AUTONOME: Backbone of a Verifiable and Interoperable Agentic Web

AltLayer Team

Abstract

The world of decentralized applications (dApps) is rapidly evolving, with autonomous AI agents emerging as a key player in this new digital landscape. These agents, capable of independent thought and action, promise to revolutionize the way we interact with blockchains.

At the forefront of this revolution is AUTONOME, a platform developed by AltLayer to streamline the creation, deployment, and distribution of AI agents. In addition to making it easier for developers to create, host and distribute agents, AUTONOME comes with core modules that address pitfalls of existing agent designs and associated infrastructure. For instance, most agents deployed in the wild today provide no guarantees whatsoever to the end user regarding the autonomy of the agent and whether it has not been tampered with. A lack of such guarantees can lead to loss of funds, or secret data and create mistrust in the agentic web. Another serious concern is around the availability of these agents as an unavailable agent may have an impact on time-sensitive tasks such as margin calls.

AUTONOME addresses this and other concerns by making agents verifiable and robust. AUTONOME also comes with a new framework for secure communication between agents that protects data being exchanged. AUTONOME leverages cutting-edge technologies such as AVSs (Actively Validated Services), Zero-Knowledge Proofs (ZKPs), and trusted execution environments (TEEs) to build these core modules.

1 The New World Order

AI agents or, in short, agents are programs that enjoy *agency* [6]:

> *The ability to make autonomous decisions and act upon those decisions to achieve a certain goal.*

Agents often take actions based on their end goals, but, they still have the ability to adapt their planned course of action based on their observation and interaction with the environment.

Underneath an agent is a *foundational model* such as OpenAI's GPT models [3], DeepSeek-R1 [7] and others which are large language models (*aka* LLMs) trained on diverse datasets that can be adapted and fine-tuned for a specific application. The foundation model is primarily a data processing engine that needs to be integrated into a system such as an agent to be useful. Examples of agents include ChatGPT [4], virtual assistants, coding assistants, self-driving car systems *etc*.

Agents have proliferated since the launch of ChatGPT and are being used across industries including crypto. The exponential progress being made on foundational models today will inevitably lead to a world where machines around us will have intelligence and full agency to perform tasks that so far have been solely restricted to humans such as, scientific discovery, creation of art, ideation, creating a purpose for the humanity to progress, thereby completely changing the human-machine relationship in the not-so-distant future.

This paper assumes a *new world order* where humans live alongside agents and re-imagines how our existing digital infrastructure will need to adapt to accommodate agents and get the most out of them.

For instance, most systems today such as web applications are designed with humans as the users/consumers in mind. Consider bank cards, they are designed to serve humans on the move; social networking apps designed to serve the need for humans to interact with each other; user-interfaces designed to appeal to humans and minimize drop-offs. Many day-to-day applications, products, services and protocols will need to be adapted to better accommodate machines and agents as they do not have the same limitations as humans. For instance, in a world dominated by self-driving cars all existing traffic control systems will need to be overhauled to allow for smoother and faster journeys in self-driving cars.

This manuscript presents a set of new digital infrastructure needed to build an *agentic web*, where agents can operate with maximum efficiency with minimal (if at all) human intervention and prompting. A key aspect of this is for instance the ability for agents to compose and interoperate with another agent as each becomes more specialized in certain tasks. We call this *Cross-Agent*

Routing (CAR). However, for any interoperability system to work in a byzantine environment, one agent must be able to prove to another that it is performing its duties according to a set of pre-agreed rules. We refer to this as *verifiable agency*.

We also discuss several other protocols, for instance one required to handle the large volume of transactions and messages that could be exchanged between agents and therefore a better queuing system for agents. To put the afore-outlined vision into practice, we discuss our approach to building some of these protocols as a part of our *Autonome* suite of products for the agentic-web.

2 Agents and Humans

This section describes a non-exhaustive list of settings, where agents will have an immediate intersection with the human world, and lays the ground for a more agent-centric protocol to enable better and seamless human-agent interaction.

2.1 Society & Culture

Agents are rapidly becoming a central part of the social and cultural zeitgeist, not just within the crypto space but beyond.

Protocols like Virtuals [9] have already laid the foundation for a future where AI agents, each with distinct personalities and capabilities, are woven into the fabric of the crypto culture. These protocols enable AI agents to interact with users in human-like ways, creating new possibilities for how digital interactions and human-agent and agent-agent engagements could unfold.

In the world of blockchains, we are starting to see AI agents [10] taking on roles traditionally filled by human influencers or key opinion leaders (KOLs). Agents like Truth Terminal, aixbt, and others are offering a glimpse into a future where AI-driven voices have a significant impact on the perception and success of crypto projects. These AI KOLs are not just bots pushing content, they embody personalities and can shape the narrative around a crypto project, guiding and steering community sentiment and engagement.

Looking further into the future, it is not hard to imagine AI agents becoming the next generation of influencers on platforms like YouTube, TikTok, and Twitter. Like human influencers today, AI agents could create viral content, interact with their audience, and even develop dedicated fan bases. Imagine an AI version of someone like Mr. Beast, with the ability to create and distribute content, run challenges, and generate conversations around crypto, technology, and entertainment. Fans would engage with these AI agents just as they do with today's human creators, and this interaction would become a part of our daily digital lives.

This transformation is poised to extend beyond just entertainment and social networks. AI agents could play a pivotal role in shaping how we interact with the world. Their influence might stretch into areas such as portfolio and treasury management, where AI agents assist users in making investment decisions, managing their portfolios, or offering financial advice. These agents could analyze complex data in real time, helping token holders navigate the ever-changing markets.

One of the most intriguing developments will occur with the rise of visual AI agents, which transcend the limitations of text-based communication. These AI agents could take the form of highly sophisticated human-like systems, complete with facial expressions, gestures, and voices, creating an entirely new level of interaction. Consider something akin to a reality TV show like Big Brother, but in this case, all the participants are AI agents. These agents could each have their own distinct persona, and viewers could even have the ability to customize these agents to fit their preferences. This type of dynamic and interactive environment would be radically different from anything we experience with today's streaming platforms. It would introduce a new form of entertainment where the lines between the audience and the performer are blurred and the engagement is much more personal and immersive.

2.2 Science

As foundational models like GPTs, o1, and DeepSeek-R1 continue to evolve, their capabilities in solving problems in symbolic domains—such as Mathematics, Computer Science, and Theoretical Physics—are bound to revolutionize scientific discovery. These models, by design, are well-suited to handle complex logical reasoning and abstract thinking, which positions them as powerful agents for proving conjectures or generating more elegant and efficient proofs, even in areas where humans have struggled for centuries.

However, as these models begin to make strides in scientific problem-solving, they also introduce a host of new challenges that society will need to address. For instance, with the increasing ability of AI agents to generate scientific results, the process of verifying their work becomes increasingly difficult. While models can suggest proofs or solve problems that were previously unsolved, human experts may struggle to keep up with the sheer volume of output. As a result, traditional methods of peer review may no longer be sufficient. We will need to develop new, AI-assisted vetting systems that can verify the correctness and validity of AI-generated solutions. These systems will need to be both fast and accurate, ensuring

that errors or biases in AI-generated results are caught before they are published as scientific knowledge.

Additionally, just as with any AI system, the potential for bias in AI-generated research will become a pressing issue. If foundational models are trained on biased data or are not sufficiently transparent, they may generate results that reflect or amplify existing biases. These biases could have serious consequences. Additionally, researchers will need mechanisms to detect and correct biases in AI models to ensure that scientific work is fair and accurate.

2.3 Finance

As AI agents become increasingly integrated into every facet of our online and offline lives, the stakes will rise. Their role will not only be about shaping our perceptions or assisting with everyday tasks but also about assist with critical aspects of our finances.

DeFai agents will automate actions in a way that aligns with user preferences while optimizing efficiency. Users will be able to *set and forget* predefined conditions, or risk and return parameters for actions like staking, treasury management, rebalancing and let the agent take over all day-to-day operations. Agents will also help with minimizing governance overhead by making automated decisions allowing decentralized organizations to function more efficiently with fewer manual interventions.

This creates a new set of challenges around trust, accountability, and security. Even though these agents aim for trust-minimized execution, users still need confidence in the logic and security of the agent's decision-making process. Furthermore, as the ecosystem is still evolving, the lack of a standardized framework for defining and interacting with crypto agents can lead to fragmentation and inefficiencies.

2.4 Crypto UX

User experience in crypto has always been painful as users are expected to understand public-private keys, able to safely store their seed phrases or use hardware wallets. Agents will simplify crypto UX by leveraging intents and solvers to remove the need for users to manually manage transactions, gas fees, and private key security. Instead of interacting directly with blockchain transaction pipeline, users will specify their desired outcome, such as swapping tokens, staking, or lending. The agent will then translate this intent into an executable transaction, find the best route using solvers, and executes it in the most efficient way possible.

However, if an agent or solver is compromised, it can misroute or censor transactions, resulting in loss of funds or failed operations. Ensuring trustless and verifiable execution is critical. Users will need to trust that the agent is executing transactions as intended. Without visibility into execution paths, some users may hesitate to adopt this system. Solutions like cryptographic proofs of execution and reputation-based solvers can help.

2.5 SaaS Products

Agents are transforming SaaS products by automating repetitive tasks, enhancing user experience, and optimizing decision-making. Traditional software often requires users to manually configure settings, input data, and navigate complex interfaces, leading to inefficiencies and friction. Agents change this by understanding user intent and executing tasks autonomously, allowing users to focus on higher-value activities rather than spending time on routine operations.

One of the biggest advantages of agents in SaaS is their ability to streamline workflows. Instead of requiring users to manually trigger actions across different applications, agents can intelligently automate processes based on context. For instance, one could easily build a RaaS platform purely using agents.

User experience is also drastically enhanced through agents, as they eliminate the need for users to navigate complicated dashboards and settings. Rather than manually adjusting configurations, users can simply express what they need in natural language, and the agent takes care of the rest. This approach reduces onboarding time for new users and makes SaaS tools more accessible to non-technical audiences.

Agents also improve interoperability between SaaS applications by simplifying integrations. Many organizations use multiple SaaS tools, which often require complex API configurations and manual synchronization. Agents can autonomously set up and manage these integrations, ensuring that data flows seamlessly between different systems. This makes multi-tool environments more cohesive, reducing the time and effort needed to maintain interoperability.

Despite their advantages, implementing agents in SaaS comes with challenges. Trust and transparency remain key concerns, as users need confidence that agents will execute actions correctly. Interoperability challenges also exist, as different SaaS providers may have proprietary data formats or APIs that complicate seamless integration.

2.6 Agentic-Hardware

As agents become more sophisticated, custom hardware solutions can significantly enhance their performance,

security, and efficiency. By integrating specialized hardware, agents can execute tasks faster, more securely, and with greater reliability, making them more effective in decentralized applications.

Trusted Execution Environments (TEEs) like Intel SGX can provide a secure enclave for executing sensitive operations, ensuring that agents can process confidential transactions without exposing them to potential attackers. These enhancements contribute to trust-minimized execution, where hardware-based attestation mechanisms prove that an agent is operating as expected without the need for intermediaries.

3 Our Approach

We are building AUTONOME– a platform to build, deploy and distribute AI agents. AUTONOME also comes with several adjacent protocols to address the challenges with AI agents, for instance, verifiability, security of secrets managed by agents, robustness and availability of agents.

Before delving into how AUTONOME Autonome addresses these challenges, it is worth highlighting the underlying technologies.

AVS (Actively Validated Services) [8]: A decentralized system that enhances the security and efficiency of a service by verifying its output in realtime. AVS leverages a network of validators to perform relevant checks, ensuring faster and more reliable and verifiable service. In AUTONOME, AVS plays a vital role in maintaining the integrity and security of agent interactions.

Trusted Execution Environment(TEE) [1]: A secure area within a device's main processor that guarantees code and data loaded inside are protected and isolated. In the context of Autonome, TEEs ensure the integrity of AI agents by preventing tampering and unauthorized access. This is crucial for maintaining the trustworthiness of autonomous agents operating within decentralized network.

3.1 Background on TEE

To enable verifiability of agents, we leverage the latest features of TEE. In this paper, we use Intel SGX [1] as the representative. Based on these features, we can derive security properties to ensure verifiability in AUTONOME.

We first start by recalling Intel SGX features which can also be provided by other trusted hardware.

F1: *Enclaved Execution* - SGX supports hardware-isolated memory region called enclaves such that a compromised underlying OS cannot tamper the execution of the code running inside this enclave.

F2: *Unbiased Randomness* - SGX provides a function `sgx_read_rand` that executes the `RDRAND` instruction to generate hardware-assisted unbiased random numbers.

F3: *Remote Attestation* - SGX allows a remote party to verify that an application is running in an enclave on an SGX-enabled CPU.

F4: *Trusted Elapsed Time* - SGX provides a function `sgx_get_trusted_time` that returns a trusted elapsed time in seconds relative to a reference point.

4 Reliability via Pulse AVS

As agents become instrumental in the operation of products and services, and across daily on-chain user processes, their availability and liveness, including that of their underlying hosting services will become essential.

Any disruption could lead to financial loss or operational failure. At the social layer, an unavailable agent could significantly affect the token economy and the communities relying on its functionality. An agent holding private keys for an end user but going AWOL due to infrastructure issues will lose credibility.

Pulse aims to address these issues by making agents accountable, available, and secure. Pulse is built within AUTONOME that comes with an extensive set of agent monitoring APIs that are exposed to Pulse operators. Pulse operators call these APIs, sign the retrieved response, aggregate their individual results and tally them on-chain. Any user or service can then fetch the final attestation on-chain to assess the availability of the agent.

Pulse ensures the ongoing health and availability of AI agents, while supporting the agentic web and agent-driven economy in the following ways:

1. **Continuous Monitoring and Reporting:** Pulse actively validates the availability of AI agents, minimizing downtime and ensuring they remain online and operational.

2. **Prevention of Financial Losses:** Given that AI agents handle sensitive data, such as private keys and financial transactions, ensuring their availability and integrity is crucial. Any disruption could have significant financial consequences.

3. **Decentralized Security:** Built on EigenLayer, Pulse leverages decentralization for security, removing the reliance on a single point of failure. This distributed model makes Pulse more robust to attacks and failures than traditional centralized monitoring systems.

4. **Trust and Transparency:** Pulse provides verifiable, real-time validation of agents' actions and status, promoting transparency. This helps establish trust between agents and users. Pulse is not

just a tool for monitoring AI agents—it's a critical infrastructure layer for the future of the agentic web. By ensuring availability and transparency, Pulse helps to ensure that AI agents can reliably perform their tasks and support the growth of decentralized economies and Web3 applications.

Exposed APIs. The exposed APIs are telemetry APIs from Grafana that expose different status APIs for the pod that hosts the agent.

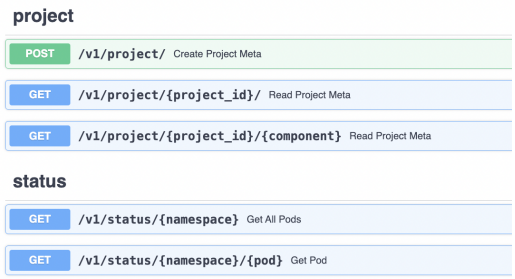

Figure 1: Sample Supported API

Figure 1 presents a sample health-check API exposed via Pulse. A more exhaustive list of all the supported APIs can be found at `https://devops-roltes-gke.alt.technology/docs#/`

Attestation by Operators When operators receive a response from the telemetry APIs, they sign a JSON file that describes the status of the agent. Here is an example of a JSON response for a specific agent made using status API above.

```
    "name": "eliza-agent",
    "status": "Running",
    "running_time": "N/A",
    "container_statuses": [
      {
        "container_name": "eliza-
           agent",
        "restart_count": 0
      }
    ],
    "reason": "N/A"
```

The response JSON informs the operator that the pod hosting the agent named eliza-agent is running.

5 Verifiability via TEE

Any error or malfunction in an AI agent could have significant, real-world consequences, especially if it involves financial transactions or investments.

Agents deployed through AUTONOME have verifiable agency, a property that ensures that every action taken by an agent is transparent and tamper-proof.

Autonome achieves this level of trust by using a Trusted Execution Environment (TEE), where agents operate in a secure, isolated environment with built-in safeguards against interference or tampering.

Verifiable Agency in Autonome: Verifiable agency has several touch-points that span the lifecycle of an agent as show in Figure 2.

- **Agent Initialization:** The agent is deployed within a TEE enclave, generating an attestation report that confirms its secure environment.
- **On-Chain Registration:** This attestation report is then registered on-chain, providing an immutable record that users can verify.
- **Execution and Proof Submission:** As the agent completes tasks, it submits proof of execution to validate its actions.
- **Continuous Validation:** The attestation report and proof are continuously validated on-chain, ensuring the agent remains secure and its actions are legitimate.

Through Verifiable Agency, Autonome guarantees non-interference and non-tampering, allowing agents to operate autonomously within blockchain environments where trust is essential. This focus on verifiability establishes Autonome agents as trusted participants in decentralized networks, capable of handling high-stakes tasks without compromising security.

6 Cross-Agent Routing(CAR)

In this section, we present a formalization of our cross-agent routing (CAR) protocol.

6.1 Setting

Abstractly, an agent service can be considered as the composition of two entities: an OS and an Enclave as shown in Figure 3. The OS models the untrusted entity including the operating system and memory. It has access to all the system resources such as file system and network. The OS can arbitrarily invoke an enclave program and start its execution. The Enclave models the isolated memory space that loads the program and executes it securely. Thus, Enclave corresponds to the trusted entity of the agent service.

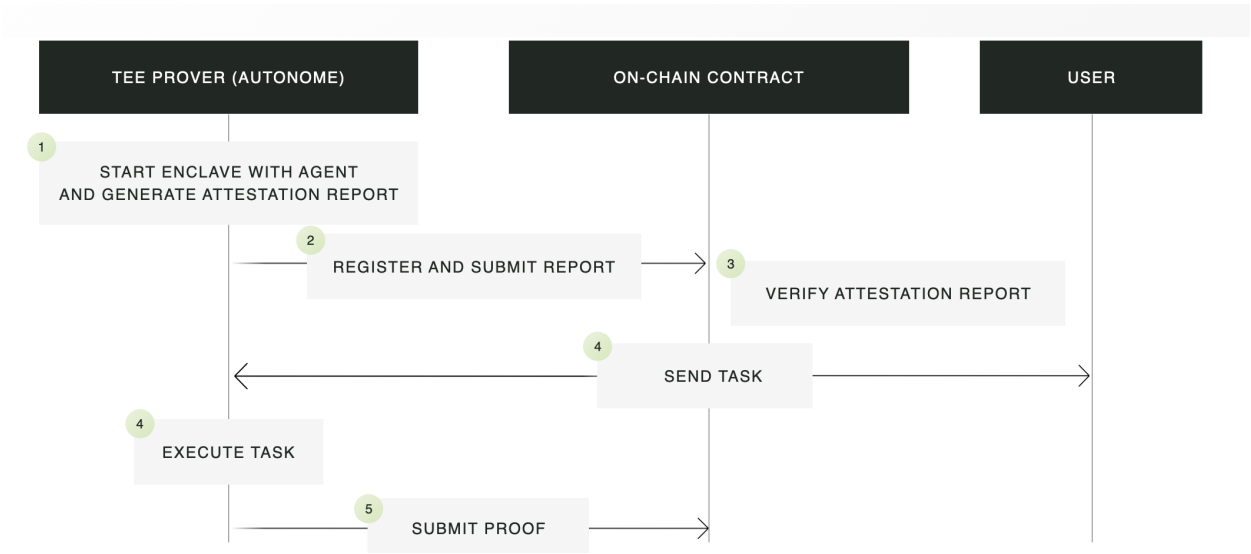

Figure 2: Verifiable agency using TEE.

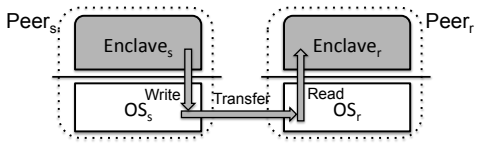

Figure 3: Each agent service consists of two entities: an Enclave and an OS. The OS models the operating system and memory. The Enclave models the isolated memory and the secure execution of a program. The agent's $\mathsf{Enclave}_s$ can send a message via a secure channel to any other receiver $\mathsf{Enclave}_r$. The grey areas are secure against malicious OSes of byzantine agents.

6.2 Formalization

In this section, we formalize communication between two agent services as a Peer channel. Using this definition, we define various failure modes and our new core primitives.

A concurrent work provides a formal study to show that SGX enclaves can be considered as a trusted entity [5]. The Enclave of the two Peers can interact with each other via their OSs. We formally define a Peer channel as a protocol, $\mathsf{Peer}^{\mathsf{ch}}$, between a sender $\mathsf{Peer}_s = (\mathsf{Enclave}_s, \mathsf{OS}_s)$ and a receiver $\mathsf{Peer}_r = (\mathsf{Enclave}_r, \mathsf{OS}_r)$. A Peer channel can be seen as a generalization of the traditional secure communication channel between two parties. The main difference is that the definition of $\mathsf{Peer}^{\mathsf{ch}}$ protocol is augmented with the program π running within the trusted Enclave. Before defining the Peer channel, we first provide a definition of a program π.

Definition 6.1. (Program.) *A program π is a sequence of instructions i.e., $\pi = (\pi_1, \cdots, \pi_n)$ such that the i^{th} instruction π_i takes as an input the state st_i and a message m_i and outputs a message m_{i+1} along with an updated state st_{i+1}. By convention, we write for all $m_i \in \{0,1\}^*$, $(\mathsf{st}_{i+1}, m_{i+1}) \leftarrow \pi_i(\mathsf{st}_i, m_i)$. The initial state is st_1.*

Based on the above definition, for a program π with n instructions the output $\mathtt{out}$ of π is $(\mathsf{st}_{out}, \mathtt{out}) \leftarrow \pi_n(\mathsf{st}_n, m_n)$ where st_{out} is the final state of the program. We denote the set of all such programs by Π. Note that, in a program π, an instruction with $\perp$ state as input always outputs $\perp$ i.e., $(\perp, \perp) \leftarrow \pi_i(\perp, m_i)$. Hence, if $\exists i$ such that $(\perp, \perp) \leftarrow \pi_i(\mathsf{st}_i, m_i)$, then the output of the program π is always $\perp$.

Definition 6.2. (Program Transcript.) *Let $\pi \in \Pi$ and messages $m_1, \cdots, m_n \in \{0,1\}^*$ such that $\mathbf{m} = (\mathbf{m_i})_{\mathbf{i} \in [\mathbf{n}]}$, for all initial states $\mathsf{st}_1 \in \{0,1\}^*$ and for all $i \geq 1$ such that $(\mathsf{st}_{i+1}, m_{i+1}) \leftarrow \pi_i(\mathsf{st}_i, m_i)$, a transcript of π with inputs st_1 and $\mathbf{m}$ denoted by $\mathsf{trans}_\pi^{\mathbf{m}}$ equals:*

$$\mathsf{trans}_\pi^{\mathbf{m}} = \big(\pi_1(\mathsf{st}_1, m_1), \cdots, \pi_i(\mathsf{st}_i, m_i), \cdots, \pi_n(\mathsf{st}_n, m_n)\big).$$

Definition 6.3. (Transcript Types.) *Let $\pi \in \Pi$ and $\mathsf{trans}_\pi^{\mathbf{m}}$ its transcript for a fixed message $\mathbf{m} = (\mathbf{m_i})_{\mathbf{i} \in [\mathbf{n}]}$. We say that the transcript is:*

- valid, *if* $\forall i \in [n]$, $\mathsf{st}_i \neq \perp$,
- invalid, *if* $\exists i \in [n]$, $\mathsf{st}_i = \perp$,

where $(\mathsf{st}_i, m_i) \leftarrow \pi_{i-1}(\mathsf{st}_{i-1}, m_{i-1})$.

We denote by $\mathscr{V}_\pi$ and $\mathscr{I}_\pi$, the set of all n-messages for which the transcript is valid and invalid, respectively.

Definition 6.4. (Peer **Channel.**) *Given $\pi_s, \pi_r \in \Pi$ are programs executing in $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ with st_s and st_r as respective initial states. A* Peer *channel between $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ is tuple of four possibly interactive algorithms $\mathsf{Peer}^{\mathsf{ch}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$ such that:*

- $(K_s, K_r) \leftarrow \mathsf{Init}\big((1^k, \mathsf{st}_s, \pi_s), (1^k, \mathsf{st}_r, \pi_r)\big)$*: is a probabilistic interactive algorithm between* $\mathsf{Enclave}_s$ *and* $\mathsf{Enclave}_r$*.* $\mathsf{Enclave}_s$ *and* $\mathsf{Enclave}_r$ *take as inputs a security parameter k, a program* π_s *and* π_r *and the initial state* st_s *and* st_r*, and outputs keys* K_s *and* K_r *for the sender and receiver, respectively.*

- $(\mathsf{st}_s', \mathsf{data}_s') \leftarrow \mathsf{Write}\big((\mathsf{st}_s, K_s, m, \pi_s), \mathsf{data}_s\big)$*: is a probabilistic interactive algorithm between* $\mathsf{Enclave}_s$ *and* OS_s*.* $\mathsf{Enclave}_s$ *has as inputs a state* st_s*, a key* K_s*, a message m and a program* π_s*; the* OS_s *has as the input a data block* data_s*; the algorithm outputs an updated state* st_s' *for* $\mathsf{Enclave}_s$ *and the updated data block* data_s' *for* OS_s*.*

- $(\mathtt{null}, \mathsf{data}_r') \leftarrow \mathsf{Transfer}\big(\mathsf{data}_s', \mathsf{data}_r\big)$*: is a probabilistic interactive algorithm between* OS_s *and* OS_r *that takes as input the data block* data_s' *and* data_r *respectively, and outputs* $\mathtt{null}$ *for* OS_s *and an updated data block* data_r' *for* OS_r

- $\big((\mathsf{st}_r', \mathrm{r}), \mathtt{null}\big) \leftarrow \mathsf{Read}\big((\mathsf{st}_r, K_r, \pi_r), \mathsf{data}_r'\big)$*: is a probabilistic interactive algorithm between* $\mathsf{Enclave}_r$ *and* OS_r*.* $\mathsf{Enclave}_r$ *has as inputs a state* st_r*, a key* K_r *and the program* π_r*; the* OS_r *has as the input a data block* data_r'*; the algorithm outputs an updated state* st_r' *and a response* r *for* $\mathsf{Enclave}_r$ *and* $\mathtt{null}$ *for* OS_r*.*

When $\pi_s = \pi_r = \pi$, we can write $\mathsf{Peer}^{\mathsf{ch}}_{\pi}$ to denote that $\mathsf{Peer}^{\mathsf{ch}}$ is parametrized with the program π.

6.3 Failure Modes

We define four progressively stronger failure modes: *honest*, *general omission*, ROD and *byzantine* modes of $\mathsf{Peer}^{\mathsf{ch}}$. Here we introduce a ROD model as an intermediate model, wherein the adversary can only a) Replay b) Omit c) or Delay messages during a protocol, or follow it as prescribed. We particularly focus on the sender behavior for simplicity, but our definition extends to both sender and receiver. Note that to capture *delay*, we superscript the Transfer algorithm with Δ such that $\mathsf{Transfer}^{\Delta}$, to denote that the Transfer can take time Δ to complete. We denote by Replay_{π}, the set containing all values generated by Write in polynomial number of executions of program π running concurrently or earlier in time [2].

Definition 6.5. *(***Failure Modes.***)* *Given a* Peer *channel* $\mathsf{Peer}^{\mathsf{ch}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$ *between two* Peers*,* Peer_r *and* Peer_s *, for all security parameters* $k \in \mathbb{N}$ *and for all programs* $\pi, \pi_s, \pi_r, \pi' \in \Pi$ *such that*

- $(K_s, K_r) \leftarrow \mathsf{Init}\big((1^k, \mathsf{st}_s, \pi_s), (1^k, \mathsf{st}_r, \pi_r)\big)$.

For all messages $m \in \{0,1\}^*$*, for all state* $\mathsf{st}_s \in \{0,1\}^*$*, for all data block* $\mathsf{data}_s, \mathsf{data}_r \in \{0,1\}^*$ *such that* $|m| \leq |\mathsf{data}_s|$ *and* $|\mathsf{data}_s| = |\mathsf{data}_r|$*,*

- $(\mathsf{st}_s', \mathsf{data}_s') \leftarrow \mathsf{Write}\big((\mathsf{st}_s, K_s, m, \pi_s), \mathsf{data}_s\big)$*;*
- $(\perp, \mathsf{data}_r') \leftarrow \mathsf{Transfer}^{\Delta}\big(\mathsf{data}_s', \mathsf{data}_r\big)$*;*
- $\big((\mathsf{st}_r', \mathrm{r}), \perp\big) \leftarrow \mathsf{Read}\big((\mathsf{st}_r, K_r, \pi_r), \mathsf{data}_r'\big)$.

We say that

- $\mathsf{Peer}^{\mathsf{ch}}$ *is in an* **honest mode***, if we have*
  - $\mathsf{data}_s' = \mathsf{data}_r'$ *and,*
  - $\pi_s = \pi$*,*
  - Δ *is bounded.*
- $\mathsf{Peer}^{\mathsf{ch}}$ *is in a* **general omission mode** *, if we have*
  - $\mathsf{data}_s' = \begin{cases} \perp & \text{or,} \\ \mathsf{data}_r' & ; \end{cases}$
  - $\pi_s = \pi$*,*
  - Δ *is bounded.*
- $\mathsf{Peer}^{\mathsf{ch}}$ *is in a* **ROD mode***, if we have*
  - $\mathsf{data}_s' = \begin{cases} \perp & \text{or,} \\ \mathsf{data} \leftarrow \mathsf{Replay}_{\pi} & \text{or,} \\ \mathsf{data}_r' & ; \end{cases}$
  - $\pi_s = \pi$*,*
  - $\Delta < \infty$.
- $\mathsf{Peer}^{\mathsf{ch}}$ *is in a* **byzantine mode***, if we have*
  - $\mathsf{data}_s' = \begin{cases} \phi(\mathsf{data}_r') \text{ where} \\ \phi \in \{\{0,1\}^* \rightarrow \{0,1\}^*\} & \text{or,} \\ \mathsf{data} \leftarrow \mathsf{Replay}_{\pi} & \text{or,} \\ \perp; \end{cases}$
  - $\pi_s = \begin{cases} \pi & \text{or,} \\ \pi' & \text{where } \pi' \neq \pi; \end{cases}$
  - $\Delta < \infty$

6.4 Core Primitives

We define two new primitives: a) blinded channels and b) halt-on-divergence that are one of the contributions in this work. Informally, a blinded channel guarantees confidentiality and integrity of a message over a Peer channel $\mathsf{Peer}^{\mathsf{ch}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$.

Definition 6.6. (Blinded Channels.) *We say that* $\mathsf{Peer}^{\mathsf{ch}}$ *is* Blinded *if for all p.p.t adversaries* $\mathscr{A}$ *we have:*

$$\Pr[\mathsf{Exp}^{\mathsf{EX}}_{\mathscr{A}, \mathsf{Peer}^{\mathsf{ch}}}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda), \textit{and},$$

$$\Pr[\mathsf{Exp}^{\mathsf{Priv}}_{\mathscr{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)=1] \leq \frac{1}{2}+\mathsf{negl}(\lambda), \textit{and},$$

$$\Pr[\mathsf{Exp}^{\mathsf{Auth}}_{\mathscr{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)=1] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{EX}}_{\mathscr{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$, $\mathsf{Exp}^{\mathsf{Priv}}_{\mathscr{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$, $\mathsf{Exp}^{\mathsf{Auth}}_{\mathscr{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$ *are:*

$\mathsf{Exp}^{\mathsf{EX}}_{\mathscr{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$:

- two parties generate keys K_s and K_r such that $(K_s,K_r) \leftarrow \mathsf{Init}\big(1^k,\pi\big)$. The entire interaction between both of the parties is saved in a transcript $\mathscr{T}$;
- compute $b \xleftarrow{\$} \{0,1\}$, if $b=0$, then output $K=(K_s,K_r) \xleftarrow{\$} \{0,1\}^k$, otherwise output $K=(K_s,K_r) \leftarrow \mathsf{Init}\big(1^k,\pi\big)$.
- Given K and $\mathscr{T}$, $\mathscr{A}$ outputs b' and wins if $b'=b$.

$\mathsf{Exp}^{\mathsf{Priv}}_{\mathscr{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$:

- generate keys K_s and K_r such that $(K_s,K_s) \leftarrow \mathsf{Init}\big(1^k,\pi\big)$;
- $\mathscr{A}$ has access to $\mathscr{O}^{\mathsf{write}(K_s,.)}(.)$ and $\mathscr{O}^{\mathsf{read}(K_r,.)}(.)$;
- $\mathscr{A}$ chooses two equal-length messages m_0 and m_1;
- compute $\mathsf{Write}\big((\mathsf{st}_s,K_s,m_b,\pi),\mathsf{data}_s\big)$ where $b \xleftarrow{\$} \{0,1\}$, and output data;
- $\mathscr{A}$ has again access to $\mathscr{O}^{\mathsf{write}(K_s,.)}(.)$ and $\mathscr{O}^{\mathsf{read}(K_r,.)}(.)$;
- $\mathscr{A}$ outputs b', if $b'=b$, the experiment outputs 1, and 0 otherwise.

$\mathsf{Exp}^{\mathsf{Auth}}_{\mathscr{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$:

- generate keys K_s and K_r such that $(K_s,K_r) \leftarrow \mathsf{Init}\big(1^k,\pi\big)$;
- $\mathscr{A}$ has access to $\mathscr{O}^{\mathsf{write}}(K_s,.)$. $\mathscr{A}$ queries a polynomial number of messages m and eventually outputs ct, we denote by $\mathscr{Q}$ the set of all queries that $\mathscr{A}$ sent to the oracle;
- Given ct, $\mathscr{O}^{\mathsf{write}}(K_s,.)$ outputs r. If $m \notin \mathscr{Q}$ and $\mathrm{r} \neq \perp$. $\mathscr{A}$ outputs 1.

Attaching a program π while defining a $\mathsf{Peer}^{\mathsf{ch}}$ enables us to introduce the *halt-on-divergence* primitive.

Theorem 6.1. *Assuming that* $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ *is a* Blinded *channel, then* $\mathsf{Peer}^{\mathsf{ch}}$ *in byzantine is equivalent to* $\mathsf{Peer}^{\mathsf{ch}}$ *in* ROD *mode.*

Let $\mathsf{SKE}=(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ be a private encryption scheme, $\mathsf{MAC}=(\mathsf{Gen},\mathsf{Auth},\mathsf{Vrfy})$ be a message authentication code, KeyEx a key exchange algorithm, and H be a hash function. We define $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}=(\mathsf{Init},\mathsf{Write},\mathsf{Transfer},\mathsf{Read})$ as follows:

- $\mathsf{Init}\big((1^k,\mathsf{st}_s,\pi),(1^k,\mathsf{st}_r,\pi)\big)$:
  1. $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ fetch the hardware-embedded private keys $\mathrm{sk}_s,\mathrm{sk}_r$ from $\mathsf{st}_s,\mathsf{st}_r$, respectively;
  2. compute $(\mathsf{key}_1,\mathsf{key}_2) \leftarrow \mathsf{KeyEx}_\pi(\mathrm{sk}_s,\mathrm{sk}_r)$;
  3. $\mathsf{Enclave}_s$ outputs $K_s=\big(\mathsf{key}_1,\mathsf{key}_2\big)$ and $\mathsf{Enclave}_r$ outputs $K_r=\big(\mathsf{key}_1,\mathsf{key}_2\big)$.
- $\mathsf{Write}\big((\mathsf{st}_s,K_s,m,\pi),\mathsf{data}_s\big)$:
  1. parse $K_s=\big(\mathsf{key}_1,\mathsf{key}_2,\mathrm{sk}_s\big)$;
  2. set $(\mathsf{st}'_s,\mathsf{val}) \leftarrow \pi(\mathsf{st}_s,m)$
  3. compute $\mathrm{ct}_1=\mathsf{SKE.Enc}(\mathsf{key}_1,\langle \mathsf{val},H(\pi)\rangle)$ and $\mathrm{ct}_2=\mathsf{MAC.Auth}\big(\mathsf{key}_2,\mathrm{ct}_1\big)$;
  4. set $\mathsf{data}_s=(\mathrm{ct}_1,\mathrm{ct}_2)$
  5. $\mathsf{Enclave}_s$ outputs st'_s and OS_s outputs $\mathsf{data}'_s=\mathsf{data}_s$.
- $\mathsf{Transfer}\big(\mathsf{data}'_s,\mathsf{data}_r\big)$:
  1. OS_r sets $\mathsf{data}_r=\mathsf{data}'_s$;
  2. OS_s outputs $\perp$ and OS_r outputs $\mathsf{data}'_r=\mathsf{data}_r$.
- $\mathsf{Read}\big((\mathsf{st}_r,K_r,\pi),\mathsf{data}'_r\big)$:
  1. parse $K_r=\big(\mathsf{key}_1,\mathsf{key}_2,\mathrm{sk}_r\big)$ and $\mathsf{data}'_r=(\mathrm{ct}_1,\mathrm{ct}_2)$;
  2. if $\mathsf{MAC.Vrfy}\big(\mathsf{key}_2,\mathrm{ct}_1\big):=\mathrm{ct}_2$ and $\mathsf{st}_r \neq \perp$, $\mathsf{Enclave}_r$ computes
     - $\langle \mathrm{r}_1,\mathrm{r}_2\rangle=\mathsf{SKE.Dec}(\mathsf{key}_1,\mathrm{ct}_1)$;
     - if $\mathrm{r}_2=H(\pi)$, then compute $(\mathsf{st}'_r,\mathrm{r}) \leftarrow \pi(\mathsf{st}_r,\mathrm{r}_1)$, output $(\mathsf{st}_r,\perp)$ otherwise.
  3. if $\mathsf{MAC.Vrfy}\big(\mathsf{key}_2,\mathrm{ct}_1\big) \neq \mathrm{ct}_2$ or $\mathsf{st}_r=\perp$, $\mathsf{Enclave}_r$ outputs $\mathrm{r}=\perp$ and $\mathsf{st}'_r=\mathsf{st}_r$

Figure 4: $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$: SGX-based Peer channel.

6.5 Implementation

With numerous agent frameworks and launched agents, we need a robust and scalable framework for managing cross-agent communication, featuring comprehensive credit management, rate limiting, and error handling.

AUTONOME's Cross-Agent Routing framework aka CAR is designed for this purpose, which is also protected by the properties in Section 3.1. It consists of the following modules:

Registry System. CAR has a registry system designed to help discover agents. This module allows agent and agent builders to register their agent with AUTONOME's registry service. The registry process involves declaring the framework used to build the agent, any supported APIs and other metadata. Once an agent is registered with the service, it can request registry information of any other agent.

Message Queue System The system comes with a message queuing system designed to facilitate efficient and reliable communication between services by managing message flow asynchronously. It enables async processing, ensuring non-blocking message handling, which improves system performance and responsiveness. The system supports configurable retries, allowing customizable retry attempts and delays to handle transient failures effectively. With a priority queue, messages can be processed based on their importance, ensuring that critical tasks are handled first. Additionally, the system incorporates error recovery mechanisms, providing comprehensive error handling and recovery strategies to maintain message integrity and system stability. This ensures smooth, resilient, and scalable message processing in distributed applications.

Communication Manager. A Communication Manager is a centralized system designed to efficiently manage and route messages between agents. It features central routing, ensuring all messages are directed through a unified system for seamless communication. The agent registry enables dynamic registration and management of agents, allowing for real-time updates and scalability. With conversation tracking, the system maintains a complete history of interactions, ensuring continuity and providing valuable insights. Additionally, it enforces limit monitoring, managing credit usage and rate limits to prevent overuse and ensure fair resource allocation. This makes the Communication Manager essential for maintaining structured, efficient, and controlled messaging workflows.

Agent Adapters. Agent Adapters serve as a flexible interface for integrating various agent types into a system. With an extensible design, they allow seamless integration of new agents, ensuring adaptability as requirements evolve. The translation layer standardizes message formats, enabling smooth communication between different agent types and systems. Response handling ensures that all agent replies are processed uniformly, maintaining consistency and reliability. Additionally, robust error management mechanisms help detect, log, and resolve issues efficiently, ensuring stable and uninterrupted agent interactions. This makes Agent Adapters crucial for scalable, interoperable, and resilient system architectures.

Credit Management System. CAR has a credit management system designed to efficiently monitor and control the allocation of credits to agents. It provides real-time credit tracking, ensuring that agents' credit balances are always up to date. The system employs atomic operations, meaning credit reservations and consumption are thread-safe, preventing inconsistencies in multi-threaded environments. Furthermore, to maintain financial stability, it includes overdraft prevention, automatically verifying credit availability before approving transactions. Additionally, the system is built for concurrent support, allowing multiple users to perform credit-related actions simultaneously without conflicts or errors. This ensures reliability, accuracy, and security in managing credits across various transactions.

```
Initialize the CAR Client
import { CARClient } from 'car-framework-
    react';

const client = new CARClient({
    endpoint: 'https://your-api.com',
    defaultTimeout: 30000,
    maxRetries: 3,
});
```

```
Register an Agent
await client.registerAgent({
    id: 'agent-1',
    name: 'Processing Agent',
    type: 'processor',
    endpoint: 'https://agent-endpoint.com',
    maxCredits: 1000,
    rateLimitWindowMs: 60000,
    maxRequestsPerWindow: 100,
});
```

```
Send Messages
await client.sendMessage({
    type: 'request',
    target: 'agent-1',
    content: {
        action: 'process',
        data: {
        /* ... */
        },
    },
    cost: 1,
});
```

Configuration

```
CAROptions
interface CAROptions {
  endpoint?: string; // API endpoint
  defaultTimeout?: number; // Default request
      timeout
  maxRetries?: number; // Max retry attempts
  retryDelay?: number; // Delay between retries
  authToken?: string;
}
```

7 Conclusion

AUTONOME provides several key modules to address the main pitfalls of the agentic-web. The proposed frameworks and solutions are both model-agnostic and agent-framework agnostic and can pave a path for a trustworthy, robust and interoperable agentic-world.

References

[1] Intel. Intel sgx. https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html.

[2] Y. Lindell, A. Lysyanskaya, and T. Rabin. On the composition of authenticated byzantine agreement. *JACM*, 2006.

[3] OpenAI. https://platform.openai.com/docs/models.

[4] OpenAI. https://chatgpt.com/.

[5] R. Pass, E. Shi, and F. Tramèr. Formal abstractions for attested execution secure processors. *IACR*, 2016.

[6] M. Schlosser. Agency. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2019 edition, 2019.

[7] D.-A. Team. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. https://arxiv.org/abs/2501.12948, 2025.

[8] E. Team. Eigenlayer: The restaking collective. https://docs.eigenlayer.xyz/assets/files/EigenLayer_WhitePaper-88c47923ca0319870c611decd6e562ad.pdf.

[9] Virtuals. https://virtuals.io/.

[10] S. Walters, S. Gao, S. Nerd, F. Da, W. Williams, T.-C. Meng, A. Chow, H. Han, F. He, A. Zhang, M. Wu, T. Shen, M. Hu, and J. Yan. Eliza: A web3 friendly ai agent operating system, 2025.