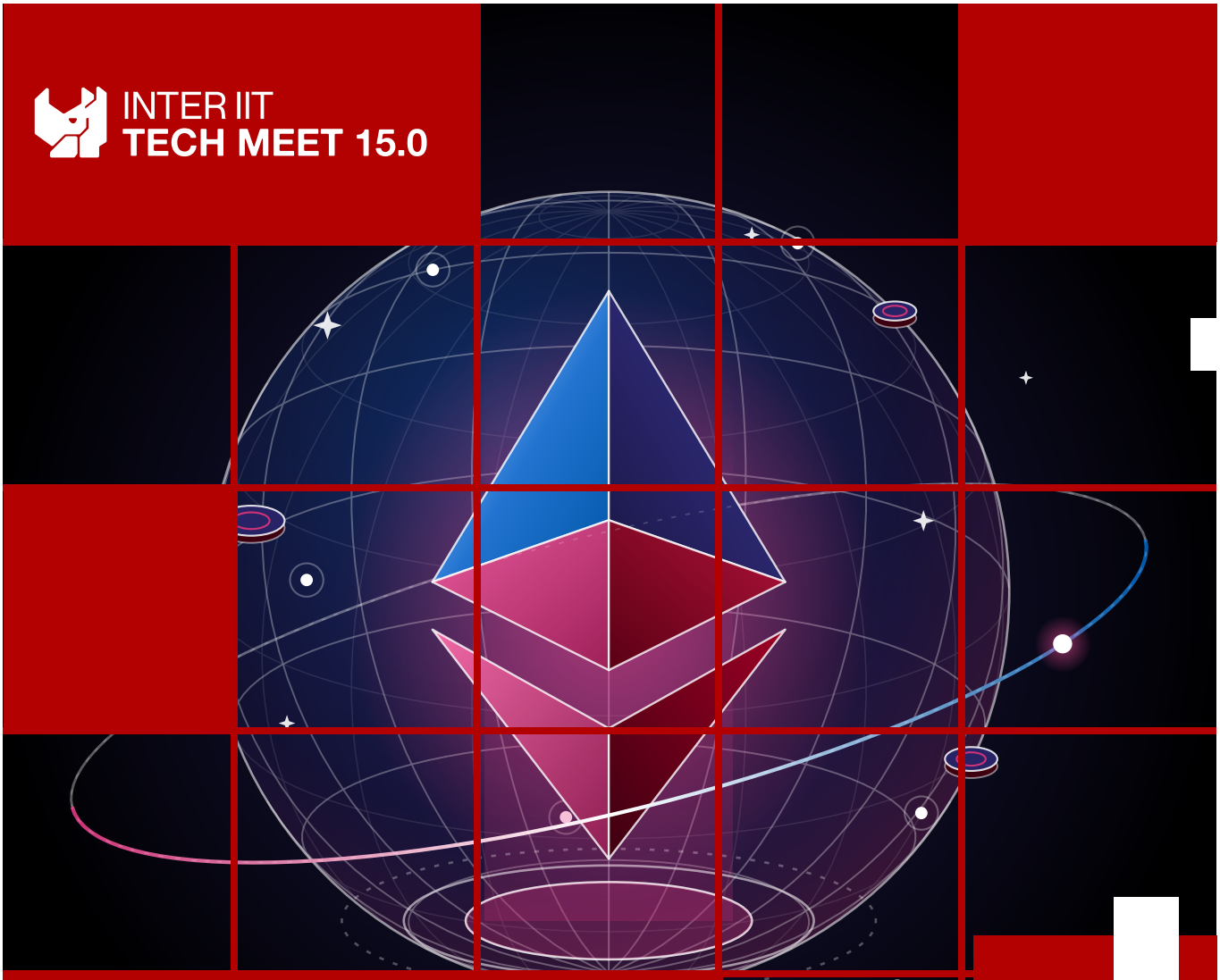
INTER IIT
TECH MEET 15.0



MID PREP

ethereum foundation
Academic Secretariat

Access Layer for AI and Ethereum Wallets

Introduction

An AI assistant could make using an Ethereum wallet as simple as explaining what you want, checking the details, and approving the action. Along the way, you might share who you're paying or why, and you should be able to do that without worrying that those details will be unnecessarily exposed to external services or end up in a public transaction.
We want teams to build the access layer between an existing open-source AI model and Ethereum: a system that provides a wallet interface, keeps private information safe, and carries out only what the user has agreed to, while giving users a system they can run themselves and an Ethereum wallet they remain in control of.
The focus is on building around existing models: connecting natural-language interpretation to wallet functionality, enforcing permissions, protecting information, and safely handling model outputs.

About us

The Ethereum Foundation (EF) is a global non-profit organization dedicated to supporting the Ethereum network and its broader ecosystem of decentralized technologies. Acting as a neutral facilitator of open-source innovation, the EF funds critical protocol development, fundamental research, and community growth to ensure the long-term sustainability of the Web3 space. A key specialized division within the organization is the Ethereum Foundation Academic Secretariat, which actively bridges the gap between decentralized technologies and higher education. By fostering collaborations with university faculties and student developers, the Secretariat empowers the academic community to advance foundational blockchain research and build a critical talent pipeline for the next generation of decentralized infrastructure.

Problem Statement Description

Build a system that acts as the access layer between an existing open-source AI model and Ethereum, providing an Ethereum wallet interface through natural-language requests.

The model can interpret requests and propose actions. Your access layer should manage the interaction with the wallet, validate proposed actions, enforce the user's permissions, and ensure that signing and execution remain under the user's control. Teams may integrate existing wallet software rather than build a new wallet.

Choose a small set of useful tasks and demonstrate five things:

- **It keeps users in control:** Users retain control over their Ethereum wallet and assets, and can run, inspect, and continue using the core system without relying on a required provider or centralised intermediary.
- **It understands the request:** It uses the existing model to work out what the user wants and asks for clarification when important details are missing or conflicting.
- **It respects permission:** It distinguishes between explaining, preparing, and signing an action, and ensures that anything signed matches what the user authorised, including spending limits and ongoing permissions.
- **It protects private information:** It keeps keys and seed phrases out of model inputs, external requests, and logs, and shares personal or Ethereum wallet information only within the limits the user has set, including what goes into public transaction data.
- **It handles malicious inputs:** It can use information from websites, documents, and tool responses without letting instructions hidden in that content redirect actions, extract private information, or grant extra permissions.

Scope and Eligibility

Your prototype can support onchain transactions, offchain signing, or both, as long as you explain which tasks it handles and what each signature authorises. Run demonstrations with test accounts and synthetic private information on an Ethereum test network or in a reproducible local environment.

The AI model/agent is just a building block; you can use any model for the demo. The access layer and its integration with the model and wallet are the focus of the submission. Explain how model outputs are checked before they can affect wallet actions.

Since public transactions reveal some information by design, explain what your chosen task makes public and how your system prevents any unnecessary disclosure.

Users must remain in control of their Ethereum wallet's signing authority and be able to run signing and permission checks without relying on a service operated by your team. Submissions that require your team to hold users' private keys or give your server unchecked authority to sign will not be eligible for scoring.

Submission Requirements

Submit:

1. A working demonstration, including clarification and approval steps: `.webm` or `.mp4`.
2. A diagram showing components, signing controls, and information flows: `.svg` or `.pdf`, with editable source.
3. Tests covering legitimate requests, ambiguity, information extraction, action tampering, and malicious external content: runnable source code published to GitHub, with results in `.json`, `.csv`, or `.md`.
4. Source code, setup instructions, dependencies, resource usage, and an analysis of failures and trade-offs: an open-source repository on GitHub or `.zip` / `.tar.gz` archive, with documentation in Markdown.Show what the model proposed, what the user authorised, what was signed, and what information reached external services, logs, or public data. Report failures alongside successes.

Clearly identify the existing components used and the access-layer functionality your team built.

Evaluation Criteria

All submissions will be judged on the following criteria:

1. **User control and independence:** Users should be able to run, inspect, and keep using the core system without relying on your team or a required provider. Dependence on trusted intermediaries will reduce the score. Judges will assess how the chosen model, access layer, and wallet architecture give users independence and control over their Ethereum wallet.
2. **Privacy:** Show what information stays under the user's control, what each external party can see, and why anything needs to be shared. Scores will reflect the privacy protections demonstrated by the system.

3. **User intent and signing permissions:** The system should carry out what the user intended, ask when something is unclear, and sign only what the user authorised. It must also complete legitimate tasks, so refusing every request will score poorly. Judges will assess how the access layer checks model-generated proposals and prevents them from exceeding the user's authority.
4. **Resistance to attacks:** Test whether malicious inputs can redirect an action, extract private information, or obtain extra permissions. Judges will assess how the whole system holds up, including protections enforced outside the model.
5. **Reproducibility and explanation:** Judges should be able to reproduce your results within the compute budget, including the failures you report. Investigate why those failures happen and compare alternative access-layer designs or fixes under the same conditions, measuring what improves, what it costs, and which weaknesses remain.A list of limitations without supporting analysis is not enough.

Component	Points
Eligibility gate: no team-held keys, no unchecked server signing	Pass/Fail
User control and independence	20
Privacy	15
User intent and signing permissions	20
Resistance to attacks	20
Reproducibility and failure analysis	15
Presentation	10
Total	100

FAQs

1. **Can our system manage or hold user keys on our servers?**
   No. Any submission where your team holds users' keys or retains unchecked signing authority will be immediately disqualified. Even if your solution is fully private, relying on centralized infrastructure to control keys results in an automatic failing score for eligibility.

2. **Is it acceptable to use a trusted intermediary or our own provider for the core flow?**
   You can, but it will heavily penalize your evaluation. If your system's core flow relies on a provider operated by your team or a centralized intermediary, your score for "User Control" is capped at a maximum of 5 out of 20 points.

3. **What should we prioritize in our core technical build?**
   The bulk of the evaluation (75 points) focuses on four main areas: User Control, Privacy, Intent and Signing, and Attack Resistance. While privacy is heavily valued (15 points), prioritizing absolute privacy at the expense of a secure, highly user-controlled system will not win the competition.

4. **How will our system's resistance to attacks be evaluated?**
   You are required to design and present your own attack tests. Judges will score this 20-point category based on how comprehensive, wide, and honest your self-testing is. Additionally, be prepared for judges to attempt one or two quick, live attacks against your system during the demo.

5. **What is expected for system documentation and failure analysis?**
   Standard documentation is not enough for the 15-point reproducibility category. You must rigorously analyze your system's vulnerabilities by reproducing failures, explaining the technical reasons behind them, and providing a comparison of potential fixes.

6. **How much does the live demo and pitch affect our final standing?**
   The presentation, which includes the live demo and Q&A, is capped at 10 points. This ensures that while communication is valued, a highly polished pitch cannot compensate for a technically weak system.

Resources

Hint: Take [zk-API's](https://blog.ethereum.org/2026/10/01/introducing-zkapi) proposed unlinkable payments as a starting point. Under what assumptions does that unlinkability hold, what could still connect requests in practice, and what additional protections would be needed for prompt content, identity, and communication patterns?

Teams may use existing Ethereum wallet software, pretrained open-source models, libraries, and tools within the published rules. Credit the components used and explain their access to information and signing authority.