Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 26 September 2026

R. Sharif
CyberSecAI Ltd

25 March 2026

Trust Scoring and Identity Verification for Autonomous AI Agent Payment Transactions

draft-sharif-agent-payment-trust-00

Abstract

This document specifies a protocol for trust scoring, identity verification, and spend limit enforcement for autonomous AI agents that initiate financial transactions. As AI agents gain the capability to make payments via protocols such as the Machine Payments Protocol (MPP), a standardised mechanism is needed to verify agent identity, assess trustworthiness, and enforce financial limits based on behavioural history.

The protocol defines a five-dimension trust scoring model, per-agent cryptographic identity using ECDSA P-256 key pairs, challenge-response identity verification, spend limit tiers derived from trust scores, anomaly detection for financial behaviour, and a public trust query API for third-party platforms.

This specification complements draft-sharif-mcps-secure-mcp, which provides message-level cryptographic security for the Model Context Protocol (MCP). Together, the two specifications address protocol security (MCPS) and financial trust (this document) for the AI agent economy.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 26 September 2026.

Table of Contents

1. Introduction

The emergence of autonomous AI agents capable of initiating financial transactions represents a fundamental shift in payment infrastructure. Protocols such as Stripe's Machine Payments Protocol (MPP) [STRIPE-MPP], launched in March 2026, enable AI agents to create payment intents, process charges, and manage subscriptions without human intervention.

Existing payment security mechanisms -- Strong Customer Authentication (SCA) under PSD2 [PSD2], fraud detection systems, and Know Your Customer (KYC) processes -- are designed for human users who are present at the point of transaction. These mechanisms do not address the unique characteristics of AI agent transactions:

- Agents operate at machine speed, potentially executing thousands of transactions per minute.
- Agents act autonomously, without real-time human oversight for each transaction.
- Multiple agents may operate under a single developer account, each with different behavioural patterns and risk profiles.
- Agent identity is typically established through static API keys that provide no per-agent differentiation.
- There is no standardised mechanism for a platform to assess an agent's trustworthiness before granting financial access.

This document specifies a protocol that addresses these gaps through four mechanisms:

1. Per-agent cryptographic identity using ECDSA P-256 key pairs with challenge-response verification.
2. A five-dimension behavioural trust scoring model that dynamically adjusts based on observed agent behaviour.
3. Spend limit tiers derived from trust scores, enforced at the protocol level before transactions reach payment networks.
4. A public trust query API that enables any platform to assess agent trustworthiness without requiring a pre-existing relationship with the trust authority.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Agent: An autonomous software entity that initiates financial transactions on behalf of a principal (human or organisation) via standardised protocols.

Agent Passport: A cryptographically signed identity document containing the agent's public key hash, developer identity, authorised scope, and issuance metadata.

Challenge: A cryptographically random value issued by a verifier

to test whether a claimant possesses a specific private key.

Developer: The human or organisational entity that creates, manages, and is accountable for one or more agents.

Trust Authority: The service that maintains agent registrations, computes trust scores, verifies identities, and responds to trust queries.

Trust Score: A numerical value (0-100) representing the assessed trustworthiness of an agent based on multiple behavioural dimensions.

Trust Level: A discrete classification (L0-L4) derived from the trust score, with associated financial limits.

3. Problem Statement

Consider the following scenario: an AI agent operating as a procurement assistant for a small business is authorised to purchase cloud computing resources. The agent holds an API key for a payment processor and can create payment intents autonomously.

An attacker who obtains the agent's API key -- through a compromised development environment, leaked configuration file, or man-in-the-middle attack -- can:

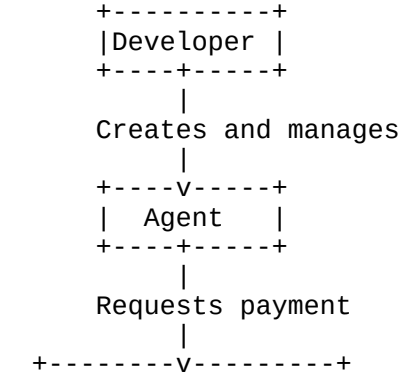
- Impersonate the agent and make unauthorised purchases.
- Exceed the intended spending authority of the agent.
- Create transactions at machine speed before the compromise is detected.
- Operate across multiple platforms simultaneously using the same stolen credentials.

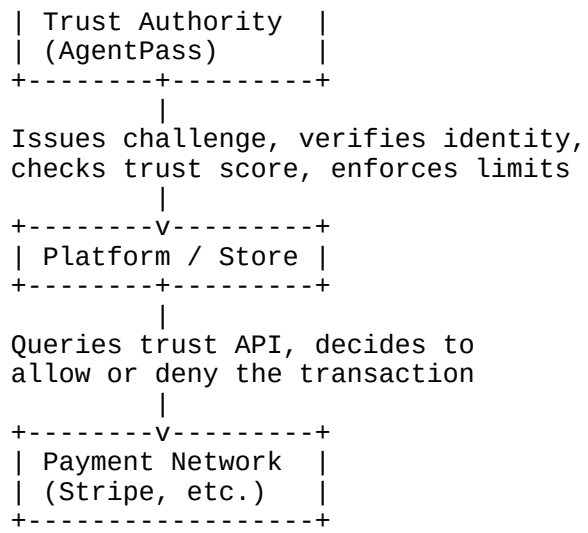
Current mitigations (API key rotation, IP allowlisting) are reactive rather than preventive. They do not address the fundamental question: "Is this specific agent, making this specific transaction, trustworthy?"

This document specifies a protocol that answers that question through cryptographic identity verification and behavioural trust scoring.

4. Architecture Overview

The protocol defines interactions between four roles:





The Trust Authority MUST NOT be in the critical path of payment processing. Platforms query the Trust Authority before initiating payments; if the Trust Authority is unavailable, platforms SHOULD apply a fail-closed policy for new agents and a cached-trust policy for previously verified agents.

5. Agent Identity

5.1. Key Pair Generation

Each agent MUST have a unique ECDSA key pair using the P-256 curve [FIPS186-4].

Key pairs MAY be generated by the Trust Authority and provided to the developer (the private key MUST be returned exactly once and MUST NOT be stored by the Trust Authority), or generated by the developer and the public key registered with the Trust Authority (RECOMMENDED for production deployments).

5.2. Public Key Registration

The developer MUST register the agent's public key with the Trust Authority at agent creation time. The Trust Authority stores ONLY the public key. The private key MUST remain under the developer's control at all times.

For production deployments, the private key SHOULD be stored in a Hardware Security Module (HSM), cloud Key Management Service (KMS), or platform-native secure storage (e.g., iOS Keychain with kSecAttrAccessibleWhenUnlockedThisDeviceOnly).

5.3. Agent Passport

Upon registration, the Trust Authority issues an Agent Passport -- a signed JSON document containing:

- agentId: Unique identifier for the agent.
- agentPublicKeyHash: SHA-256 hash of the agent's public key, binding the passport to a specific key pair.
- developerId: Identifier of the owning developer.
- scope: Array of authorised action categories.
- issuedAt: ISO 8601 timestamp of issuance.
- issuer: Identifier of the Trust Authority.

The passport is signed by the developer's key (not the agent's key), establishing an issuance chain: Trust Authority issues to

Developer, Developer delegates to Agent.

6. Challenge-Response Identity Verification

When a platform needs to verify an agent's identity before granting access or processing a payment, it uses the following challenge-response protocol.

6.1. Challenge Issuance

The platform (or Trust Authority on behalf of the platform) generates a challenge:

- challenge: 32 bytes of cryptographically random data, hex-encoded (64 characters).
- expiresAt: Timestamp, MUST be no more than 60 seconds from issuance.
- agentId: The agent for which the challenge is issued.

Each challenge MUST be single-use. The Trust Authority MUST reject any attempt to verify a challenge that has already been used (replay prevention).

6.2. Challenge Signing

The agent signs the challenge using its private key:

```
signature = ECDSA-Sign(SHA-256(challenge), agentPrivateKey)
```

The signature is hex-encoded for transmission.

6.3. Signature Verification

The Trust Authority verifies the signature using the agent's registered public key:

```
valid = ECDSA-Verify(SHA-256(challenge), signature,
                     agentPublicKey)
```

If valid, the Trust Authority returns the agent's current trust score, level, and recommendation.

6.4. Failure Handling

If verification fails:

- The Trust Authority MUST log the failure as a potential impersonation attempt.
- The agent's trust score MUST be penalised (RECOMMENDED: -10 points per failed attempt).
- The agent's anomaly counter MUST be incremented.
- The response MUST include an error code indicating the failure type: IMPERSONATION_DETECTED, CHALLENGE_EXPIRED, CHALLENGE_REPLAYED, or AGENT_MISMATCH.

7. Trust Scoring Model

7.1. Scoring Dimensions

The trust score is computed from five dimensions, each contributing a weighted component to the overall score:

1. Code Attestation (CA): Whether the agent's code has been cryptographically verified against its declared specification.

2. Execution Success Rate (ES): The proportion of the agent's past transactions that completed without anomaly, dispute, or reversal.

3. Behavioural Consistency (BC): The statistical consistency of the agent's transaction patterns (amounts, recipients, timing) relative to its established baseline.

4. Operational Tenure (OT): The duration the agent has been registered and actively transacting.

5. Anomaly History (AH): The inverse of the count and severity of detected anomalies (fewer anomalies = higher score).

7.2. Dimension Weights

Each dimension contributes 20% to the overall score:

```
W_CA = W_ES = W_BC = W_OT = W_AH = 0.20
```

Implementations MAY adjust weights provided the sum equals 1.0 and no single dimension exceeds 0.40.

7.3. Score Computation

The raw trust score is computed as:

```
raw = (CA * W_CA) + (ES * W_ES) + (BC * W_BC) +
      (OT * W_OT) + (AH * W_AH)
```

Each dimension value is in the range [0, 100].

The final score incorporates a trust bonus (earned through successful transactions, capped at +/- 30) and a dormancy penalty:

```
score = clamp(raw + bonus + dormancy, 0, 100)
```

7.4. Trust Levels

The trust score maps to discrete levels with associated financial limits:

Level	Score	Per-TX Limit	Daily Limit
L0	0-19	$0	$0
L1	20-39	$10	$50
L2	40-59	$100	$500
L3	60-79	$1,000	$5,000
L4	80-100	$50,000	$200,000

No trust level SHALL have unlimited financial access. L4 represents the maximum tier with mandatory enhanced auditing.

7.5. Dynamic Adjustment

The trust score adjusts dynamically based on observed behaviour:

- Successful transaction: +0.5 trust bonus.
- Blocked transaction (over limit): -2 trust bonus.
- Anomaly detected: -5 per anomaly.
- Critical anomaly (3+ simultaneous): -20 trust bonus.
- Failed identity verification: -10 trust bonus.

- Probing detected: -15 trust bonus.

Self-dealing transactions (agent-to-agent payments within the same developer account) MUST earn zero trust bonus.

7.6. Dormancy Decay

Agents that are inactive for extended periods MUST have their trust score reduced:

- 30 days inactive: -10 penalty.
- 60 days inactive: -20 penalty.
- 90+ days inactive: -30 penalty (effectively drops to L0).

The dormancy penalty is applied at score computation time and is removed when the agent resumes activity.

8. Spend Limit Enforcement

8.1. Per-Transaction Limits

Each transaction MUST be checked against the per-transaction limit of the agent's current trust level. Transactions exceeding the limit MUST be rejected with error code MCPS-SPEND-TX-LIMIT.

8.2. Daily Aggregate Limits

The Trust Authority MUST track daily aggregate spending per agent within a rolling 24-hour window. Transactions that would cause the aggregate to exceed the daily limit MUST be rejected with error code MCPS-SPEND-DAILY-LIMIT.

8.3. Developer Aggregate Limits

To prevent Sybil attacks (creating many agents to circumvent per-agent limits), the Trust Authority MUST enforce aggregate daily spending limits across ALL agents belonging to a single developer.

8.4. Promotion Rate Limiting

To prevent trust farming, agents MUST NOT be promoted to a higher trust level until a minimum period has elapsed at the current level:

- L0 to L1: No minimum.
- L1 to L2: 7 days.
- L2 to L3: 14 days.
- L3 to L4: 30 days.

9. Anomaly Detection

The Trust Authority MUST monitor agent financial behaviour for the following anomaly types.

9.1. Magnitude Anomaly

A transaction amount that exceeds 5x the agent's historical average.

9.2. Velocity Anomaly

More than 10 transactions within a 60-second window.

9.3. Temporal Anomaly

Transactions initiated outside the agent's established operational time window.

9.4. Recipient Anomaly

Transactions to recipients the agent has never previously transacted with, flagged after the agent has established a baseline of 5+ transactions (informational severity only).

9.5. Probing Anomaly

Three or more transactions within one hour where the amount exceeds 85% of the per-transaction limit. This pattern indicates systematic probing of authorisation boundaries.

Probing detection MUST trigger immediate blocking and an accelerated trust penalty.

9.6. Self-Dealing Detection

Transactions between agents belonging to the same developer MUST be flagged as self-dealing. Self-dealing transactions MUST NOT contribute positive trust bonus to either agent.

10. Public Trust Query API

10.1. Request Format

Platforms query agent trustworthiness via HTTP GET:

```
GET /api/public/trust/{agentId}
```

No authentication is required for public trust queries (same model as DNS lookups or certificate transparency logs).

10.2. Response Format

The response includes:

- agentId: Agent identifier.
- status: ACTIVE or REVOKED.
- trust.score: Numerical trust score (0-100).
- trust.level: Trust level (0-4).
- recommendation: ALLOW, ALLOW_WITH_LIMITS, or DENY.
- spendLimits: Per-transaction and daily limits.
- history: Transaction count, success rate, operational days.

10.3. Batch Queries

For efficiency, platforms MAY query multiple agents in a single request:

```
POST /api/public/trust/batch
Body: { "agentIds": ["agent_abc", "agent_def"] }
```

10.4. Rate Limiting

The Trust Authority MUST enforce rate limiting on public trust queries to prevent abuse (RECOMMENDED: 120 queries per minute per source IP).

10.5. Information Disclosure Controls

The public trust API MUST NOT return trust scoring dimension breakdowns. Full dimension data is available only via authenticated API to the agent's owner. This prevents attackers from identifying the weakest dimension and gaming it specifically.

11. Transaction Signing and Receipts

11.1. Transaction Envelope

Each authorised transaction MUST be wrapped in a signed envelope containing: agent identity, trust level at time of authorisation, unique transaction nonce, timestamp, and ECDSA signature covering the canonical serialisation of all fields.

11.2. Hash Chain

Each transaction record MUST include a reference to the SHA-256 hash of the previous transaction in the agent's audit trail. This creates a tamper-evident chain where modification of any historical record is computationally detectable.

11.3. Non-Repudiable Receipts

The signed transaction envelope constitutes a non-repudiable receipt proving: which agent made the request, the exact parameters at the time of signing, when the request was created, and the agent's trust level at authorisation time.

12. Revocation

12.1. Agent Revocation

An agent MAY be revoked by its developer at any time. Revocation MUST take effect within 1 second and MUST be reflected in all subsequent trust queries. Revoked agents MUST receive a DENY recommendation on all trust queries.

12.2. Key Rotation

Developers SHOULD rotate agent key pairs periodically (RECOMMENDED: every 90 days). Key rotation MUST NOT reset the agent's trust score.

12.3. Emergency Freeze

The Trust Authority MUST support an emergency freeze mechanism that immediately halts all financial transactions for a specific agent, developer, or globally.

13. Threat Model

This section defines the adversary profiles, attack vectors, and mitigations that the protocol is designed to address.

13.1. Adversary Profiles

The protocol considers four adversary types:

A1. External Attacker: Has no legitimate access to the system. Goal: impersonate a trusted agent to make unauthorised payments. Capabilities: network interception, credential

harvesting from public sources, brute-force key guessing.

A2. Compromised Developer: A legitimate developer whose infrastructure has been breached. Goal: use stolen agent credentials to make payments before the compromise is detected. Capabilities: possesses valid API keys and potentially agent private keys.

A3. Malicious Developer: A legitimate account holder who intentionally abuses the system. Goal: extract maximum financial value through trust manipulation, Sybil attacks, or self-dealing. Capabilities: full control over their agents and account.

A4. Compromised Trust Authority: The Trust Authority itself is breached. Goal: forge trust scores, issue false identity verifications, or exfiltrate stored data. Capabilities: access to public keys, trust scores, and transaction metadata (but NOT agent private keys if the protocol is correctly implemented).

13.2. Attack Tree: Agent Impersonation

Attack goal: Make a payment as Agent X without possessing Agent X's private key.

```
Attack Path 1: Steal the private key
|-- 1a. Compromise developer infrastructure (A2)
|   |-- Mitigation: HSM/KMS storage (Section 5.2)
|   |-- Mitigation: Keychain with device-only access
|   |-- Residual risk: If key is in plaintext on disk,
|       attacker succeeds. Protocol CANNOT prevent this;
|       it can only detect anomalous behaviour post-compromise.
|
|-- 1b. Intercept key during generation (A1)
|   |-- Mitigation: TLS on all API communications
|   |-- Mitigation: Developer-managed keys (key never
|       transmitted)
|
|-- 1c. Extract from Trust Authority (A4)
    |-- Mitigation: Trust Authority MUST NOT store private
        keys (Section 5.2). This path is blocked by design.

Attack Path 2: Bypass identity verification
|-- 2a. Replay a valid challenge-response (A1)
|   |-- Mitigation: Single-use challenges (Section 6.1)
|   |-- Mitigation: 60-second expiry window
|
|-- 2b. Forge a signature without the key (A1)
|   |-- Mitigation: ECDSA P-256 computational infeasibility
|   |-- Estimated cost: >2^128 operations
|
|-- 2c. Present a different agent's passport (A1)
|   |-- Mitigation: Passport contains agentPublicKeyHash
|       (Section 5.3), binding it to a specific key pair
|
|-- 2d. Skip identity verification entirely (A1)
    |-- Mitigation: Platform policy. If the platform does
        not require challenge-response, this succeeds.
        RECOMMENDATION: Platforms SHOULD require identity
        verification for transactions above L2 limits.

Attack Path 3: Social engineering
|-- 3a. Convince a platform to trust a low-score agent (A3)
    |-- Mitigation: Public trust API provides objective score.
        Platform makes its own decision.
```

13.3. Attack Tree: Trust Score Manipulation

Attack goal: Artificially inflate an agent's trust score to gain higher spending authority.

```
Attack Path 1: Trust farming
|-- Make many small legitimate payments to build score
|-- Then make one large fraudulent payment at elevated level
|-- Mitigation: Promotion rate limiting (Section 8.4)
|   Minimum 7/14/30 days per level before promotion.
|-- Mitigation: Magnitude anomaly detection (Section 9.1)
|   Sudden increase in transaction size is flagged.
|-- Residual risk: Patient attacker over 60+ days could
    reach L4. Mitigated by L4 cap ($50,000/tx) and
    mandatory enhanced auditing.

Attack Path 2: Self-dealing
|-- Create two agents, send payments back and forth
|-- Mitigation: Self-dealing detection (Section 9.6)
|   Agent-to-agent payments within same developer earn
|   zero trust bonus.

Attack Path 3: Sybil attack
|-- Create 100 agents at L1 ($50/day each = $5,000/day)
|-- Mitigation: Developer aggregate limits (Section 8.3)
|   Total spending across ALL agents is capped.

Attack Path 4: Score gaming via dimension targeting
|-- Query public API, identify weakest dimension, game it
|-- Mitigation: Public API hides dimension breakdown
|   (Section 10.5). Full dimensions only via authenticated
|   API for the agent's owner.
```

13.4. Attack Tree: Denial of Service

Attack goal: Prevent legitimate agents from making payments.

```
Attack Path 1: Exhaust rate limits
|-- Flood trust API queries to trigger rate limiting
|-- Mitigation: Per-IP rate limiting (Section 10.4)
|-- Residual risk: Distributed attack from many IPs.
    Mitigation: WAF/CDN layer (implementation-specific).

Attack Path 2: Trigger false anomalies
|-- Make unusual payments to trigger anomaly flags on a
    target agent
|-- Mitigation: Only the agent's own developer can make
    payments from that agent. External attackers cannot
    trigger anomalies without the agent's credentials.

Attack Path 3: Challenge flooding
|-- Request thousands of challenges for an agent
|-- Mitigation: Rate limiting on challenge issuance.
|   Challenges are lightweight (no crypto operations
|   until verification).
```

13.5. Attack Tree: Financial Fraud Post-Compromise

Attack goal: Maximise financial extraction after obtaining valid agent credentials.

Scenario: Attacker has stolen Agent X's private key and API key (Adversary A2).

```
|-- Step 1: Verify identity to confirm credentials work
```

```
|   Detection: Challenge-response succeeds, but from
|   a new IP/location. Implementations SHOULD log IP
|   and alert on geographic anomalies.
|
|-- Step 2: Make payments up to current trust level limit
|   Detection: Magnitude anomaly if patterns differ from
|   historical baseline. Velocity anomaly if transactions
|   are rapid. Temporal anomaly if outside normal hours.
|
|-- Step 3: Attempt to maximise extraction speed
|   Mitigation: Daily aggregate limits cap total damage.
|   At L3: max $5,000/day. At L4: max $200,000/day.
|   Per-TX limits prevent single catastrophic transactions.
|
|-- Step 4: Developer detects and revokes
|   Mitigation: Instant revocation (Section 12.1).
|   All trust queries immediately return DENY.
|
|-- Maximum exposure window: Time between compromise and
    detection. Anomaly detection reduces this window.
    RECOMMENDED: Developers configure webhook alerts for
    unusual activity.
```

Maximum financial exposure by trust level:

Level	Max per day	Max before likely detection (est.)
L1	$50	$50
L2	$500	$500
L3	$5,000	$5,000
L4	$200,000	$200,000

Note: These are worst-case figures assuming the attacker perfectly mimics the agent's historical behaviour to avoid anomaly detection. In practice, behavioural differences between the legitimate agent and the attacker will trigger anomalies earlier, reducing the exposure window.

13.6. Residual Risks

The following risks are acknowledged but not fully mitigated by this protocol:

R1. Private key compromise with HSM bypass: If an attacker obtains the private key through a zero-day HSM vulnerability, the protocol cannot distinguish the attacker from the legitimate agent until behavioural anomalies are detected.

R2. Trust Authority compromise: If the Trust Authority is breached, the attacker can forge trust scores. Mitigation: trust query responses SHOULD be signed by the Trust Authority's own key, enabling platforms to verify response authenticity.

R3. Collusion between developer and attacker: A malicious developer who intentionally provides their agent credentials to a third party cannot be distinguished from a compromised developer. Mitigation: developer aggregate limits cap total exposure. Legal/contractual controls are out of scope.

R4. Race conditions in distributed deployments: Under high

concurrency, two transactions may simultaneously pass limit checks. Mitigation: implementations MUST use atomic operations. Residual risk depends on implementation correctness.

R5. Currency arbitrage: Limits are enforced on raw transaction amounts. An agent transacting in a volatile currency could exploit exchange rate differences. Mitigation: implementations SHOULD normalise all amounts to a base currency (RECOMMENDED: USD) at transaction time.

R6. Long-horizon trust farming: A patient adversary who operates legitimately for 60+ days can reach L4. Mitigation: L4 has finite limits ($50,000/tx, $200,000/day), mandatory enhanced auditing, and 30-day promotion cooldown from L3. The cost of maintaining a legitimate operation for 60+ days to reach L4 limits may exceed the potential fraud proceeds.

14. Privacy Considerations

The public trust API returns agent-level data (trust score, transaction count, operational tenure) that may reveal information about the developer's agent deployment patterns. Implementations SHOULD consider whether this level of disclosure is appropriate for their deployment context.

Agent private keys MUST NOT be transmitted to or stored by the Trust Authority in production deployments.

15. IANA Considerations

This document has no IANA actions.

16. References

16.1. Normative References

[FIPS186-4] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-4, July 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.

16.2. Informative References

[MCPS] Sharif, R., "Secure Model Context Protocol (MCPS): Cryptographic Message Security for AI Agent Communication", Internet-Draft draft-sharif-mcps-secure-mcp-00, March 2026.

[PSD2] European Parliament and Council, "Directive (EU) 2015/2366 on payment services in the internal market", November 2015.

[STRIPE-MPP] Stripe, Inc., "Machine Payments Protocol",

March 2026.

[PCI-DSS] PCI Security Standards Council, "Payment Card Industry Data Security Standard v4.0.1", March 2024.

Appendix A. PSD2 SCA Mapping

This appendix maps the protocol's mechanisms to PSD2 Strong Customer Authentication requirements. The cryptographic delegation certificate (Agent Passport) serves as the human principal's pre-signed authorisation, satisfying the "something the user has" factor through possession of the signing key used to create the delegation.

Full PSD2 mapping: See CyberSecAI EBA Position Paper (2026).

Appendix B. PCI DSS v4.0.1 Mapping

This appendix provides a high-level mapping of the protocol to PCI DSS v4.0.1 requirements. The protocol's cryptographic signing, access controls, audit logging, and anomaly detection address requirements across all 12 PCI DSS requirement families.

Full PCI mapping: See CyberSecAI PCI DSS Paper (2026).

Appendix C. Trust Level Reference Implementation

A reference implementation of the trust scoring model is available as an npm package (@proofxhq/agentpass) and an iOS Swift SDK, both with zero external dependencies.

Author's Address

Raza Sharif
CyberSecAI Ltd
United Kingdom

Email: contact@agentsign.dev
URI: https://agentpass.co.uk