in an enclave generates a local key pair at start-up, and obtains a signed token from Borg Prime over the public half of that key pair; the signed token is an assertion by Borg Prime that it verified (through remote attestation) that the contained public key is indeed under control of a genuine enclave running a specific software image. These local key pairs are used for mutual server authentication in the ALTS protocol.

In effect, Borg Prime is the central gatekeeper. All server components authenticate each other by validating that Borg Prime was convinced of the genuineness of enclaves and correctness of software images; the validation of the Borg tokens is relative to the inherent knowledge of the root public key (installed in each system through the MPCA procedures) and, to some extent, to the correct knowledge of the current date and time (since the Borg tokens are time-limited).

Privacy Model and Limitations

The target model for user data privacy in the Private AI Compute system is to provide users with strong assurance that their data will be processed exclusively for the AI-powered inference they request and cannot be extracted from any of the involved systems. It is desirable that, to the extent that it is feasible, this protection is achieved even if the provider itself *is* the attacker. However, the actual inference work must happen on some physical, hardware system which necessarily sees the data that it is processing; this implies that the designers and builders of that hardware system always retain, whether they want it or not, the power to extract the data by adding some exfiltration channel to their chips. This can theoretically be mitigated, in a mathematical sense, with cryptographic techniques relying on fully homomorphic encryption, but the best known methods to do so are very far, by many orders of magnitude, from working with an efficiency adequate for an AI inference task. A somewhat less perfect method, but still infeasible in practice, would be to split each AI inference task among several independent computing providers, leveraging multi-party computation protocols to make it so that each provider works on blinded data and only a collusion of all providers, or at least a substantial quorum of them, may reveal the real unblinded user data. Fully homomorphic encryption and multi-party computations are an active research area and some progress has been made along the years, making these techniques able to handle some simple tasks (e.g. generating a digital signature), but any modern AI inference is millions to billions of times more computationally intensive. For the time being and the foreseeable future, any practical AI inference must be done over unblinded, plain data, and the vendor of the hardware that makes the processing must be trusted. In the Private AI Compute system, the AI inference hardware is the Hardened TPU Platform, and it is designed and built by Google.

Faced with this inherent limitation, Google's Private AI Compute security model aims for the next best thing, really the best *feasible* model, which is strong protection against malicious insiders: a rogue system administrator should not be able to obtain any private user data, even if abusing hypervisor-level access rights on any number of internal servers. The entire architecture is built with that goal in mind:

- All intermediate servers, that have access to the user data as it flows through the system, run in enclaves with hardware-enforced isolation.
- Internal servers communicate with each other only under the protection of strong cryptographic protocols with mutual authentication bound to remote attestation of their

hardware and software image. The valid software images are further tracked with an internal transparency log, to enhance robustness against malicious software alterations.

- Internal servers take care to never store user data on any physical medium, and to keep user data in RAM only and for the minimum possible amount of time. Crash dumps, if any, are proactively expunged of user data as much as possible, and access to such dumps is covered by additional mitigations such as dual control.
- An additional blinding step is added to mask the source IP address of the client, as an extra line of defence: even if an attacker could compromise one of the internal nodes, they would find it difficult to efficiently target the private data of a specific user.

The architecture is centralized on Borg Prime (and Keystore): user data may be exfiltrated only if at least Borg Prime, Keystore, or the TPU hardware allows it. Each of these components is managed under its own procedures that prevent a single inside attacker from compromising them to that extent. User data may thus be exposed to unexpected processing or outsiders only if Google, as a whole organization, decides to do so (in practice, this could be done by making Borg Prime issue fraudulent authorization tokens to a modified Private AI Compute Frontend server that would also send a copy of the user data to an outside system; in that case, only the IP address blinding will offer any protection against attacks targeted to a single user). As explained above, it is logically unavoidable that Google ultimately has that power; using Borg Prime as a centralized authorization system fits well within that model and has the benefit of leveraging internal security procedures already in place and optimized for Google's internal network.

3 Attestation and Encryption Between Frontend Services and Model Serving Component

NCC Group performed a cryptographic security assessment of the implementation of attestation and encryption between Private AI Compute frontend services and internal servers. NCC Group provided a private report of our findings and observations to Google. This section summarizes that report.

Scope

The following components were in scope:

- Project Oak Session Library implementation: This open-source library provides an implementation (git commit e172cbb6) of Oak end-to-end encrypted attested sessions between communicating parties.
- Implementation of attestation and encryption between frontend services and internal servers: This included the following communications:
  - between the client and the Private AI Compute Frontend server, supported by the Oak Session protocol;
  - between the Private AI Compute Frontend server and internal servers (e.g. Inference Tasks orchestration Pipeline) supported by the ALTS protocol.

Limitations

The review of the implementation of attestation and encryption between frontend services and the internal servers focused on correct use of the security protocols, including Oak Session and ALTS. The ALTS protocol implementation itself was out of scope and therefore was not reviewed during this project. In general, the codebases were complex, written in multiple languages, and challenging to navigate, especially across foreign-function interface boundaries. Nevertheless, the team achieved reasonable coverage of the provided code within the allotted time and reported several security findings.

Key Findings

The assessment uncovered a number of issues. The most notable findings were:

- Finding "Unbounded Session Message Queues", where an attacker could exhaust the resources of a client/server application that uses the `oak_session` library by sending, or coercing the sending of, a large number of messages.
- Finding "No Limit on Attestation Data Request and Response", again where one may exhaust the memory of an application using the `oak_session` library by sending spurious attestation data before the Noise handshake occurs.
- Finding "Lack of Full Protocol Transcript", which may result in various protocol attacks.

NCC Group reported all findings to Google (see Table of Findings). NCC Group understands that Google has mitigations under development for the first two findings and is actively working on designing a fix for the third finding. NCC Group also captured a number of notes and observations in an “Engagement Notes” section of the private report, which is repeated later in this section.

Strategic Recommendations

Strengthening the Oak Session protocol and its implementation: The Oak Session protocol would benefit from recording and verifying a complete transcript of its execution, further reducing its attack surface. Its implementation should communicate expectations more explicitly, both in the API documentation and in the code itself, to aid maintenance and

security analysis. Some functions seem to assume that inputs are always valid at a given stage of execution, an assumption that may not hold in an adversarial environment; moreover, several of these functions do not fail safely by default.

Engagement Notes

This informational section highlights a number of observations that the NCC Group team gathered during the engagement and that do not warrant security-related findings on their own.

Oak Session Library

- The library is mostly responsible for managing the secure sequencing and interactions of user configured handlers, and providers with session data. A failure in any of these components would likely cause a security compromise.
- Function `verify_session_binding()` in file *session.rs* verifies each binding provided by a peer. If there is no attestation result with status `VerifierResult::Success`, then the function reports a success. The lack of `VerifierResult::Success` attestation result should not happen in normal circumstances. However, any issues in the implementation elsewhere may assist in triggering and exploiting such conditions. Therefore the function is a bit fragile. Consider returning a failure if no bindings were verified.

```
fn verify_session_binding(
    binding_verifier_providers: &BTreeMap<String, Arc<dyn SessionBindingVerifierProvider>>,
    attestation_results: &BTreeMap<String, VerifierResult>,
    bindings: &BTreeMap<String, SessionBinding>,
    handshake_hash: &[u8],
) -> Result<(), Error> {
    for (verifier_id, result) in attestation_results {
        if let VerifierResult::Success { result: r, .. } = result {
            let binding_verifier = binding_verifier_providers
                .get(verifier_id)
                .ok_or(anyhow!(
                "no session binding verifier provider supplied for the verifier ID
                ↳ {verifier_id}"
            ))?
                .create_session_binding_verifier(r)?;
            binding_verifier.verify_binding(
                handshake_hash,
                bindings
                    .get(verifier_id)
                    .ok_or(anyhow!(
                        "handshake message doesn't have a binding for ID {verifier_id}"
                    ))?
                    .binding
                    .as_slice(),
            )?;
        }
    }
    Ok(())
}
```

- The `oak_session` library does not implement the Noise KN pattern. Both the Oak Session client and server invoke `core::unimplemented!()` if the session is configured for the Noise KN pattern, which would crash the application at configuration time for the former, and when receiving a handshake response for the latter. This should not be an issue in most deployment scenarios, except if the application using the `oak_session` library allows to

dynamically configure the `oak_session` Noise pattern via an exposed control plane, and if an attacker can influence this configuration. The implementation team should consider returning an error to the application, instead of forcing a crash. The issue is illustrated in the code extract below for the server:

```
impl ProtocolEngine<HandshakeRequest, HandshakeResponse> for ServerHandshakeHandler {
    /// Gets the outgoing `HandshakeResponse` to be sent to the client.
    ///
    /// This message is generated after processing the client's initial
    /// `HandshakeRequest`. It will contain the server's contribution to the
 // SNIP
    fn put_incoming_message(
        &mut self,
        incoming_message: HandshakeRequest,
    ) -> anyhow::Result<Option<()>> {
        if self.handshake_result.is_some() {
            // The handshake result is ready - no other messages expected.
            return Ok(None);
        }
        if let Some(noise_response) = self.noise_response.take() {
            self.handshake_result = Some(HandshakeResult {
                crypter: noise_response.crypter,
                handshake_hash: noise_response.handshake_hash.to_vec(),
                session_bindings: incoming_message.attestation_bindings,
            });
        } else {
            let noise_response = match incoming_message.r#handshake_type {
                Some(handshake_request::HandshakeType::NoiseHandshakeMessage(noise_message)) =>
                ↳ {
                    match self.handshake_type {
                        HandshakeType::NoiseKN => core::unimplemented!(),
                        HandshakeType::NoiseKK => respond_kk(
```

- Attestation messages are unencrypted and may include sensitive information, depending on the configured attestation provider. Consider documenting this risk for users of the library.
- The library operates multiple state machines: one for `SessionRequest` / `SessionResponse` messages for both client and server, and one for attestation request/response, handshake request/response, and the open phase messages. NCC Group attempted to identify ways to cause these state machines to fail in an unsafe manner but did not identify any, assuming the user configured handlers and providers are secure.

Private AI Compute Frontend Client and Server

- Forward secrecy: The client and server engage in a Noise NN handshake, thus the data payload has forward secrecy for the sender, since encryption involves an ephemeral-ephemeral DH (“ee”).

Ciphertext Size, and Other Side-Channels

Inference plaintext data is encrypted when the ALTS and Oak Session protocols are used, preventing observers from decrypting it without the necessary keys. However, since no padding is applied to the plaintext, an observer can still infer the length of the plaintext, both for the request and the response, from the length of the ciphertext. This side-channel has low bandwidth and is, in general, not exploitable by itself, though knowing the message length may help observers mount further attacks, such as guessing the correct plaintext under certain conditions, depending on the application requesting the inference. Additional

side channels include the timing of inferences (e.g., when an email or call arrives and a communication summary is generated). Applications handling especially sensitive inference requests should consider countermeasures such as padding plaintext before encryption to reduce these risks.

4 Private AI Compute Frontend Server, IP-blinding Relay, T-Log, Crash Dump, and Outbound RPC Enforcement

NCC Group performed a security assessment of the Private AI Compute Frontend server, IP-blinding relay, T-Log binary matching/verification, crash dump export and admin access, and the Outbound RPC Enforcement configuration. The consultants provided a private report of our findings and observations to Google. This section summarizes that report.

Scope

The following components were in scope:

- Private AI Compute Frontend server
- IP-blinding relay
- T-Log binary matching/verification
- Crash dump export and admin access
- Outbound RPC Enforcement configuration

Limitations

NCC Group completed all in-scope components with full coverage.

Key Findings

Timing-Based Side Channels that may be able to unmask target Private AI Compute users when particular conditions are met. The exploitability rating was marked as 'Low'.

Denial of Service for legitimate users as a result of malicious applications exhausting victims' certificate quota. Note that this finding may be addressed by compensating controls and is being reviewed by Google internally. This is not a privacy concern.

NCC Group separately reported all findings to Google (see Table of Findings). NCC Group also captured a number of notes and observations in other sections of the private report, which are repeated later in this section. Google filed bugs for some of these observations and reported working on fixing them and/or having fixed them to NCC Group.

Strategic Recommendations

- Implement additional anonymization features as discussed in the findings in order to better preserve user privacy and mitigate behavioral-timing correlations.
- Ensure the system undergoes dynamic testing to validate that design controls are met with sufficient technical control equivalents.
- Ensure the logging rulesets have been sufficiently evaluated by third-party auditors to ensure sensitive data never enters application logs.

Private AI Compute Frontend Server

The Private AI Compute Frontend server is a public-facing service that interfaces client devices with the internal Inference Tasks Orchestration Pipeline server, after validating attestations of both clients and servers. Consultants reviewed the full codebase of the Private AI Compute Frontend server, as well as parts of relevant dependencies such as the Oak Session implementation (cryptographically reviewed previously).

General observations of the code show good security practices, employing generally safe methods for handling untrusted input. User inference requests are parsed by generated protocol buffers code, and no security-relevant processing is performed on them. These

requests are transformed into the appropriate format to pass on to the Inference Tasks Orchestration Pipeline server for further processing.

NCC Group did not find issues with the Oak Session handling and Noise protocol code. Binding key with initial setup and periodic refresh appears well-designed. The implementation of parts of the binding key manager in Rust is a welcome addition, and consultants did not find issues with the C++ interface implementation to the Rust routines.

One of the important functions in the Private AI Compute Frontend server is the attestation verification. The logic appears correct, with device state, app integrity, and other aspects checked. However, NCC Group reported certificate chain validation weaknesses in Phase 2 Stage 1 of the project in finding "Fragile and Possibly Insecure Verification of Android Client Attested Keys". Errors are handled in all cases they should be, and communicated properly for logging.

Overall, the Private AI Compute Frontend is a concisely-defined service that does not attempt to provide more functionality than is required at this layer of the inference system. This is evident from the quality implementation of the small number of functions it provides. While it is always possible in a complex system like Private AI Compute that issues may arise from the overall architecture, NCC Group did not find any issues in the Private AI Compute Frontend server outside of the aforementioned certificate chain validation finding.

Admin Access ACLs

NCC Group evaluated the “Set of ACLed Administration commands” Access Control Lists (ACLs) used to guard the Private AI Compute system from unauthorized access to confidential data or manipulation of configuration. The “Set of ACLed Administration commands” is used to manage access to files, services, commands, and other sensitive assets in Google’s infrastructure. Google provided consultants with two sets of ACLs to review:

- An initial deployment set of controls applied while the Private AI Compute service loads, which can be used for server tasks like repairs. This deployment implements narrower restrictions to target entities, enabling more sweeping permissions. However, according to Google documentation, this state requires a fresh installation and cannot be used to access data while the Private AI Compute service is loaded.
- A stricter set of controls used while Private AI Compute is running, which overrides other rules deployed to the server.

NCC Group verified that the “Set of ACLed Administration commands” restrictions appear appropriate and do not excessively enable access to Private AI Compute services or files. Additionally, according to Google, the service performs workload validation at boot time to ensure that critical files are not tampered with, preventing the lower-trust mode from interfering with or manipulating the higher-trust mode. Note that this review did not include analysis of the implementation of these claims.

Limitations

NCC Group possesses limited in-depth and domain-specific knowledge of Google’s internal service configurations, security processes, and the validity of compensating controls. Consequently, although NCC Group could review the general reasonableness of the provided ACLs, validating their sufficiency, completeness, and implementation would require a significant amount of additional effort. NCC Group relied on Google for clarification when the deployment or rule application process appeared unclear. There were no indications of underlying weaknesses.

Crash Dumps

Google's Private AI Compute service processes significant quantities of sensitive information. Consequently, normal debugging procedures such as core dump analysis on application crash may inadvertently reveal information that violates Google's privacy claims to customers. In order to mitigate this possibility, Google has disabled the core dump functionality from production Private AI Compute binaries. In order to validate this behavior, NCC Group reviewed the configuration files associated with key Private AI Compute binaries:

- Private AI Compute Frontend
- Inference Tasks Orchestration Pipeline
- Model Serving component

Because NCC Group lacked sufficient internal permissions to access this data directly, Google provided a screenshare of the organization's internal deployment management systems and demonstrated that no core dump was generated when a crash was induced in any of the three running binaries. However, the crash investigation tool provided access to the standard error of the application, which would expose any sensitive data that enters the relevant output stream. NCC Group advised Google to validate and demonstrate that standard error will never log data covered by Private AI Compute privacy guarantees.

After NCC Group issued the report, Google explained that they believe that their implementation does have sufficient mitigations for this risk (i.e. Google believes the recommendation made in the report is what is implemented). Google stated the following:

> The log redaction step is implemented via tagging all processing that happens in contexts where user data is handled with an attribute that restricts debug logging. This context is preserved across RPCs and all layers of software within Google systems. Systems within the TCB are then configured to redact logging messages when they detect this attribute. Redaction means that any non-literal string is removed from the output.
>
> While it would be possible to escape this log redaction, this would either have to be via:
>
> - Deliberately circumventing the redaction (detectable via code review)
> - Changing the system configuration to disable redaction (detectable in code review and auditable via our release control policies)
> - Bugs in redaction (a possibility we cannot discount, but we have automated tests in place to catch)

Although the Model Serving component binary configuration disabled core dump functionality, two settings enabled a remote core dump configuration option. According to Google, this feature enabled metadata to be collected without exposing memory contents to Google's team. NCC Group also observed that the crash status page provided developers access to error log output, which does not appear to provide access to sensitive data (potentially due to redaction) but has been recorded in detail in finding "Crash Dump Debugging Exposes Standard Output".

Limitations

Google's documentation claims that only the three in-scope binaries process raw user data. Although NCC Group validated Google's core dump policies for the provided binaries, NCC

Group could not feasibly validate that *no* user data enters *any* other binary leveraged by the application. NCC Group also had to rely on the trustworthiness of Google's internal toolsets to validate crash dump claims and the trustworthiness of Google's claims that the binary standard error output will never contain private data.

IP-blinding Relay

Using documentation and walkthroughs provided by Google, NCC Group evaluated the key security and privacy claims of the IP Blinding component of the Private AI Compute system.

Google's IP Blinding functionality leverages a multi-stage process to obscure the identity of users submitting queries to inference servers. This process aims to accomplish two goals:

1. Ensure that, even if the inference server were compromised, queries cannot be traced back to the originating users, save for the content of the queries themselves.
2. Ensure that, despite users remaining unidentifiable, the system can sufficiently rate limit queries and prevent abuse.

The entire inference system also aims to maximize the falsifiability of Google's claims. In other words, any deviation from the privacy, security, availability, and other standards that Google attests to should be easily detectable by third parties. This attribute ensures that, if at some point the underlying premises that ground Google's claims were to be violated, these violations could not be exploited in secret.

Inference Pipeline

Clients contain a certificate chain burned into a trusted computation zone by the device manufacturer. Google signs downstream certificates with their root certificate and can revoke previously issued certificates in the event of data leak or compromise. These certificate chains are paramount to device-level attestation.

Initial Authentication

The client establishes a trust relationship with the issuance server by submitting an authentication request. The server signs a timestamped nonce with Google's private key, which is verified by the client. After receiving the nonce, the client signs a fresh certificate chain with its built-in certificate that also includes the nonce from the attestation server.

Blinded Token Exchange

Clients also generate a token, which is blinded according to the Blind RSA scheme. The token and generated certificate chain are submitted to the previously contacted issuance server, which verifies that the chain is valid and that the client has not submitted exceedingly many signing requests. If both checks pass, the server signs the blinded token, and returns it to the client.

The client unblinds the token to acquire a signed token that it can later present to the proxy server for validation. This blinding-unblinding process prevents correlation between the original signing request submitted by the client and later signature presentations to the inference server.

Proxy Connection

In order to remove all Internet Protocol (IP)-related information from incoming requests, the Private AI Compute system proxies requests from client devices through third party servers (hosted by Cloudflare and Fastly), which establish a direct connection to the inference server using an API token for authentication. In order to authenticate to the proxy, the client submits the now-unblinded, signed token and proxy connection key that it received from the issuance server, which are verified by the proxy. Once the initial handshake is validated, the proxy passes client traffic to the inference server, establishing a MASQUE tunnel. The client

repeats the blinded token exchange process with a Noise session nonce returned by the inference server.

Security Analysis

Privacy

Malicious Insiders

The identity blinding features offer robust defense against direct user unmasking, even given a compromised or hostile inference server. Threat actors situated across multiple access levels are prevented from attributing a particular inference request to a particular user. For example:

- Threat actors situated at the third party proxy are incapable of reading the contents of user requests due to Google's use of the *Noise* encryption protocol. Although these entities can correlate encrypted packets to their source, they cannot impersonate Google to decrypt the contents of user inference requests.
- Threat actors situated at the issuance server do not receive the contents of the user's inference request, and instead see the user's attestation certificate chain and the user's blinded token.
- Threat actors collaborating across issuance servers and the inference server cannot directly correlate a particular inference request to the initial handshake used to establish a session due to token blinding.

Worth noting, the contents of requests themselves may contain sensitive and identifiable information, which is necessarily processed by the inference server during its computational tasks. However, that attribute is unavoidable due to the nature of inference services and will not be mitigatable in the foreseeable future.

Session Ephemerality

IP Blinding supports short-lived sessions and requires renegotiation with the attestation server whenever the user submits a new prompt. User sessions do not persist in a fashion that would enable Google to draw correlation between their prompts. This ephemerality of sessions augments the anonymization of requests: not only are individual requests unattributable to a specific user, but it also prevents detecting whether any two requests originate from the same user. Additionally, the IP Blinding feature is designed such that even if relevant metadata were to be stored, user privacy would not be impacted.

Attacker Collaboration

The notion of "threat actor" used above covers not only a malicious insider, but also multiple insiders collaborating together, or even the entirety of Google, under the extreme assumption of Google being organizationally intent on unmasking users. IP blinding through an external third party offers its protection to users even in that case. Similarly, users remain safe in the hypothetical situation of the network third party (Cloudflare or Fastly) being entirely compromised. However, this scheme would not prevent *collaboration* between Google and Cloudflare or Fastly to unmask arbitrary user requests and correlate the contents of an inference query to the user who produced the query. This attack vector is documented in finding "User Exposure Via Collaboration Between Google And Proxy", but encounters the following limitations:

- Proxy providers unmasking users would violate their contract with Google.
- Compromising the necessary infrastructure of both entities would likely prove prohibitively difficult for most threat actors.

- Such activity would result in significant reputational harm for both organizations, if discovered.
- Accessing the contents of specific requests would prove to be technically challenging, given Google's technical controls.

The most applicable attack scenario would likely comprise a required collaboration between Google and Cloudflare or Fastly through legal means such as court order. However, Google's plans to verifiably attest to the binaries in execution on the inference server will likely render even this attack scenario detectable.

Timing Attacks

During NCC Group's interview with the engineering team, Google acknowledged that timing-based side-channel attacks are inherent to this form of solution, given worst-case restraints. For example, if a user initiates an attestation handshake before sending an inference request and is the only user on the platform during the time period, Google could derive the originating identity of the inference request. However, the system as implemented resists this form of attack through its multi-user nature adding sufficient noise to disrupt user-query correlations. The system's four-step handshake to establish a connection poses risky implications for user privacy due to the upper and lower bounds for valid candidate user requests established by each step of the process. This risk has been discussed in finding "Timing-Based Side-Channel Attacks Can Unmask Users", and NCC Group understands that Google has a mitigation under development.

Other forms of side-channel correlation may also apply. For example, consider a user who regularly interacts with large query sizes, such as through document uploads. Google could, over time, build a correlation between the user's initial attestation handshake and the resultant large query on the inference server by examining the metadata of requests within a short window after the handshake completes. Similarly, users who always submit two requests at a time may exhibit identifiable behavior by comparing timestamps across issuance and inference servers. Although this process could not be used to target arbitrary users, it may be applicable to unmask users who predictably submit atypical requests.

Similarly, a user who submits a large query followed by another query in succession may be identifiable in the event that the timestamp between issuance server requests relatively matches the inference processing span of a query submitted within that window.

Each of these side-channel attacks serve to construct weak correlations between users and their resultant queries and present technically challenging problems to exploit. However, timing-based attacks remain the most prominent risk to the privacy of users in the Private AI Compute system, due to their low likelihood of detection, and conceptual applicability by lone insiders. The high-volume of usage and required network privilege/position significantly increase the difficulty of exploitation.

Denial of Service

The issuance server's ability to validate clients relies on the attestation certificate chain, which is signed by device manufacturers and cannot be changed by end users. Attackers who compromise a manufacturer's signing certificate can spawn arbitrary certificates and use Google's services without limits. However, this attack is unlikely due to the requisite effort and relatively mild profit. Instead, exposed certificates used in other, unrelated attack vectors may induce Google to add those certificates to its revocation list and reject valid user devices, denying access to the inference system. This attack vector has been explored in finding "Manufacturer Certificates May Abuse Resources Or Trigger Denial-Of-Service For Users".

Malicious applications may be able to induce the on-device certificate to generate multitudinous certificate chains, rapidly increasing the apparent number of requests the system has made to Google's inference server. Attackers who exploit this behavior may be able to flag users as potential service abusers and deny access to the Private AI Compute platform. This attack vector is being investigated by Google's internal team and is outlined in finding "Malicious Applications May Exhaust Certificate Quota".

Service Regulation Failure

Google's ability to limit the service depends on the legitimacy of on-device attestation, which includes the number of chains it has generated in its metadata. Because these keys exist within user possession, determined attackers may be able to extract their values from the device using device vulnerabilities, advanced equipment, or some other unknown mechanism. The difficulty of extracting these keys would likely prove to exceed their profitability, but could enable users to submit arbitrarily many requests to the Private AI Compute platform. This attack vector has been explored in finding "Device Attestation Relies On At-Risk Data".

T-Log Binary Matching/Verification

Google Transparency Log (T-Log) is a tamper-resistant ledger of metadata generated within the Private AI Compute project's supply chain. NCC Group reviewed this system and the documented methodology that is used to ensure tamper evidence and publication to the T-Log. NCC Group's Cryptography Services team also reviewed some aspects of its cryptography design and implementation. The review was based on the following provided documents:

- Tamper Evidence
- Tracing back endorsements to source code

This was not an exhaustive cryptographic review of the T-Log system, which is complex. The team attempted to identify potential cryptographic issues that might prevent the system from meeting its security assurance goals within the time allotted for this effort. Further cryptographic work on T-Log would permit deeper analysis and could be included in future phases. This section highlights several observations NCC Group gathered during the engagement.

Keysets May Be Ambiguously Encoded

A keyset hash is a resource that maps to a list of endorsement keys. Document "Tracing back endorsements to source code" provides the following example recipe to compute the keyset hash:

```
# Example recipe to extract Model Serving Component's keyset hash

CONFIG_ID=78653
KEYNAME=tr_pi_model_server_tpu_verifying_key
KEYSTORE_ADDRESS="blade:keystore-signing-prod-fastconfig"

echo -n "$KEYSTORE_ADDRESS $CONFIG_ID $KEYNAME" | sha256sum
```

Specifically, the keystore address, key name, and configuration ID values are joined using a single space, before the result is hashed using SHA-256. There is no apparent formal specification of these field values, e.g., what is permissible or not. If whitespaces are

allowed, multiple input values may result in the same keyset hash. For instance, the following values:

```
CONFIG_ID='alice bob'
KEYNAME='charlie'
KEYSTORE_ADDRESS='david'
```

would resolve to the same keyset hash as these values:

```
CONFIG_ID='alice'
KEYNAME='bob charlie'
KEYSTORE_ADDRESS='david'
```

The impact appears to be minimal with NCC Group's current knowledge of the system. It would be judicious to specify the allowed domains for these variables, and provide tools to validate and compute the keyset hash non-ambiguously to reduce its attack surface. Sourcing of these hashes should be clarified; there is an open issue linked from the document:

> These hashes should be shared somewhere (`b/425262111`)

Component Implementation Weaknesses, `oak_endorsement_verification_cli`

The "Tracing back endorsements to source code" document explains how to list all endorsements signed with any key version in a keyset in step 2. This step relies on Project Oak's `oak_endorsement_verification_cli` component. This is how one would list all endorsements for keyset SHA-256 value "6117df9277c7f05dd6453d92344f6621df86df0a526bab10848f3e3c83b7cffc":

```
bazel run oak_endorsement_verification_cli:endorscope -- list --fbucket=14633 --ibucket=29702 -
↳ -endorser-keyset-hash="sha2-256:6117df9277c7f05dd6453d92344f6621df86df0a526bab10848f3e3c83b7c
↳ ffc
```

The component does not perform validation of its input and will echo back any provided input as-is. An attacker can leverage this to output contents of their choice (be it to the terminal, log file, Unix pipe, etc.) and therefore mislead users, or other processes, about the outcome of the endorsement verification process. *This is contingent on how the `oak_endorsement_verification_cli` component sources its input as part of the auditor workflow*. While NCC Group understands that this has subsequently been fixed, the following rough proof-of-concept command screenshot would visibly echo in a terminal that a fake MPID is valid:

```
bazel run oak_endorsement_verification_cli:endorscope -- list --fbucket=14633 --ibucket=29702\
--endorser-keyset-hash="sha2-256:6117df9277c7f05dd6453d92344f6621df86df0a526bab10848f3e3c83b7cffc

    # SNIP a large amount of newline characters

  ✅ sha2-256:cad324cb42620d19d2262d24e2886520a39b76bc5c44e539d32b0a4ce3944ba6
       Subject:     sha2-256:d2445490e5dc005afb999e5ab0b4cd7ff006fb5dd67b5274ab2d1a48e0392b90
       Not before:  2025-06-09 6:10:27.0 +00:00:00
       Not after:   2026-06-09 6:10:27.0 +00:00:00
       🍀 MPM version ID: 1-not_a_valid_mpm_id
       ❌ https://github.com/project-oak/oak/blob/main/docs/tr/claim/92939.md (Open Source)
       ❌ https://github.com/project-oak/oak/blob/main/docs/tr/claim/68317.md (Runnable Binary)
       ❌ https://github.com/project-oak/oak/blob/main/docs/tr/claim/52637.md (Published Binary)
       🟦 https://github.com/project-oak/oak/blob/main/docs/tr/claim/31543.md (Distributed as MPM)

"
```

Note that `fbucket`, `ibucket`, `endorser-keyset-hash` are used to construct URLs to fetch resources, and maybe used to perform path traversals, presumably with no significant adverse impacts.

Furthermore, on Linux at least, the return value of the command would show no issue with the malicious input, making it less suitable for automation:

```
$ echo $?
0
```

The Rekor client tool mentioned in the "Tracing back endorsements to source code" document shows an improved approach:

```
~/go/bin/rekor-cli search --sha="b25983a1b6ba042384f4189a40
aaaaaa"
```

It validates the input, and outputs it back encoded if it is invalid:

```
Error: invalid argument "b25983a1b6ba042384f4189a40\naaaaaa" for "--sha" flag: error parsing
↳ sha flag: invalid SHA512 value
Usage:
  rekor-cli search [flags]

Flags:
      --artifact fileOrURL     path or URL to artifact file
      --email email            email associated with the public key's subject
  -h, --help                   help for search
      --operator operator      operator to use for the search. supported values are 'and' and
      ↳ 'or'
      --pki-format pkiFormat   format of the signature and/or public key (default pgp)
      --public-key fileOrURL   path or URL to public key file
      --sha sha                the SHA512, SHA256 or SHA1 sum of the artifact

Global Flags:
      --config string      config file (default is $HOME/.rekor.yaml)
      --format format      Command output format (default default)
      --rekor_server url   Server address:port (default https://rekor.sigstore.dev)
      --retry uint         Number of times to retry HTTP requests (default 3)
      --store_tree_state   whether to store tree state in between invocations for additional
      ↳ verification (default true)
      --timeout format     HTTP timeout (default 30s)

invalid argument "b25983a1b6ba042384f4189a40\naaaaaa" for "--sha" flag: error parsing sha
↳ flag: invalid SHA512 value
```

It returns an error in case of issue with the input:

```
$ echo $?
1
```

NCC Group recommends amending component `oak_endorsement_verification_cli` such that:

- it validates all input,
- it does not echo back invalid input, or output encodes input such that it cannot confuse users,
- and it returns an error code in case of issue with the input.

After NCC Group issued the report, Google stated that they implemented these recommendations. NCC Group did not review how contents fetched by

`oak_endorsement_verification_cli` using HTTP are parsed and validated. This could be an area of interest for a future project.

Endorsement Keys Lack of Validation

The "Tracing back endorsements to source code" document does not explain how to validate the endorsement keys' values used to sign artifacts. In the absence of such information, the documented process would "pass" for arbitrary (potentially malicious and/or compromised) endorsement keys. This process should be specified.

Endorsement Verification

- Endorsement statement validation: the `oak_endorsement_verification_cli` component's `list` operation outputs a list of endorsements for a given keyset, as detailed in the "Tracing back endorsements to source code" document. It checks the time range validity of a given endorsement statement. The demonstration data provided in this document does not appear to contain required endorsement statement claims. In the absence of required claims, the system does not check whether the endorsement statement contains the necessary claims. Presumably, the auditors must manually review that all the required claims are present by inspecting the output of the `list` operation. NCC Group recommends clarification here, and for the next item.
- Rekor log inclusion: the demonstration data provided in the "Tracing back endorsements to source code" document does not appear to enforce verification (`verifying_key_reference_value::Type::Skip`) of appropriate inclusion of logs in Rekor, therefore inclusion is not verified. Again, it is assumed that auditors must manually review inclusion in Rekor, as documented in section "Static check: All Rekor log entries are represented in GCS" of the aforementioned document.

Artifact Signatures

Signatures use NIST's P-256 curve, and the SHA-256 algorithm. NCC Group validated the correct usage of cryptographic APIs to verify signatures of artifacts in the `oak_endorsement_verification_cli` component. However, as noted above, an auditor cannot verify that the signatures were produced with "trusted" private keys without additional information.

Methodology for External Auditors

NCC Group reviewed the methodology used by Google themselves to verify binaries provenance. Google indicated that this methodology was still in development and evolving.

Auditors must be currently granted internal access to Google systems in order to perform an audit. In effect, only Google, and external auditors granted access by Google, can verify whether components of its Private AI Compute system were compromised or not.

The process for an auditor who is not familiar with the internal tools and techniques is potentially cumbersome and may involve some trial-and-error to get working. NCC Group recommends that Google provides a wrapper tool to automate selecting a package to verify, providing a link to source code and binaries, getting the keyset, listing endorsements, checking signatures and getting metadata via Stubby. This could improve repeatability. Having periodic automated spot-checks would also help demonstrate that the running binaries are regularly verified as being endorsed by the Private AI Compute team.

It was not immediately clear to an auditor if some binaries rely on configuration packages that may contain security-relevant variables but are not themselves tracked by the T-Logs. These should also be included to provide full coverage.

Outbound RPC Enforcement

NCC Group reviewed the access control lists (ACLs) on outbound RPCs as documented in the following spreadsheet:

- Private AI Compute OutboundACLs Audit List

The Private AI Compute team pulled the ACLs from production jobs as of 2025-06-20 and the number of service dependencies have been reduced to what is considered necessary to run jobs. NCC Group reviewed over 370 service dependencies spread across the Private AI Compute Frontend, the Inference Tasks Orchestration Pipeline, and the Model Serving component. The review looked at whether the service was likely to be privacy impacting, based on information gleaned from the linked proto definitions within the service source code repositories.

Due to the number of services a light-touch review was done with limited background context, designed to identify any services that may be privacy impacting. The results were added to the spreadsheet using the pre-defined columns for “Privacy Issue?” and “Notes”.

This approach was designed to reduce the accidental exposure of privacy relevant data via RPCs to unintended services. As noted in “NCC Code audit RFP” the Private AI Compute team acknowledges that the ACLs add some additional defence-in-depth but are not designed to outright prevent malicious binaries from exfiltrating data as they can simply ignore the ACLs.

It was confirmed through review of the manifest for the Private AI Compute Frontend server that SLSA Outbound RPC Enforcement checks appeared to be enabled and enforced.

Overall, there were few obvious privacy-impacting services that were identified. Some possible (and likely required) ACLs were marked as “Lean Yes” as certainty would require additional context. These should be reviewed to understand if they are indeed relevant, and needed. They appeared to be privacy related, but not necessarily a privacy issue.

The review was challenged by the following caveats:

- Some of the links to the proto files returned zero results.
- Sometimes there were multiple results and the most likely candidate was selected.
- It is possible some proto files returned in the links were not relevant to the service.

As such, NCC Group recommends Google review the process for assessing the ACLs to be able to provide an external auditor with accurate information on what each service does and the data it handles. It should be kept in mind external auditors may not be familiar with Google services, code locations and nomenclature.

5 Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors.

Title	Status	ID	Risk
Unbounded Session Message Queues	Reported	73W	Medium
No Limit on Attestation Data Request and Response	Reported	K6T	Medium
Timing-Based Side-Channel Attacks Can Unmask Users	Reported	D9U	Medium
Incorrect Function API Documentation and Possible Unexpected Behaviors	Reported	92F	Low
Lack of Full Protocol Transcript	Reported	DR3	Low
Fragile and Possibly Insecure Verification of Android Client Attested Keys	Reported	UHL	Low
Malicious Applications May Exhaust Certificate Quota	Reported	LTM	Low
User Exposure Via Collaboration Between Google And Proxy	Reported	QPV	Info
Manufacturer Certificates May Abuse Resources Or Trigger Denial-Of-Service For Users	Reported	HQC	Info
Device Attestation Relies On At-Risk Data	Reported	6UU	Info
Crash Dump Debugging Exposes Standard Output	Reported	RGL	Info

6 Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Rating	Description
Critical	Implies an immediate, easily accessible threat of total compromise.
High	Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.
Medium	A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.
Low	Implies a relatively minor threat to the application.
Informational	No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

Rating	Description
High	Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
Medium	Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
Low	Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

Rating	Description
High	Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

Rating	Description
Medium	Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.
Low	Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Category Name	Description
Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.