Agents Have Eaten the World:

How Nillion Can Help Secure an Agentic World

nillion

Introduction

Agents will change AI and the world, reshaping industries and empowering individuals by allowing them to leverage information, expertise, and time they never had before. These agents could range from personal assistant agents managing your schedule and paying your bills, to enterprise-grade automated customer support agents helping customers 24/7. But it is not a foregone conclusion that this will play out to benefit everyone. We have been here before; software did indeed eat the world but also created an ecosystem in which individuals' data can be exploited by organisations that have all the leverage without benefiting them (and sometimes even harming them).

While there will be many challenges to rolling out ubiquitous AI agents in a way that empowers individuals, Privacy-Enhancing Technologies (PETs) offer some ways to mitigate the potential downsides by providing mechanisms that can keep users in control and agents in check. In particular, PETs make it possible to turn agents into cryptomatons — programs that keep their internal state and computations encrypted so that even the cloud infrastructure on which they operate cannot see it — that can exist and operate in private while still being just as useful and capable as they would be otherwise.

The goal of this paper is to formalise how we can minimise the negative impact caused by rogue or malicious agents and show how Nillion's technology can aid in realising this goal.

To this end, in this paper, we first introduce a new framework (Agent Specialisation and Agent Enforcement) to help reason about the security and privacy of agentic systems. Second, we show how we can leverage Nillion's SecretSDKs (most pertinently SecretVault) to design general agentic systems (multiple agents chained together) that satisfy our Agent Specialisation and Enforcement framework.

Framework: Introducing Agent Specialisation and Enforcement

In this section, we introduce the framework and concepts that will help us reason about the security and privacy of agentic systems. In particular, we introduce the concepts of Agent Specialisation and Enforcement.

At a high level, our motivation is to leverage Privacy-Enhancing Technologies (PETs) in designing an agent-centric ecosystem that extends the principle of least privilege beyond its traditional boundaries. While the principle can exist independently of PETs, we leverage Nillion's SecretSDKs to establish concrete privacy and security guarantees that span both individual agent processes and agent-to-agent interactions. Our framework formalises a key constraint: an agent's capabilities and access levels—along with those of its underlying infrastructure—should be precisely sufficient to accomplish its goals, nothing more. We argue that if this is realised, then our goal of minimising the negative impact of rogue, malicious or poorly designed agents is achieved.

Agent Specialisation

Intuitively, Agent Specialisation demands that individual agents should be modular and composable units that have a specific, well-defined, and narrowly scoped purpose across the data they have access to, their scope, and the other agents they interact with. We split Agent Specialisation into two concepts, Modularisation and Communication.

MODULARISATION

Agent Modularisation is described by the following two principles:

1. **Data access**: The agent's access to external data is restricted to only what is essential for an agent's specific tasks.

2. **Scope Policy**: The agent does not stretch beyond the defined scope set out in the User Agent Scope Policy which it receives as input from the user.

COMMUNICATION

Agent Communication is described by the following two principles:

1. **Communication Network**: Each agent can only communicate (output data to) a pre-defined set of other agents and systems.

2. **Communication Policy**: The agent follows the Communication Policy it receives from the end user when interacting with other agents and systems. This policy describes how it is allowed to interact. For example, it outlines the type of information it may share with different agents.

Agent Enforcement

Agent Specialisation is a great idea in theory, but is not enough on its own. We need a method to hold the agent accountable for following the policies it must abide by. To this end, we introduce the concept of a Policing Agent. This is an agent that is paired with every User Agent. In short, the Policing Agent communicates with the outside world (not the User Agent itself) and the User Agent must convince the Policing Agent to send the desired output to the intended recipient. We define Agent Enforcement with two concepts; independence and transparency.

1. **Independence**: The Policing Agent operates independently from the User Agent, ensuring unbiased enforcement of rules.

2. **Transparency**: All decisions are transparent, with clear logs and rationales for approvals or denials.

Next, we demonstrate how Nillion's SecretSDKs can be used to design agentic systems that satisfy Agent Specialisation and Enforcement.

Nillion's role in securing an agentic world

We now illustrate how Nillion's SecretSDKs can be used to realise Agent Specialisation and Agent Enforcement in an agentic system. For each concept, we provide an example system architecture that satisfies its principles.

Agent Specialisation

Agent Specialisation is split into two concepts; Modularisation and Communication. We discuss each in turn below.

MODULARISATION

The Modularisation concept requires that a User Agent only has access to a limited dimension of user data and that it takes, as input, the User Agent Scope Policy when deciding on actions on behalf of the user.

Fig 1. illustrates how both of these principles can be realised using Nillion's SecretVault and SecretTEE SDKs.

In Fig 1. User Agent A has permissions to access the user's health data and the Users Agent Scope Policy. Both of these are stored inside a Nillion SecretVault (under MPC), meaning no single node in the decentralised network can reconstruct the stored data. The whole system is run inside a SecretTEE to further increase the trustworthiness of the ecosystem.

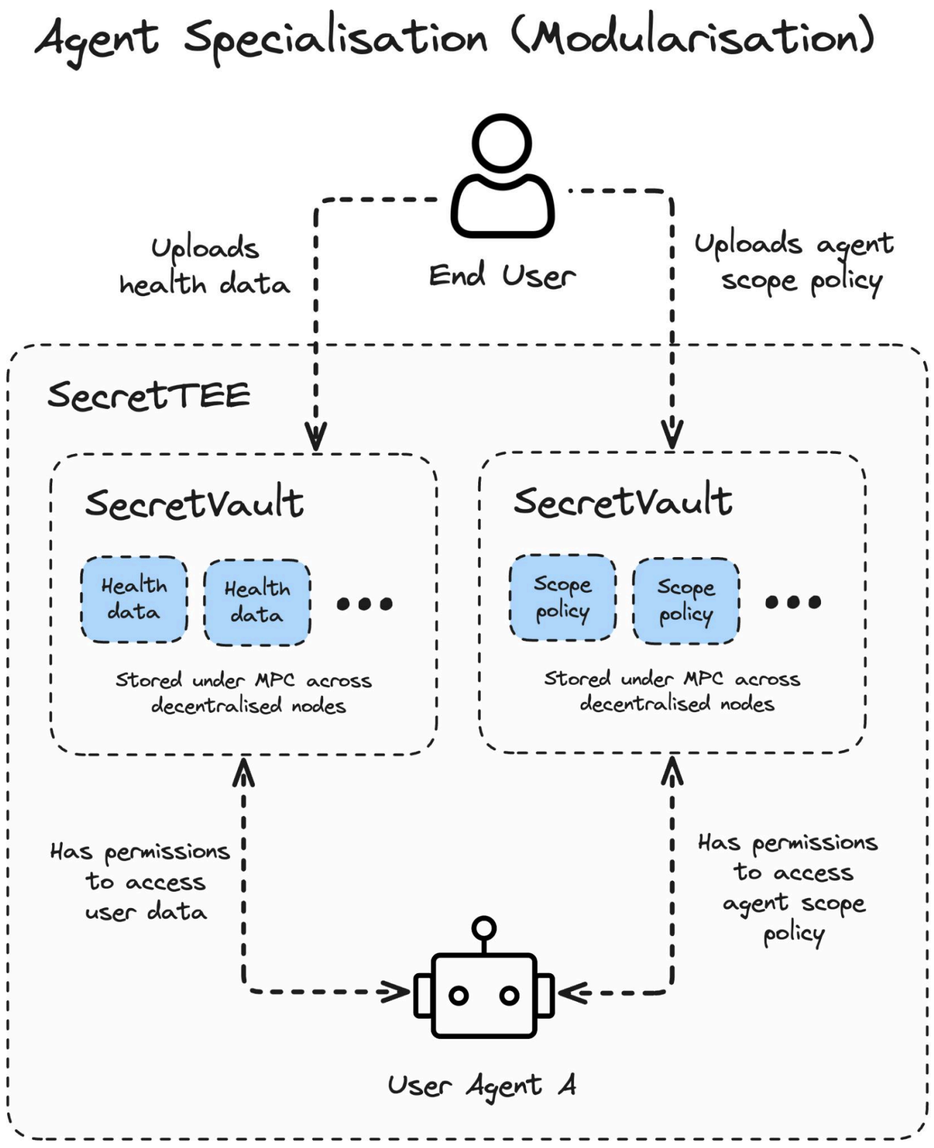

Fig 1: Agent Specialisation (Modularisation) achieved with Nillion

The Data Access principle is realised as the User Agent is the only entity (other than the end user) that has permissions to reconstruct and read the user's health data. The SecretVault guarantees no other agent, nor any node (or subset of colluding nodes) that form a SecretVault cluster can access this data. The agent has access to the Agent Scope policy, uploaded by the user, meaning Scope Policy principle is also realised.

COMMUNICATION

The Communication concept requires that a User Agent has a pre-defined set of other agents and systems it may interact with, and that the agent takes the User Agent Communication Policy as input to determine how to interact with other agents.

Fig 2. illustrates how both of these principles can be realised using Nillion's SecretVault and SecretTEE SDKs and the idea of a "secure postbox" between any two communicating agents.

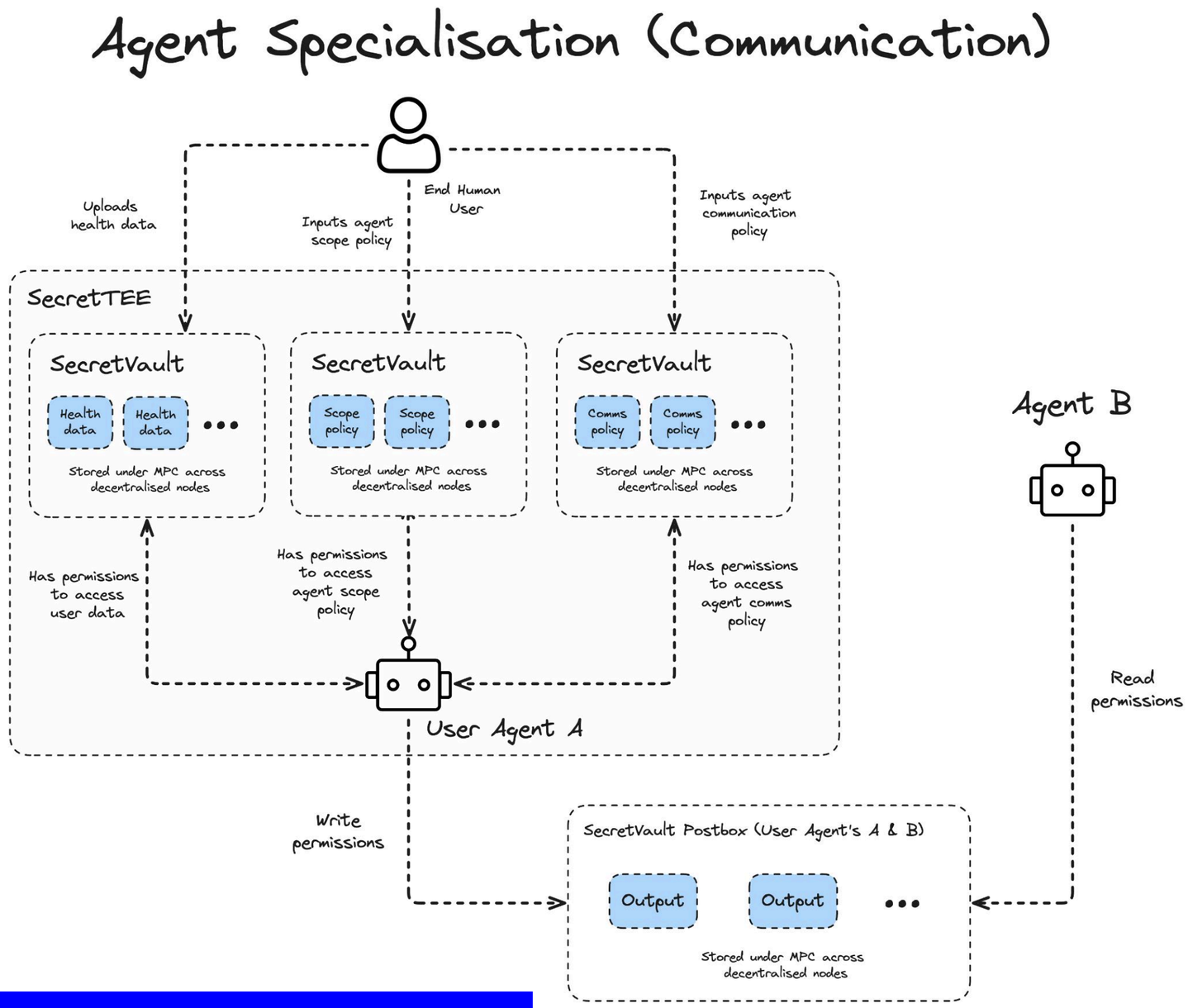

Fig 2: Agent Specialisation (Communication) achieved with Nillion

In Fig 2. User Agent A is communicating with Agent B. User Agent A has access to the user's health data, the User Agent Scope Policy and the User Agent Communication Policy, that have all been uploaded to distinct SecretVault by the end user. User Agent A communicates with Agent B via their shared SecretVault Postbox. In this example, User Agent A is able to send output to Agent B, however Agent B is not allowed to send output to User Agent A. This is enforced by the permissions on their shared postbox (SecretVault) - in particular User Agent A has write permissions, while Agent B only has read permissions (to read what User Agent A has outputted).

The Communication Network principle is realised as the set of agents with which User Agent A can communicate is well defined by the permissions set on the SecretVault Postboxes they share. The Communication Policy principle is satisfied because User Agent A has access to the user's communication policy - note only User Agent A has access to this due to the permission set on the SecretVault in which it is stored.

Agent Enforcement

Both flavours of Agent Specialisation provide strong guarantees that help minimise the negative impact a rogue or malicious agent can cause. The proposed solutions to realise these concepts, however, are only effective if the User Agent Scope and Communication policies are enforced. In this section we show how Nillion's Secret SDKs can be used to achieve Agent Enforcement by means of a Policing Agent.

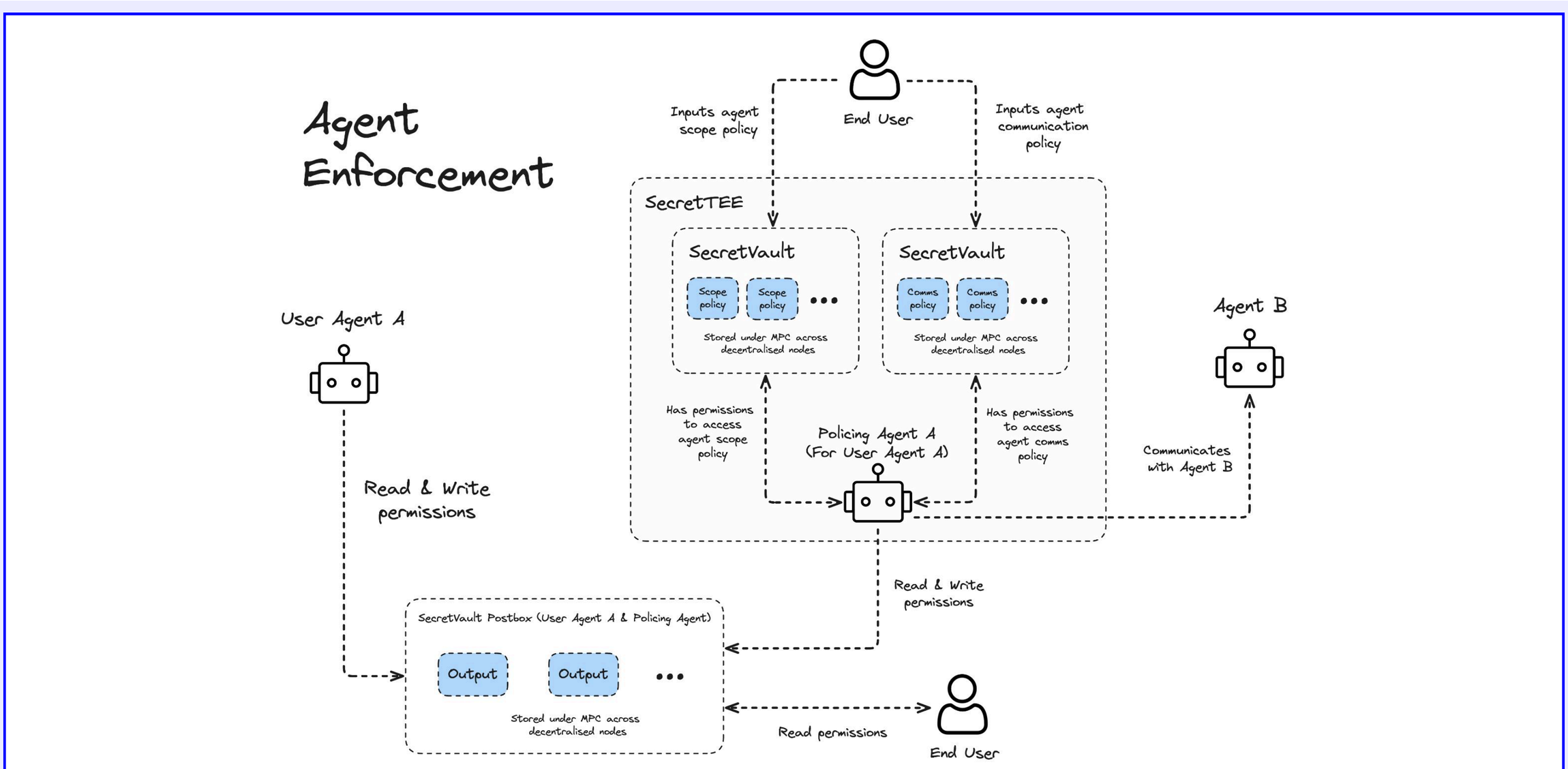

Fig 3: Agent Enforcement achieved with Nillion

Fig 3. shows how a Policing Agent can be used to realise Agent Enforcement. Here, User Agent A wants to communicate with Agent B, however it is not able to do so directly. Instead, it must communicate with its Policing Agent. In particular, User Agent A must convince its Policing Agent to relay the desired output to Agent B - in reality, this will require back and forth communication between the two agents. The Policing Agent has access to both the User Agent Scope and Communication Policies (stored in a SecretVault it has permissions to read from) but not the user's health data. Only when the Policing Agent has been convinced to share the output with Agent B, does it do so. While it is not illustrated in Fig 3., the Policing Agent would (as per previous situations) use a SecretVault Postbox to communicate with Agent B.

Both principles of Agent Enforcement are realised by this system design as follows. Independence is achieved by the Policing agent only having access to the two User Agent Policies (Scope and Communication), and nothing else. Transparency is achieved by the end user having read permissions on the SecretVault Postbox shared between User Agent A and the Policing Agent. This means that the end user has access to the decision making log (between User Agent A and its Policing Agent) and can thus understand any rational behind output that is allowed to be sent to Agent B.

Conclusion

In this paper, we have introduced a novel framework for reasoning about the security and privacy of agentic systems by defining Agent Specialisation and Agent Enforcement. We then showed how Nillion's SecretSDKs can be used to design agentic systems that satisfy Agent Specialisation and Enforcement.

It is clear that Agents will change AI and the world, and in doing so will require greater access to end user's data. It is vital that security and privacy are considered at every stage of this process, otherwise, the negative impacts of rogue or malicious agents could be unfathomable. By introducing our new framework and demonstrating how Nillion's SDKs can enforce it inside agentic systems we provide a clear path towards a safer agentic world.