NANDA+ANS Security Blueprint: A Federated Registry Architecture for Secure, Capability-Aware Agent Discovery

Abstract

AI Agent basic security foundation will need a robust and secure infrastructure for their discovery, identity management, and interaction. Existing mechanisms often fall short in addressing the unique security challenges posed by autonomous, capability-driven systems operating at scale. This paper presents a comprehensive security blueprint for a federated agent registry architecture, synergizing the NANDA framework's minimal Quilt root (Raskar et al., 2025) with the Agent Name Service (ANS) Public Key Infrastructure (PKI) hierarchy (Huang et al., 2025). We formalize a rigorous threat model by applying a combined MAESTRO and STRIDE methodology specifically to the proposed NANDA+ANS architecture. Our blueprint emphasizes cryptographic assurance for capabilities through verifiable credentials and Zero-Knowledge proofs, privacy-preserving discovery techniques, and a secure governance model incorporating economic incentives to deter malicious activity. We detail the reference architecture, security analysis, mitigation strategies, and a roadmap for building a scalable, secure, and trustworthy foundation for the future of agentic AI.

Keywords: Agent Security, Federated Registry, Agent Name Service (ANS), NANDA, Public Key Infrastructure (PKI), Verifiable Credentials, Zero-Knowledge Proofs, Privacy-Preserving Information Retrieval, Multi-Agent Systems, Threat Modeling, MAESTRO, STRIDE, Quilt, AgentFacts.

Executive Summary

The upcoming wide use of AI Agents provides unprecedented opportunities alongside significant security challenges. Current discovery mechanisms, like DNS and basic DID methods, are ill-equipped to handle the dynamic, security-sensitive nature of agentic

ecosystems, leading to risks such as agent impersonation, capability misrepresentation, rug pulls, squatting, and privacy breaches. This paper introduces the NANDA+ANS Security Blueprint, a federated registry architecture designed to address these critical security pain points.

By merging NANDA's minimal Quilt root with ANS's robust PKI hierarchy, we establish a tamper-evident, dual-trust foundation supporting both CA-signed and DID-based identities. A cornerstone of our architecture is a decoupled, two-hop lookup model: One hop is PrimaryFactsURL and 2nd hope is PrivateFactsURL. This separation significantly enhances security by minimizing the core registry's attack surface and enables dynamic, richly attested metadata to be managed at the edge, improving both privacy and resilience. Our solution emphasizes cryptographic assurance for agent capabilities using verifiable credentials and Zero-Knowledge proofs, ensuring that consumers can verify an agent's claimed skills before invocation. We incorporate privacy-preserving discovery techniques, allowing clients to look up capabilities without leaking sensitive intent. Furthermore, we define a secure governance model with economic incentives, such as staking and registrar fees, to deter spam, capability-squatting, and the deployment of malicious adapter code. This blueprint provides a foundational layer for secure agent discovery, verifiable trust, and interoperable communication, paving the way for a resilient and trustworthy Internet of AI Agents.

The MVP project is located at: https://github.com/appsec2008/nanda_ans

1. Problem Statement: Agent Security Gaps

The current infrastructure for service and identity discovery, largely inherited from the human-centric web, exhibits critical security gaps when applied to the emerging landscape of autonomous AI agents. These gaps pose significant risks to the reliability, trustworthiness, and safety of agentic AI systems.

- **Phishing and Agent Impersonation:** Without strong, verifiable identity mechanisms, malicious agents can easily masquerade as legitimate entities, deceiving users or other agents to steal credentials, exfiltrate data, or execute unauthorized actions. Traditional domain ownership verification (as in DNS) is insufficient for attesting to an agent's identity or authenticity.
- **Endpoint Spoofing and Manipulation:** DNS vulnerabilities (e.g., cache poisoning, hijacking) and insecure DID resolution pathways can redirect agent communication to malicious endpoints, enabling man-in-the-middle attacks, data interception, and command injection.

- **Capability Squatting and Misrepresentation:** Malicious actors can register agent names or advertise capabilities deceptively similar to those of reputable agents, leading to user confusion and the invocation of untrustworthy services. There's often no reliable way to verify if an agent *truly* possesses the capabilities it claims.
- **Revocation Lag and Persistence of Compromise:** Delays in propagating certificate revocation information (e.g., via CRLs in PKI or updates in DID systems) mean that compromised agents can continue to operate and cause harm long after their credentials should have been invalidated. This is particularly problematic for fast-moving, autonomous agents.
- **Privacy Leaks in Discovery:** Current lookup mechanisms often reveal the identity of the querying agent and the nature of its query (e.g., desired capability). This exposes sensitive information that can be used for profiling, targeted attacks, or competitive intelligence, violating privacy principles.
- **Lack of Granular Trust:** Existing systems typically offer coarse-grained trust models (e.g., a domain is trusted or not). Agentic AI requires more nuanced trust, including verifiable attestations of specific skills, compliance certifications, and operational history.
- **Insecure Supply Chain for Agent Components:** Agents often rely on third-party models, tools, and data. The lack of verifiable attestations for these components creates vulnerabilities for supply-chain attacks, where malicious code or data can be injected.

Addressing these security gaps is paramount for fostering user trust and enabling the safe deployment of agentic AI technologies at scale. The NANDA+ANS Security Blueprint aims to provide a foundational infrastructure that directly mitigates these risks.

2. Threat Landscape & Taxonomy

Understanding the threat landscape for agentic AI requires a structured approach that considers the unique characteristics of these systems. We adopt a combination of the **MAESTRO** framework for its layered analysis of agentic architectures and the **STRIDE** methodology for categorizing specific threat types.

The MAESTRO framework (Huang, 2025) deconstructs agentic AI systems into seven layers: Foundation Models, Data Operations, Agent Frameworks, Deployment & Infrastructure, Evaluation & Observability, Security & Compliance, and the Agent Ecosystem. This layered perspective is crucial because vulnerabilities in one layer can cascade and impact others.

STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) provides a well-established taxonomy for classifying security threats.

By combining MAESTRO and STRIDE, we can systematically identify and categorize potential vulnerabilities within the NANDA+ANS architecture. This approach allows us to move beyond generic security concerns and focus on threats specifically relevant to a federated agent registry.

Common Attacker Personas in the Agent Registry Context:

- **Malicious Agent Developer:** Intentionally creates and registers agents designed to deceive, defraud, or harm other agents or users.
- **Network Eavesdropper/Man-in-the-Middle:** Attempts to intercept or manipulate communication between agents and the registry, or between agents themselves.
- **Compromised Infrastructure Operator:** An attacker who gains control of registry components (e.g., a federated shard, a CA/RA node) to inject false information or deny service.
- **State-Level Actor:** Possesses significant resources to conduct sophisticated attacks, including large-scale Sybil attacks or attempts to undermine the root of trust.
- **Insider Threat:** A privileged user within an organization operating a registry node or CA/RA who abuses their access.

A detailed application of MAESTRO and STRIDE to the specific components of the NANDA+ANS reference architecture, mapping identified threats to their respective layers and categories, will be presented in Section 7: Security Analysis & Mitigation Matrix. This foundational understanding of the threat landscape informs the design principles and security mechanisms of our proposed blueprint.

3. Security Centric Design Principles

The NANDA+ANS Security Blueprint is architected upon a set of core design principles, prioritizing security, verifiability, and privacy:

- **Least-Trust Registry Core:** The central registry components (NANDA Quilt root and core ANS pointers) are designed to be minimal, storing only essential, immutable, or slowly changing static metadata (e.g., agent cryptographic identifiers, pointers to AgentFacts, CA roots). This minimizes the attack surface of the core infrastructure and reduces the impact of a compromise at this level. Dynamic and sensitive data are pushed to the edges (AgentFacts). The NANDA registry resolver does not return raw metadata or service endpoints. Instead, it returns a cryptographically signed, cacheable object called AgentAddr, which serves as a lightweight address record for the agent. This object encapsulates the agent's identifier, TTL (time-to-live), and pointers to metadata and optional routing infrastructure.

    -
    - *agent_id (Machine-readable ID)): A globally unique decentralized identifier compliant with [i.e. W3C DID Core] specifications.*
    - *agent_name (URN): A human-readable alias encoded as a URN (e.g., urn:agent:salesforce:starbucks).*
    - *facts_url (FactsURL): A reference to the AgentFacts hosted at the agent's domain (e.g., https://salesforce.com/starbucks/.agent-facts).*
    - *private_facts_url (PrivateFactsURL): An optional, privacy-enhanced reference to the AgentFacts hosted on a third-party or decentralized service (e.g., https://agentfacts.nanda.ai/...).*
    - *adaptive_router_url (optional): An optional endpoint for dynamic routing services (e.g., https://router.example.com/dispatch) that handle load balancing, failover, or geo-aware dispatching.*
    - *ttl (Time-To-Live): The maximum cache duration before the client must re-resolve the record.*
    - *signature: A cryptographic signature from the registry resolver that binds the contents of the AgentAddr.*
    - *The AgentAddr object is signed and cacheable, allowing clients to redistribute it and perform verification without repeated lookups. Its role is similar to a DNS record, but extended to include verifiable metadata pointers and flexible routing instructions.*

- **Verifiable Metadata (AgentFacts):** All detailed agent information, including capabilities, endpoints, policies, and attestations, is encapsulated in cryptographically signed and verifiable structures (AgentFacts, leveraging W3C Verifiable Credentials). This ensures data integrity and authenticity, allowing consumers to trust the information presented.

The AgentFacts functions as an intermediary data structure between a stable registry record and an agent's operational configuration. It offers the following advantages:

- **Dynamic Updatability:** AgentFacts can be updated independently of the registry, supporting frequent changes to capabilities, telemetry, or endpoints.
- **Credentialed Assertions:** All claims (e.g., skills, certifications) are signed by issuers, ensuring tamper-resistance and reputation integrity.
- **Deployment Flexibility:** Facts can be hosted at the agent's domain (**PrimaryFactsURL -** URL1) or on neutral infrastructure (**PrivateFactsURL -** URL2) to meet privacy or reliability requirements.
- **Multi-Endpoint Support:** Enables load balancing, routing to specialized instances, and failover mechanisms.

AgentFacts serve as the functional equivalent of a decentralized service manifest. Each facts includes:

- Versioned metadata and labels (e.g., "Translation Assistant v1.2.1").
- Lists of capabilities (e.g., "streaming", "authentication").
- Skills and modalities (e.g., "text → speech" translation).
- Telemetry endpoints for observability (e.g., via OpenTelemetry).
- Signed evaluations and certifications.

    - Endpoint URIs and resolution TTLs.

All AgentFacts must be signed by an issuer authorized under the NANDA trust framework or domain-specific federated authority.

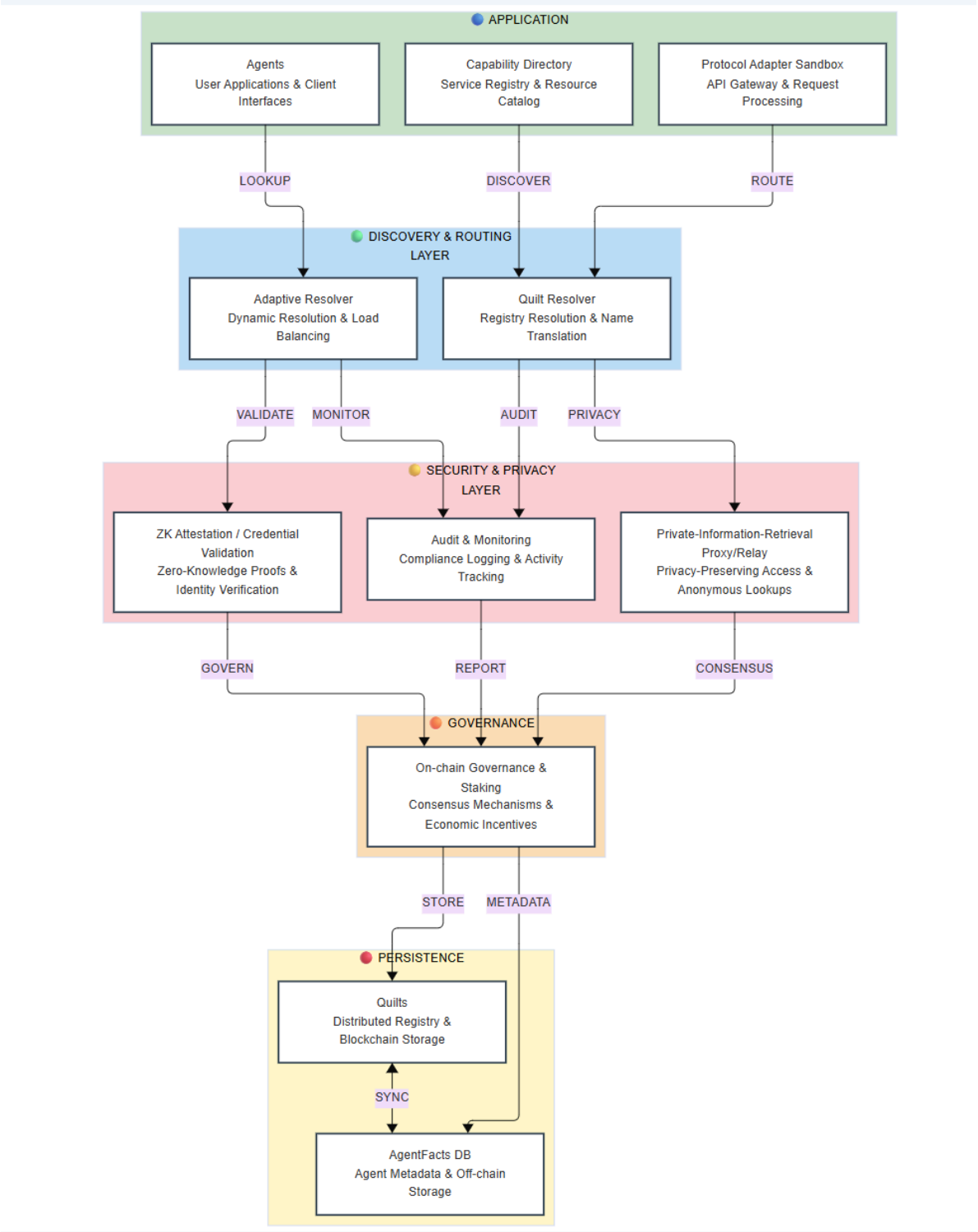
NANDA proposes to treat AgentFacts as verifiable claims requiring W3C Verifiable Credential v2 cryptographic attestation, with issuers including enterprises, consortiums, or federated certification authorities operating within domain-specific trust zones that can cross-sign each other, and each claim linked to credentialing paths anchored in issuer DIDs with revocation status via VC-Status-List. Please note this requires W3C Verifiable Credential infrastructure, DID resolution systems, cross-signing protocols between trust zones, and VC-Status-List revocation mechanisms. In future, we may expand trust zone federation models and enhanced revocation mechanisms while maintaining the core principle of cryptographically attested, auditable agent metadata for decentralized, extensible, and trustworthy discovery infrastructure.

- **Capability-First Naming for Enhanced Security:** The ANS naming structure explicitly includes `agentCapability`. This, combined with cryptographic attestations for these capabilities, allows for more precise and secure discovery. It helps prevent capability misrepresentation and enables clients to verify that an agent is authorized and competent to perform a claimed function *before* interaction.
- **Privacy-by-Default Queries:** The architecture incorporates mechanisms like Privacy-Preserving Information Retrieval (PIR-light techniques) and carefully designed resolution paths (e.g., PrivateFactsURL) to allow agents to query the registry for capabilities or other agents without revealing their specific intent or identity to the registry operator or passive observers.
- **Auditable and Transparent Governance:** The governance model for the federated registry, including processes for registration, revocation, dispute resolution, and policy updates, is designed to be transparent and auditable. This builds trust in the system's impartiality and operational integrity. Economic incentives (staking, fees) are aligned with security objectives.
- **Dual-Trust Anchors (CA-signed & DID-based):** The system supports both traditional PKI (CA-signed certificates via ANS) and decentralized identity (DIDs managed through NANDA). This hybrid approach offers flexibility, accommodates diverse trust preferences, and provides resilience by not relying on a single identity paradigm.
- **Secure by Default Configuration:** Default configurations and reference implementations (e.g., the Rust SDK) prioritize secure settings, such as mandatory certificate validation, secure communication protocols (mTLS), and robust input validation.
- **Defense in Depth:** Security is layered throughout the architecture, from the minimal root to the dynamic resolution layer. No single point of failure or compromise should lead to a catastrophic system-wide breach.

These principles guide the design of each component and interaction within the NANDA+ANS ecosystem, aiming to create a resilient and trustworthy infrastructure for agentic AI.

4. Reference Architecture with Security

The NANDA+ANS reference architecture is a federated system designed to provide secure and scalable agent discovery. It integrates NANDA's minimal Quilt root and AgentFacts concept with ANS's PKI-based identity and structured naming.



4.1. Core Architectural Components & Security Roles:

- **NANDA Hybrid Root (Minimal Quilt Root + ANS PKI Linkage):**

    - **NANDA Quilt Root:** Provides the foundational, immutable layer for the federated registry. It anchors trust for DID-based identities and pointers to AgentFacts Stores. Its minimality reduces its attack surface.
    - **ANS PKI Linkage:** It securely links to the ANS Certificate Authority (CA) hierarchy, establishing the root of trust for CA-signed agent identities(Huang, Narajala, Yeoh, et al., 2025). This linkage itself is cryptographically secured.
    - *Security Role:* Establishes the ultimate source of truth and trust for the entire federated system, supporting dual identity models.
- **AgentFacts Store (Decentralized Metadata Store):**
    - Stores AgentFacts: These are JSON-LD documents containing detailed, versioned, and cryptographically signed agent metadata (capabilities, endpoints, attestations, telemetry configurations, `protocolExtensions`).
    - Utilizes content-addressable storage (e.g., IPFS) or agent-hosted `.well-known` paths.
    - *Security Role:* Decouples dynamic metadata from the core registry, allowing frequent updates without registry writes. Verifiability is ensured by W3C Verifiable Credentials and issuer signatures.
- **ANS Certificate Authority (CA) / Registration Authority (RA) Hierarchy:**
    - **CA:** Issues X.509 certificates to agents, binding their public keys to their verified identities (ANS names, organizational affiliations).
    - **RA:** Validates agent registration and renewal requests according to defined policies before instructing the CA to issue certificates. Manages the lifecycle of CA-signed identities.
    - *Security Role:* Provides strong, centrally auditable identity verification for agents opting into the PKI trust model. Manages certificate revocation (CRL/OCSP).
- **Protocol Adapter Layer (Interoperability & Security Gateway):**
    - Mediates between the NANDA+ANS registry and specific agent communication protocols (A2A, MCP, ACP).
    - Parses and validates protocol-specific metadata from AgentFacts' `protocolExtensions` (e.g., A2A Agent Card, MCP Tool Description).
    - Can enforce protocol-specific security policies during discovery or before interaction.
    - *Security Role:* Ensures that protocol-specific data is handled securely and that interoperability does not introduce new vulnerabilities.
- **Signed AdaptiveResolver (Dynamic & Secure Endpoint Selection):**
    - An optional component referenced in AgentFacts or AgentAddr for dynamic routing.
    - Inspects request context (location, load, client capabilities) and securely returns an optimal, potentially ephemeral, endpoint.
    - Responses (endpoint URIs, session tokens) are cryptographically signed by the AdaptiveResolver (itself a registered and trusted agent).

    - *Security Role:* Prevents endpoint spoofing in dynamic routing scenarios, supports DDoS mitigation (shuffle-sharding), and enables capability-specific routing.
- **Privacy-Preserving Information Retrieval (PIR) Relay:**
    - An optional layer that clients can use to query AgentFacts Stores or other registry components without revealing their query content to the server.
    - Implements PIR-light schemes or interfaces with more advanced differential privacy mechanisms.
    - *Security Role:* Protects client query privacy, preventing profiling or leakage of sensitive intent.
- **Zero-Knowledge (ZK) Attestation Service:**
    - Allows agents to prove possession of certain attributes or compliance with policies (e.g., "I am certified for HIPAA and my model version is > X") without revealing the underlying sensitive data.
    - ZK proofs are included in AgentFacts or presented during interaction.
    - *Security Role:* Enhances privacy while maintaining verifiability of claims, crucial for sensitive capabilities or regulated environments.

4.2. Key Secure Workflows

- **Secure Agent Registration (PKI-based via ANS):**

    1. **Agent → RA:** Registration request with metadata, CSR, and proof of identity/ownership.
    2. **RA:** Validates request against policies.
    3. **RA → CA:** Validated CSR.
    4. **CA → RA:** Issues X.509 Certificate.
    5. **RA → Agent Registry (ANS component) & AgentFacts Store:** Stores AgentAddr (containing certificate hash/pointer) and AgentFacts (containing full certificate and other metadata). The AgentAddr is anchored in the NANDA Quilt root. *All communications are secured, and requests/responses are signed.*

- **Secure Agent Registration (DID-based via NANDA):**

    1. **Agent:** Generates DID and associated key pairs.
    2. **Agent → AgentFacts Store:** Publishes signed AgentFacts (containing DID, public keys, capabilities) to a self-hosted location or decentralized store.
    3. **Agent → NANDA Registry (Quilt Root Registrar):** Request to anchor AgentAddr (containing DID and pointer to AgentFacts) in the Quilt root, potentially requiring staking or other anti-spam measures. *AgentFacts are self-signed or attested by relevant DID issuers.*

- **Secure Capability Discovery & Resolution:**

    1. **Client Agent → NANDA+ANS Resolver (potentially via PIR Relay):** Query for agent with `agentName` or `agentCapability`.
    2. **Resolver → NANDA Quilt Root:** Resolves `agentName` to AgentAddr (signed by the registry shard).
    3. **NANDA Quilt Root → Resolver:** Signed AgentAddr (contains DID/PKI info, `facts_url`, `private_facts_url`, `adaptive_router_url`, TTL).
    4. **Resolver → Client Agent:** Signed AgentAddr.
    5. **Client Agent:** Verifies AgentAddr signature.
    6. **(Optional) Client Agent →AgentFacts Store (via `facts_url` or `private_facts_url`):** Fetch AgentFacts (JSON-LD + VC).
    7. **AgentFacts Store → Client Agent:** Signed AgentFacts.
    8. **Client Agent:** Verifies AgentFacts signature and any embedded VCs (including ZK attestations). Validates against its trust policies (e.g., checks issuer DIDs, CA trust chain, revocation status via VC-Status-List or OCSP).
    9. **(If AdaptiveRouter is used) Client Agent → AdaptiveRouter (via `adaptive_router_url`):** Request for endpoint, providing context.
    10. **AdaptiveRouter → Client Agent:** Signed ephemeral endpoint or session token.
    11. **Client Agent:** Verifies AdaptiveRouter response.
    12. **Client Agent → Target Agent Endpoint:** Initiates secure communication (e.g., mTLS using verified certificates/DIDs). *All steps ensure cryptographic verification and honor TTLs.*

This reference architecture provides a layered security model, where trust is established and maintained through cryptographic mechanisms at each stage of agent registration, discovery, and interaction.

5. Security Analysis & Mitigation Matrix

This section applies the MAESTRO framework and STRIDE methodology to analyze potential security vulnerabilities within the proposed NANDA+ANS reference architecture and outlines corresponding mitigation strategies.

MAESTRO Layer	STRIDE Category	Threat within NANDA+ANS Architecture	Mitigation Strategies in NANDA+ANS	NANDA+ANS Component(s) Implicated
Foundation Models (Relevant if agents *are* or *use* foundation models, and registry stores related attestations)	Tampering	**Capability Misrepresentation (Forged Attestation):** An agent registers with a ZK proof or VC attesting to a capability (e.g., based on a specific model version) that it doesn't actually possess or whose underlying model has been tampered with.	Require ZK proofs/VCs to be issued by trusted, auditable attestors. Link attestations to specific model hashes. Implement continuous monitoring and challenge-response for capability verification post-discovery.	AgentFacts Store, ZK Attestation Service
Data Operations (Registry data, AgentFacts)	Information Disclosure	**Leakage of AgentFacts via Unsecured Hosting:** AgentFacts hosted on insecure agent-controlled servers (`facts_url`) are exposed or tampered with.	Encourage use of `private_facts_url` with IPFS/secure relays. Client-side verification of AgentFacts signatures (mandatory). Implement strict validation for self-hosted `.well-known` paths during registration.	AgentFacts Store, Client Resolver
	Tampering	**Registry Shard Compromise**	NANDA's minimal root	NANDA Quilt Root & Shards

MAESTRO Layer	STRIDE Category	Threat within NANDA+ANS Architecture	Mitigation Strategies in NANDA+ANS	NANDA+ANS Component(s) Implicated
		(Quilt): A malicious operator of a NANDA Quilt registry shard injects false AgentAddr records or censors legitimate ones.	provides an anchor for verifying shard integrity. Cross-shard consistency checks. Staking/slashing for shard operators. Client-side multi-shard queries for resilience.	
Agent Frameworks (Protocol Adapters)	Elevation of Privilege	**Malicious Protocol Adapter:** A compromised or maliciously designed protocol adapter manipulates data during translation or exceeds its intended permissions.	Sandbox adapter execution. Define strict, least-privilege interfaces for adapters. Code signing and verification for adapter modules. Community vetting and security audits for common adapters.	Protocol Adapter Layer
	Tampering	**Exploitation of `protocolExtension` Schema:** Poorly defined or validated `protocolExt`	Rigorous JSON-LD schema validation for all `protocolExtensions`. Input sanitization. Define clear	Protocol Adapter Layer, **AgentFacts** Store

MAESTRO Layer	STRIDE Category	Threat within NANDA+ANS Architecture	Mitigation Strategies in NANDA+ANS	NANDA+ANS Component(s) Implicated
		`ension` schemas allow for injection attacks or data corruption.	guidelines and best practices for `protocolExtension` development.	
Deployment & Infrastructure (Registry nodes, CA/RA, Resolvers)	Denial of Service	**DDoS on Core Registry Components:** NANDA Quilt Root, ANS CA/RA, or primary resolvers are targeted, disrupting global discovery.	Distributed architecture for all critical components. Anycast addressing. Rate limiting, traffic scrubbing. TTL-based caching at various layers to reduce load on core components. Staking for access to resource-intensive operations.	All Core Infrastructure
	Elevation of Privilege	**Compromise of RA/CA Operator Node:** An attacker gains control of an RA or sub-CA node, enabling issuance of fraudulent certificates or agent registrations.	Strict operational security for RA/CA nodes (HSMs, MFA, network segmentation). Segregation of duties. Regular security audits. Certificate Transparency logs for issued	ANS CA/RA Hierarchy

MAESTRO Layer	**STRIDE Category**	**Threat within NANDA+ANS Architecture**	**Mitigation Strategies in NANDA+ANS**	**NANDA+ANS Component(s) Implicated**
			ANS certificates.	
Evaluation & Observability (Registry logs, agent interaction telemetry if mediated)	Information Disclosure	**Query Pattern Analysis on Resolvers:** Aggregated query data at resolvers reveals trends or sensitive interests, even with PIR for individual queries.	Differential privacy techniques applied to aggregated statistics. Strict data retention and access control policies for resolver logs.	NANDA+ANS Resolvers, PIR Relay
	Repudiation	**Disputed AgentFact Attestation:** An agent denies the validity of a credential found in its AgentFacts, or an issuer denies having issued it.	All AgentFacts and VCs are digitally signed. Secure timestamping. Immutable ledger properties of the NANDA Quilt for registration events can provide evidence.	**AgentFacts** Store, ANS CA
Security & Compliance (Trust framework, policies)	Spoofing	**Weak Identity Verification during Registration:** Malicious actor registers an agent with a spoofed identity (DID or	Rigorous RA validation policies (domain control validation, organizational checks for CA-backed; DID ownership	ANS RA, NANDA Registrar

MAESTRO Layer	STRIDE Category	Threat within NANDA+ANS Architecture	Mitigation Strategies in NANDA+ANS	NANDA+ANS Component(s) Implicated
		CA-backed) due to lax RA validation.	proofs). Multi-factor verification for high-value registrations.	
	Tampering	**Manipulation of Governance Rules:** An attacker influences governance mechanisms (e.g., voting on policy changes) to weaken security.	Transparent, auditable governance processes. Resilient consensus mechanisms. Staking-weighted voting or similar mechanisms to prevent undue influence.	Governance Body/DAO
Agent Ecosystem	Spoofing, Tampering	**Capability Squatting & Deceptive Naming:** Malicious agents register names or advertise capabilities highly similar to popular, trusted agents.	Stricter validation for names resembling known entities. Dispute resolution process for naming conflicts. Capability bonds or attestations required for high-value capabilities. User education and UI cues.	ANS RA, NANDA Registrar

MAESTRO Layer	STRIDE Category	Threat within NANDA+ANS Architecture	Mitigation Strategies in NANDA+ANS	NANDA+ANS Component(s) Implicated
	Denial of Service (Economic)	**Registry Spam/Resource Exhaustion:** Malicious creation of vast numbers of low-value agent registrations to consume registry resources.	Registration fees, staking requirements (capability bonds). Rate limiting on registration APIs. Periodic cleanup of inactive/unfunded registrations.	NANDA Registrar, ANS RA

This matrix highlights that security in the NANDA+ANS architecture is not a single component's responsibility but an emergent property of the interaction between its layers and the cryptographic mechanisms embedded throughout.

6. Governance, Revocation, & Economic Incentives

A robust governance model, effective revocation mechanisms, and well-designed economic incentives are crucial for the long-term security, stability, and trustworthiness of the NANDA+ANS federated registry.

6.1. Registrar Workflows & Secure Registration:

- **Policy-Driven Validation:** Registration Authorities (RAs) for ANS and registrars for NANDA Quilt entries operate under clearly defined, publicly auditable policies. These policies govern identity verification (e.g., domain control validation for CA-signed identities, cryptographic proof of DID ownership), capability attestation requirements, and compliance checks.
- **Automated & Manual Checks:** Workflows incorporate automated checks for schema compliance, signature validity, and basic policy adherence. High-assurance registrations or those involving sensitive capabilities may trigger manual review processes.

- **Cryptographic Proof of Submission:** All registration and update requests are digitally signed by the requesting agent or its authorized representative, ensuring authenticity and non-repudiation.
- **Audit Trails:** All registration, update, and revocation events are logged in a tamper-evident manner (e.g., anchored to the NANDA Quilt or a permissioned blockchain accessible to auditors).

6.2. Staking, Slashing, and Capability Bonds (Economic Deterrents):

- **Registration Staking:** To deter Sybil attacks and spam registrations, agents may be required to stake a nominal amount of cryptocurrency or a digital asset. This stake can be slashed (forfeited) if the agent is found to be malicious or to have violated registry policies.
- **Capability Bonds:** For agents advertising high-impact or critical capabilities (e.g., financial transactions, healthcare advice), a larger "capability bond" might be required. This bond acts as a form of insurance or collateral, which can be used to compensate victims or fund remediation efforts in case of proven misbehavior or gross negligence directly attributable to a false capability claim.
- **Registrar Fees:** Nominal, cost-recovery fees for registration and renewal can further disincentivize frivolous registrations and contribute to the operational sustainability of the registry infrastructure.
- **Slashing Mechanics:** Clear, transparent, and community-agreed-upon rules govern the conditions under which stakes or bonds are slashed. This process must be subject to the dispute resolution mechanism.

6.3. Arbitration and Dispute Resolution (Decentralized Adjudication):

- **Decentralized Autonomous Organization (DAO) or Adjudication Body:** A dedicated DAO or a panel of elected/appointed adjudicators is responsible for resolving disputes related to:
    - Naming conflicts (capability squatting).
    - Validity of capability attestations.
    - Allegations of malicious agent behavior linked to registry information.
    - Appeals against stake/bond slashing decisions.
- **Evidence-Based Process:** Dispute resolution relies on verifiable evidence, including signed AgentFacts, VCs, audit logs, and potentially off-chain data.
- **Service Level Agreements (SLAs):** Defined SLAs for acknowledging, investigating, and resolving disputes to ensure timely action.
- **Enforcement:** Decisions by the arbitration body can trigger actions such as forced revocation of an agent's registration, slashing of stakes, or public flagging of a problematic agent.

6.4. Lifecycle Management: Secure Revocation (Registration → Renewal → Revocation):

- **Proactive Renewal:** Agents are required to periodically renew their registrations and associated credentials (X.509 certificates, VCs). Failure to renew leads to automatic expiry and potential de-listing.
- **Rapid Revocation Mechanisms:**
    - **ANS (PKI):** Standard CRLs and OCSP are used for X.509 certificate revocation. OCSP stapling is encouraged for performance.
    - **NANDA (DID/VCs):** VC-Status-List (e.g., VC-Status-List 2021) or similar mechanisms allow for efficient, scalable revocation of Verifiable Credentials associated with AgentFacts. Revocation status can be checked quickly by clients.
    - **Sub-second Revocation Goal:** The architecture aims for near real-time propagation of revocation status, critical for mitigating damage from compromised agents.
- **Graceful Deregistration vs. Forced Revocation:** Agents can voluntarily deregister. Forced revocation is triggered by governance decisions (e.g., proven malicious activity, severe policy violation) or automated checks (e.g., compromised key detection).
- **Revocation Transparency:** Revocation events (especially forced ones) are made publicly visible (e.g., through a revocation transparency log) to alert the ecosystem.

This integrated model of governance, economic incentives, and robust lifecycle management aims to create a self-regulating and resilient ecosystem that actively discourages malicious behavior and maintains a high degree of trust.

7. Implementation & Best-Practice Guide

Bringing the NANDA+ANS Security Blueprint to life requires careful implementation choices and adherence to security best practices throughout the development lifecycle.

7.1. Rust SDK Security Hardening:

- **Memory Safety:** Leverage Rust's inherent memory safety features (ownership, borrowing, lifetimes) to prevent common vulnerabilities like buffer overflows, use-after-free, and data races.
- **Dependency Management:** Use `cargo-audit` and `cargo-geiger` to vet dependencies for known vulnerabilities and unsafe code usage. Pin dependency versions and regularly update them.
- **Secure Cryptographic Primitives:** Utilize well-vetted, standard cryptographic libraries (e.g., `ring`, `rust-openssl`, `dalek-cryptography`) for all signing,

verification, encryption, and hashing operations. Avoid implementing custom cryptography.
- **Input Validation & Sanitization:** Rigorously validate and sanitize all inputs received from external sources (network, files, user input) to prevent injection attacks (SQLi, XSS, command injection if applicable).
- **Error Handling:** Implement robust error handling that avoids leaking sensitive information in error messages and fails securely.
- **Static Analysis & Fuzzing:** Integrate static analysis tools (e.g., Clippy, Rust Analyzer checks) and fuzz testing (e.g., `cargo-fuzz`) into the CI/CD pipeline to proactively identify potential bugs and security flaws.
- **Secure Defaults:** Ensure that the SDK's default configurations prioritize security (e.g., default to mTLS, enforce certificate validation).

7.2. Secure Adapter Contract & Development:

- **Well-Defined Interfaces:** Protocol adapters must interact with the core registry and AgentFacts through strictly defined, minimal interfaces that enforce least privilege.
- **Input/Output Validation:** Adapters must rigorously validate all data exchanged with the registry and with protocol-specific agent messages against their respective schemas (JSON-LD, protocol-specific schemas).
- **Sandboxing/Isolation (Optional but Recommended):** Consider running protocol adapters in sandboxed environments or isolated processes to limit the blast radius if an adapter is compromised.
- **Code Signing & Verification:** Adapter modules should be digitally signed by their developers/publishers. The registry or client systems can verify these signatures before loading or trusting an adapter.
- **Resource Limits:** Impose resource limits (CPU, memory, network bandwidth) on adapter execution to prevent denial-of-service conditions.
- **Security Audits for Common Adapters:** Encourage or mandate security audits for widely used, open-source protocol adapters.

7.3. Test-Vector Suite for Security Verification:

- **Comprehensive Coverage:** Develop a comprehensive suite of test vectors covering:
    - Valid and invalid cryptographic signatures (AgentAddr, AgentFacts, VCs).
    - Malformed or malicious JSON-LD and `protocolExtension` data.
    - Edge cases in registration, resolution, and revocation workflows.
    - Known attack patterns (e.g., replay attacks, spoofing attempts).
    - PIR query correctness and privacy guarantees.
    - ZK proof generation and verification.
- **Negative Testing:** Emphasize negative test cases to ensure the system handles errors and malicious inputs gracefully and securely.

- **Integration Testing:** Test vectors should cover end-to-end scenarios involving interactions between multiple components (Client, Resolver, NANDA Root, ANS CA/RA, **AgentFacts** Store, Adapters).
- **Automated Testing:** Integrate the test-vector suite into the CI/CD pipeline for continuous security regression testing.

7.4. Secure Integration with A2A, MCP, ACP:

- **Leverage Native Protocol Security:** Where A2A, MCP, or ACP define their own security mechanisms (e.g., OAuth tokens, signed messages within the protocol), the NANDA+ANS discovery layer should complement, not replace, these. The registry provides the initial trust anchor and verifiable identity.
- **Consistent Identity Propagation:** Ensure that the identities verified through NANDA+ANS (DIDs, PKI distinguished names) can be securely and consistently mapped to or used within the identity models of A2A, MCP, and ACP.
- **Secure Handshake Post-Discovery:** After an agent is discovered via NANDA+ANS, the subsequent communication establishment using A2A, MCP, or ACP must itself be secured (e.g., initiating an mTLS handshake using the verified certificates).
- **Capability Attestation Mapping:** Map the high-level capabilities discovered via ANS and attested in AgentFacts to the specific tool/service definitions within MCP, action definitions in ACP, or capability descriptors in A2A. Verifiable Credentials from AgentFacts can provide strong assurance for these mappings.

Adherence to these implementation guidelines and best practices is essential for realizing the security promises of the NANDA+ANS architecture.

8. Benchmarks & Evaluation (Security-Focused)

Empirical evaluation is critical to validate the security claims and performance characteristics of the NANDA+ANS architecture, especially under adversarial conditions.

8.1. Performance of Privacy-Preserving Lookups (TTL Cache vs. PIR):

- **Objective:** Quantify the latency overhead and server-side computational cost introduced by PIR-light mechanisms compared to direct lookups (potentially with TTL caching).
- **Methodology:**
    - Simulate a large-scale registry (e.g., 10^9 – 10^{12} agent records).
    - Generate a realistic query workload emulating diverse capability lookups.
    - Measure average and percentile (e.g., P95, P99) lookup latencies for:
        - Direct resolution with varying TTL cache hit rates.

        - Resolution via a PIR relay.
    - Measure server-side CPU and memory utilization for both scenarios.
- **Expected Outcome:** Demonstrate that PIR-light techniques can offer meaningful privacy enhancements with acceptable performance overhead, especially when combined with intelligent caching of non-sensitive pointers.

8.2. Cryptographic Overhead per Lookup/Interaction:

- **Objective:** Measure the computational cost of cryptographic operations (signature generation/verification for AgentAddr and AgentFacts, VC verification, ZK proof generation/verification) on both client and server sides.
- **Methodology:**
    - Instrument the Rust SDK and registry components to measure execution time for individual cryptographic operations.
    - Simulate typical interaction flows (registration, full resolution path, capability verification).
    - Vary cryptographic parameters (key sizes, signature schemes, ZK circuit complexity) to understand trade-offs.
- **Expected Outcome:** Provide clear data on the performance impact of the security mechanisms, informing choices about cryptographic algorithms and acceptable overhead for different use cases. Identify potential bottlenecks.

8.3. Attack Simulations:

- **Objective:** Assess the resilience of the NANDA+ANS architecture against specific, targeted attacks.
- **Methodology & Scenarios:**
    - **DDoS on Resolution Endpoints (Shuffle-Shard Effectiveness):**
        - Simulate a DDoS attack targeting specific NANDA Quilt shards or ANS resolvers.
        - Measure the impact on overall resolution success rate and latency for unaffected shards.
        - Evaluate the effectiveness of adaptive routing (shuffle-sharding) in mitigating the attack.
    - **Capability Squatting & Misrepresentation Stress Test:**
        - Simulate a large number of malicious agents attempting to register with deceptive names or forged capability attestations.
        - Evaluate the effectiveness of registrar validation policies, staking/fee mechanisms, and dispute resolution in detecting and mitigating these attempts.
        - Measure the false positive/negative rates of detection.
    - **Rapid Revocation Effectiveness:**
        - Simulate the compromise of a significant number of agent credentials.

        - Measure the end-to-end time taken for revocation information (CRL/OCSP updates, VC-Status-List propagation) to become effective across the federated system and prevent further unauthorized access.
    - **PIR Relay Compromise/Analysis:**
        - Simulate a compromised PIR relay attempting to deanonymize queries.
        - Evaluate the residual privacy guarantees under such an attack scenario.
- **Expected Outcome:** Identify potential weaknesses in the implemented defenses, validate the effectiveness of mitigation strategies under pressure, and inform further security hardening efforts.

These benchmarks and evaluations will provide crucial empirical data to support the security claims of the NANDA+ANS blueprint and guide its practical deployment at scale.

9. Roadmap & Call for Contributors

The development and deployment of the NANDA+ANS Security Blueprint will be a phased, community-driven effort. We outline the following key security-focused milestones:

- **Phase 1: Specification & Foundational Security (Q1-Q2 2026)**
    - **Spec v0.5 Threat Model Review:** Public review and refinement of the MAESTRO+STRIDE threat model and initial mitigation strategies by security experts and the broader community.
    - Formalization of cryptographic protocols for AgentAddr, AgentFacts, VCs, and ZK attestations.
    - Development of initial JSON-LD schemas with security considerations (e.g., mandatory signature fields).
    - Release of a preliminary Rust SDK with core cryptographic functionalities.
- **Phase 2: Prototype & Core Security Implementation (Q3-Q4 2026)**
    - Development of a PoC (Proof of Concept) sandbox environment for NANDA Quilt root and ANS CA/RA integration.
    - Implementation of secure registration, resolution, and basic revocation workflows.
    - Initial implementation of PIR-light mechanisms and ZK attestation service stubs.
    - **Spec v0.8 Security Audit & Penetration Test Planning:** External security audit of the core specifications and implemented PoC components. Planning for comprehensive penetration testing.
- **Phase 3: Hardening, Federation & Benchmarking (Q1-Q2 2027)**

    - Implementation of advanced security features: staking/slashing, dispute resolution DAO (initial version), enhanced revocation.
    - Development of reference protocol adapters (A2A, MCP, ACP) with secure contracts.
    - Conduct initial security-focused benchmarks (crypto overhead, basic attack simulations).
    - Federation testing between independent NANDA+ANS instances.
- **Phase 4: Public Beta & Compliance (Q3-Q4 2027)**
    - Public beta release with comprehensive documentation and security best-practice guides.
    - Full-scale penetration testing by independent security firms.
    - **Spec v1.0 OWASP Top 10 for Agentic AI with OWASP AIVSS Certification:** Review and align the final specification and reference implementation with relevant OWASP guidelines (e.g., Top 10 for LLM Applications, AI Security & Privacy Guide) and pursue relevant security certifications if applicable.
    - Establishment of an ongoing security vulnerability disclosure program.

Call for Contributors:

We invite active participation from a diverse group of stakeholders to realize this security blueprint:

- **Cloud Vendors & Infrastructure Providers:** To contribute to scalable hosting solutions, robust CA/RA services, and integration with existing identity and security offerings.
- **AI/Agent Framework Developers (A2A, MCP, ACP, LangChain, AutoGen, etc.):** To develop secure protocol adapters, integrate the Rust SDK, and ensure seamless interoperability.
- **Security Researchers & Academia:** To contribute to threat modeling, formal verification of protocols, development of advanced privacy-preserving techniques, and independent security audits.
- **Cryptographers:** To advise on and contribute to the implementation of ZK-proofs, cryptographic agility, and post-quantum readiness.
- **Standards Bodies (W3C, IETF, OASIS):** To help standardize core NANDA+ANS concepts and ensure alignment with broader web and identity standards.
- **Open Source Community:** To contribute to code development, testing, documentation, and the overall growth of the ecosystem.

Interested parties are encouraged to join our working groups focusing on Core Architecture, Security & Trust, Governance, and Protocol Interoperability. Together, we can build a secure and trustworthy foundation for the Internet of AI Agents.

Appendices

A. Formal Grammars

This appendix will define the precise syntax for key NANDA+ANS identifiers and query structures using Backus-Naur Form (BNF) or Augmented Backus-Naur Form (ABNF) [RFC 5234].

A.1. ANS Naming Structure (ABNF)

```
ans-name          = protocol "://" agent-name "." agent-capability "." provider-name ".v" version ["." extension]

protocol          = "a2a" / "mcp" / "acp" / alpha-string ; Other registered protocol schemes

agent-name        = 1*safe-char

agent-capability  = 1*safe-char                ; Represents a high-level capability

provider-name     = 1*safe-char

version           = major-version ["." minor-version ["." patch-version]]

major-version     = 1*DIGIT

minor-version     = 1*DIGIT

patch-version     = 1*DIGIT

extension         = 1*safe-char                ; e.g., "hipaa", "gov", "instance123"

alpha-string      = 1*ALPHA

safe-char         = ALPHA / DIGIT / "-" / "_"   ; Characters allowed in name components
                                                ; (excluding URI reserved characters and ".")
```

Self-correction/Refinement: The `safe-char` definition needs to be carefully considered to avoid conflicts with URI syntax and ensure broad usability. It might need to be more restrictive or allow for percent-encoding for special characters.

A.2. AgentAddr Structure (Conceptual BNF – Actual structure is JSON-LD)

While AgentAddr is JSON-LD, a conceptual BNF can illustrate its components:

<AgentAddr> ::= "{"

"agent_id" ":" <DID> ","

"facts_url" ":" <URL> ","

["private_facts_url" ":" <URL> ","]

["adaptive_router_url" ":" <URL> ","]

"ttl" ":" <Integer> ","

"signature" ":" <SignatureObject>

"}"

<DID> ::= <string> ; e.g., "did:nanda:..." or "did:web:..."

<URL> ::= <string> ; Standard URI

<SignatureObject> ::= "{" "type" ":" <string>, "value" ":" <base64_string> "}"

A.3. AgentFacts Query Syntax (Simplified Example - A full query language would be more complex)

This would likely be a subset of a standard query language like SPARQL if AgentFacts are treated as RDF, or a custom JSON-Path like syntax if querying raw JSON-LD. For simplicity, a conceptual filter-based syntax:

<Query> ::= "SELECT" <Attributes> "FROM" "AgentFacts" "WHERE" <Conditions>

<Attributes> ::= "*" | <AttributeNameList>

<AttributeNameList> ::= <AttributeName> | <AttributeName> "," <AttributeNameList>

<AttributeName> ::= <string> ; e.g., "capability", "endpoints.static"

<Conditions> ::= <Condition> | <Condition> <LogicalOperator> <Conditions>

<Condition> ::= <AttributeName> <ComparisonOperator> <Value>

<LogicalOperator> ::= "AND" | "OR"

<ComparisonOperator> ::= "=" | "!=" | "CONTAINS" | ">" | "<"

<Value> ::= <string_literal> | <number_literal> | <boolean_literal>

Self-correction/Refinement: A real query syntax would need to handle JSON-LD contexts, nested structures, and potentially more complex graph traversals. This simplified version is for illustrative purposes.

B. JSON-LD Schemas

This appendix will provide complete JSON-LD context definitions and example instance schemas. (Note: Full schemas are verbose; I'll provide key snippets and structure).

B.1. NANDA+ANS Core JSON-LD Context (`@context`)

```
{
  "@context": {
    "nanda": "https://nanda.ai/ns/v1#",
    "ans": "https://ans.example.org/ns/v1#",
    "sec": "https://w3id.org/security#",
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "did": "https://www.w3.org/ns/did/v1#",
    "vc": "https://www.w3.org/2018/credentials#",
    "schema": "http://schema.org/",
    "AgentAddr": "nanda:AgentAddr",
    "agent_id": {"@id": "nanda:agentIdentifier", "@type": "@id"},
    "facts_url": {"@id": "nanda:factsURL", "@type": "@id"},
    "private_facts_url": {"@id": "nanda:privateFactsURL", "@type": "@id"},
    "adaptive_router_url": {"@id": "nanda:adaptiveRouterURL", "@type": "@id"},
```

```
    "ttl": {"@id": "nanda:timeToLive", "@type": "xsd:integer"},
    "signature": {"@id": "sec:signature"},
    "AgentFacts": "nanda:AgentFacts",
    "ansName": "ans:ansName",
    "capability": "ans:agentCapability",
    "provider": "ans:providerName",
    "version": "ans:version",
    "extension": "ans:extension",
    "endpoints": "nanda:endpoints",
    "static_endpoint": "nanda:staticEndpoint",
    "rotating_endpoint": "nanda:rotatingEndpoint",
    "attestations": {"@id": "nanda:attestations", "@type": "@id", "@container": "@set"},
    "protocolExtensions": "ans:protocolExtensions",
    "VerifiableCredential": "vc:VerifiableCredential",
    "credentialSubject": "vc:credentialSubject",
    "issuer": {"@id": "vc:issuer", "@type": "@id"},
    "issuanceDate": {"@id": "vc:issuanceDate", "@type": "xsd:dateTime"},
    "proof": {"@id": "sec:proof"},
    "A2AAgentCard": "ans:A2AAgentCard", // Example protocol extension
    "MCPToolDescription": "ans:MCPToolDescription" // Example protocol extension
  }
}
```

B.2. `AgentAddr` Instance Schema (Example)

```
{
  "@context": "https://nanda.ai/contexts/nanda-ans-core-v1.jsonld",
  "@type": "AgentAddr",
  "agent_id": "did:nanda:12345abcdef",
  "facts_url": "https://example.com/.well-known/agent-facts.jsonld",
  "private_facts_url": "ipfs://QmWxyz...",
  "adaptive_router_url": "https://router.example.com/dispatch",
  "ttl": 3600,
  "signature": {
    "type": "Ed25519Signature2020",
    "created": "2025-05-15T10:00:00Z",
    "verificationMethod": "did:nanda:registry-shard-1#key-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "z123..."
  }
}
```

B.3. `AgentFacts` Instance Schema (Partial Example)

```
{
  "@context": "https://nanda.ai/contexts/nanda-ans-core-v1.jsonld",
  "@type": "AgentFacts",
  "id": "did:nanda:agent-xyz-789", // Agent's own DID
```

```
"ansName": "a2a://myTranslator.DocumentTranslation.LinguaCorp.v1.2.certified",
"capability": "DocumentTranslation",
"provider": "LinguaCorp",
"version": "1.2",
"extension": "certified",
"description": "High-accuracy document translation agent.",
"endpoints": {
  "@type": "nanda:EndpointSet",
  "static_endpoint": ["https://api.linguacorp.com/translate/v1.2"],
  "adaptive_router_url": "https://router.linguacorp.com/translate"
},
"attestations": [
  "did:nanda:attestation-abc-123" // Pointer to a Verifiable Credential
],
"protocolExtensions": {
  "@type": "ans:ProtocolExtensionSet",
  "a2a": {
    "@type": "A2AAgentCard",
    // ... A2A specific fields ...
  }
},
"signature": {
```

```
    // Signature by the agent (did:nanda:agent-xyz-789) or its authorized publisher
  }
}
```

B.4. Verifiable Credential for Capability Attestation Schema (See Appendix D)

B.5. `protocolExtension` Base Structure (Conceptual)

The `protocolExtensions` field within `AgentFacts` would be an object where keys are protocol identifiers (e.g., "a2a", "mcp", "acp") and values are objects conforming to schemas defined by those respective protocols or by community agreement for interoperability within NANDA+ANS.

```
"protocolExtensions": {
  "a2a": { /* A2A Agent Card structure */ },
  "mcp": { /* MCP Tool Description structure */ },
  "acp": { /* ACP Profile structure */ }
}
```

Each protocol-specific structure would need its own JSON-LD context or be defined within the core context.

C. Zero-Knowledge Circuit Sketch

This appendix describes a high-level ZK circuit for a common attestation, like proving an agent's model meets a minimum version requirement without revealing the exact version.

C.1. Use Case: Proving Model Version Threshold

- **Goal:** An agent (Prover) wants to prove to a client (Verifier) that its underlying AI model version `V_model` is greater than or equal to a required minimum version `V_min`, without revealing `V_model`.
- **Public Inputs (known to Verifier):**
    - `V_min` (the minimum required version)
    - `Commitment(V_model, r)` (a cryptographic commitment to `V_model` and a random salt `r`, previously published by the agent or an attestor in its AgentFacts)

- **Private Inputs (known only to Prover):**
    - `V_model` (the agent's actual model version)
    - `r` (the random salt used in the commitment)

C.2. ZK-SNARK Circuit (High-Level Pseudocode/Description):

```
// Circuit: CheckModelVersionThreshold

// Inputs:

//   public V_min

//   public Commitment_V_model_r

//   private V_model

//   private r

function main() {

  // 1. Verify the commitment

  //    Assert that hash(V_model, r) == Commitment_V_model_r

  //    This ensures the Prover is using the V_model they previously committed to.

  assert_equal(hash(V_model, r), Commitment_V_model_r);

  // 2. Verify the version threshold

  //    Assert that V_model >= V_min

  //    This is the core statement being proven.

  assert_greater_than_or_equal(V_model, V_min);

  // If both assertions pass, the proof is valid.

  // The ZK-SNARK proof itself will not reveal V_model or r.

  return 1; // Signifies success

}
```

C.3. Workflow:

1. **Setup:** A trusted setup phase generates proving and verification keys for this specific circuit.
2. **Commitment:** The agent (or an attestor) publishes `Commitment(V_model, r)` in its AgentFacts.
3. **Proof Generation:** When a client requires proof, the agent (Prover) uses `V_model`, `r`, `V_min`, and `Commitment_V_model_r` as inputs to the ZK-SNARK proving algorithm with the proving key, generating a compact proof π.
4. **Proof Verification:** The client (Verifier) uses `V_min`, `Commitment_V_model_r`, the proof π, and the verification key as inputs to the ZK-SNARK verification algorithm. If it returns true, the client is convinced `V_model >= V_min` without learning `V_model`.

Self-correction/Refinement: The choice of ZK-SNARK (e.g., Groth16, PLONK) and hashing algorithm would impact circuit design and performance. The commitment scheme also needs careful selection. Representing versions (e.g., semantic versioning) as numbers suitable for arithmetic comparison within the circuit is a key detail.

D. Sample Verifiable Credential for Capability Attestation

This appendix provides a concrete JSON-LD example of a W3C Verifiable Credential.


```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://nanda.ai/contexts/nanda-ans-core-v1.jsonld", // For terms like 'ans:agentCapability'
    "https://w3id.org/security/suites/ed25519-2020/v1" // For the proof type
  ],
  "id": "did:nanda:attestation-lingua-translator-hipaa-compliance-2025",
  "type": ["VerifiableCredential", "ans:CapabilityAttestation"],
  "issuer": "did:web:trusted-auditors.com", // DID of the attesting organization
  "issuanceDate": "2025-04-01T12:00:00Z",
```

```
  "credentialSubject": {
    "id": "did:nanda:agent-xyz-789", // DID of the agent being attested
    "ans:agentCapability": "DocumentTranslation",
    "nanda:complianceStandard": "HIPAA",
    "nanda:attestationDetails": {
      "scope": "Translation of patient records",
      "auditReport": "https://trusted-auditors.com/reports/lingua-hipaa-2025.pdf"
    }
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2025-04-01T12:05:00Z",
    "verificationMethod": "did:web:trusted-auditors.com#key-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "zABC..." // Base58BTC encoded signature
  }
}
```

This VC attests that the agent `did:nanda:agent-xyz-789`, which offers the `DocumentTranslation` capability, is compliant with HIPAA for a specific scope, as verified by `did:web:trusted-auditors.com`.

E. Economic Model Equations & Simulations

This appendix would detail the mathematical formulations for staking, slashing, and fees, and present simulation results.

E.1. Staking Requirement Function:

```
Stake_required(AgentType, CapabilityRisk, RegistrationHistory) =
BaseStake * f(AgentType) * g(CapabilityRisk) *
h(RegistrationHistory)
```

- `BaseStake`: A minimum staking amount.
- `f(AgentType)`: Multiplier based on agent type (e.g., individual, enterprise, critical infrastructure).
- `g(CapabilityRisk)`: Multiplier based on the risk associated with the advertised primary capability (e.g., higher for financial agents).
- `h(RegistrationHistory)`: Discount factor for agents with a long, positive history.

E.2. Slashing Penalty Function:

```
Penalty_slashed(Severity, Stake_held, Impact_assessed) =
min(Stake_held, BasePenalty * k(Severity) + l(Impact_assessed))
```

- `BasePenalty`: A base penalty amount.
- `k(Severity)`: Multiplier based on the severity of the violation.
- `l(Impact_assessed)`: Additional penalty based on the assessed impact of the misbehavior.

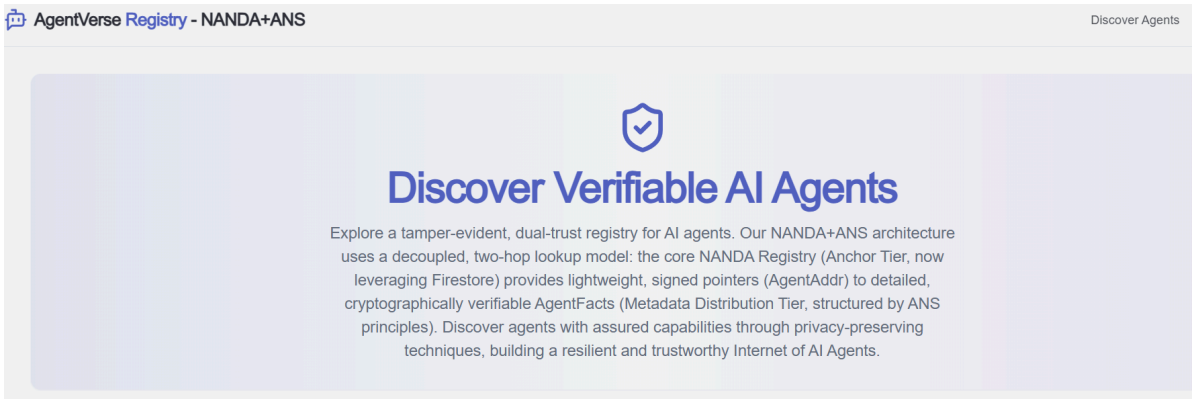
E.3. Registrar Fee Model:

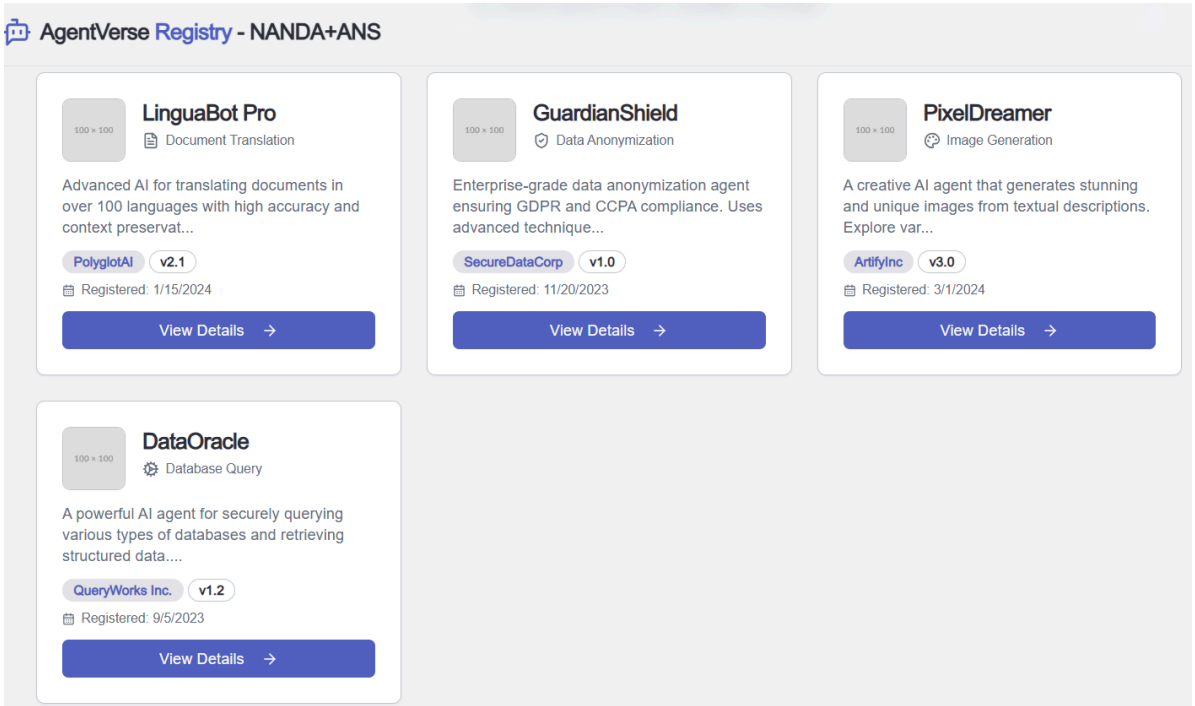
```
Fee_registration = Fixed_component +
Variable_component(Resource_usage) Fee_renewal = Discount_factor *
Fee_registration
```

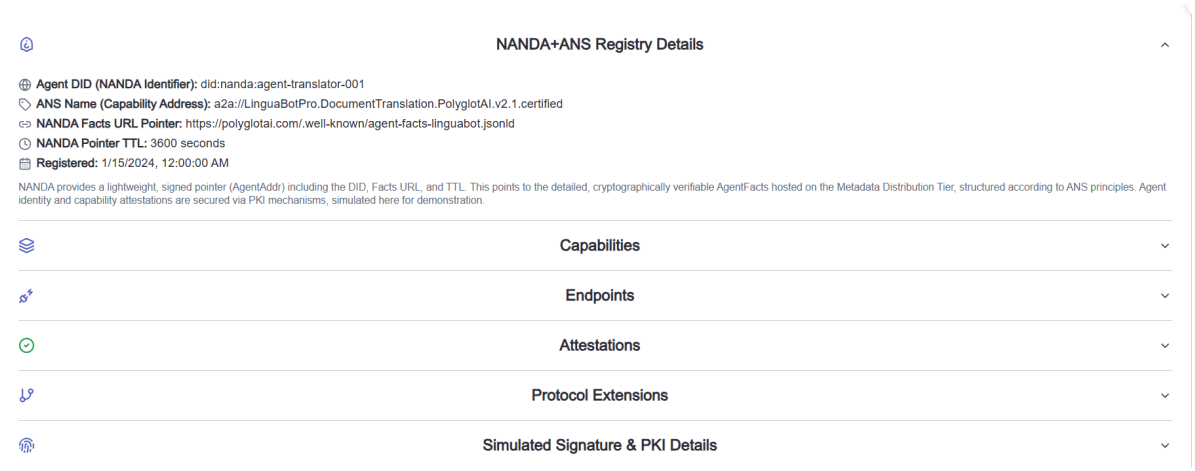
E.4. Simulation Goals & Setup (Example):

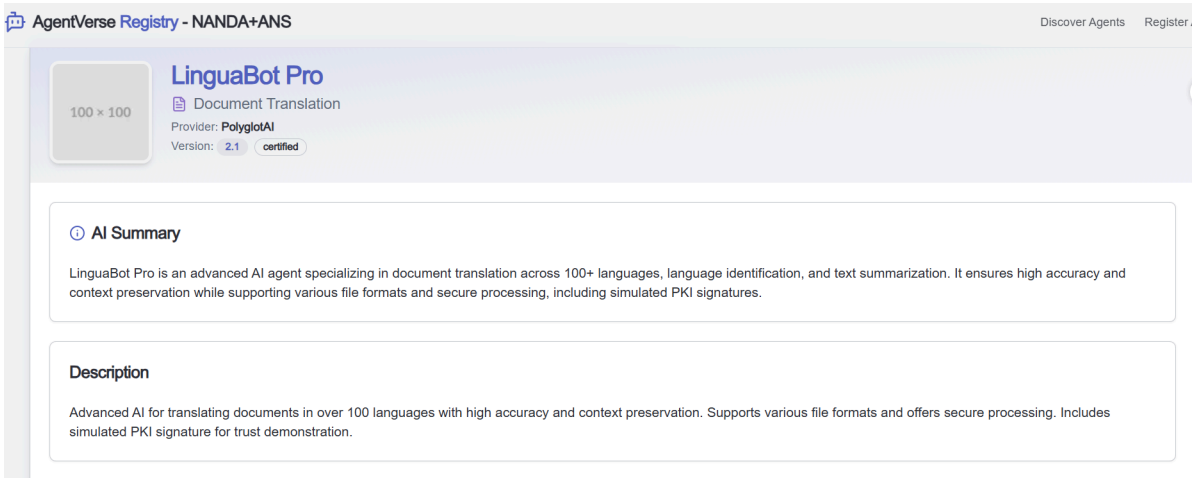
- **Goal:** Evaluate the effectiveness of staking in deterring Sybil attacks.
- **Setup:** Simulate a registry with `N` agents. Introduce `M` Sybil attackers attempting to register `S` fake identities each. Vary `BaseStake` and monitor the cost to attackers versus the number of successful malicious registrations.
- **Metrics:**
    - Cost per successful malicious registration.
    - Ratio of legitimate to malicious registrations.
    - Impact on registry resource consumption.

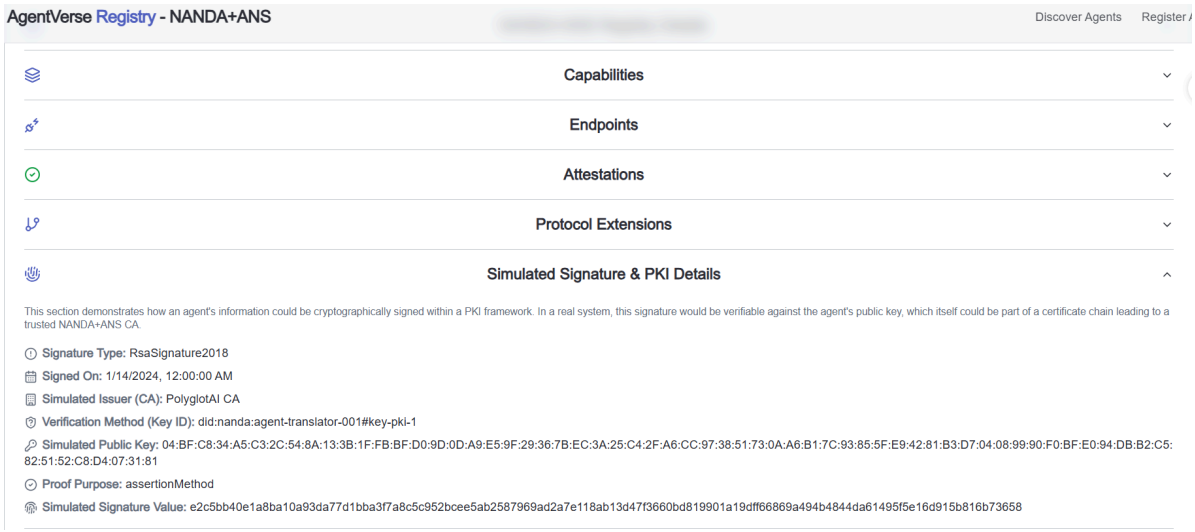
MVP Screenshots






NANDA+ANS Registry Details
Agent DID (NANDA Identifier): did:nanda:agent-translator-001
ANS Name (Capability Address): a2a://LinguaBotPro.DocumentTranslation.PolyglotAI.v2.1.certified
NANDA Facts URL Pointer: https://polyglotai.com/.well-known/agent-facts-linguabot.jsonld
NANDA Pointer TTL: 3600 seconds
Registered: 1/15/2024, 12:00:00 AM
NANDA provides a lightweight, signed pointer (AgentAddr) including the DID, Facts URL, and TTL. This points to the detailed, cryptographically verifiable AgentFacts hosted on the Metadata Distribution Tier, structured according to ANS principles. Agent identity and capability attestations are secured via PKI mechanisms, simulated here for demonstration.
Capabilities
Endpoints
Attestations
Protocol Extensions
Simulated Signature & PKI Details


AgentVerse Registry - NANDA+ANS
Discover Agents
Register
100 × 100
LinguaBot Pro
Document Translation
Provider: PolyglotAI
Version: 2.1
certified
AI Summary
LinguaBot Pro is an advanced AI agent specializing in document translation across 100+ languages, language identification, and text summarization. It ensures high accuracy and context preservation while supporting various file formats and secure processing, including simulated PKI signatures.
Description
Advanced AI for translating documents in over 100 languages with high accuracy and context preservation. Supports various file formats and offers secure processing. Includes simulated PKI signature for trust demonstration.



References

Huang, K., Narajala, V. S., Yeoh, J., Raskar, R., Harkati, Y., Huang, J., Habler, I., & Hughes, C. (2025). *A novel zero-trust identity framework for agentic AI: Decentralized authentication and fine-grained access control*. arXiv. https://doi.org/10.48550/arXiv.2505.19301

Huang, K., Narajala, S., Habler, M., & Sheriff, I. (2025). Agent Name Service (ANS): A Universal Directory for Secure AI Agent Discovery and Interoperability. https://arxiv.org/abs/2505.10609

Raskar, R., Chari, P., Lambe, M., Grogan, J., Ranjan, R., Zinky, J., Singhal, R., Gupta, S., Lincourt, R., Singh, S. N., Bala, R., Singh, A., Chopra, A., Stripelis, D., Kumar, S., & Gorskikh, M. (2025). Scaling Trust Beyond DNS: How NANDA Registry and Verified AgentFacts Unlock the Internet of AI Agents.

Huang, K. (2025). Agentic AI Threat Modeling Framework: MAESTRO, https://cloudsecurityalliance.org/blog/2025/02/06/agentic-ai-threat-modeling-framework-maestro

Surapaneni, R., Jha, M., Vakoc, M., & Segal, T. (2025). Announcing the Agent2Agent Protocol (A2A). Google for Developers Blog. https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/

Anthropic. (2024). Model Context Protocol (MCP). Anthropic News. https://www.anthropic.com/news/model-context-protocol