SACA – Structural Architecture for Controllable Machine Behavior (v2.1)

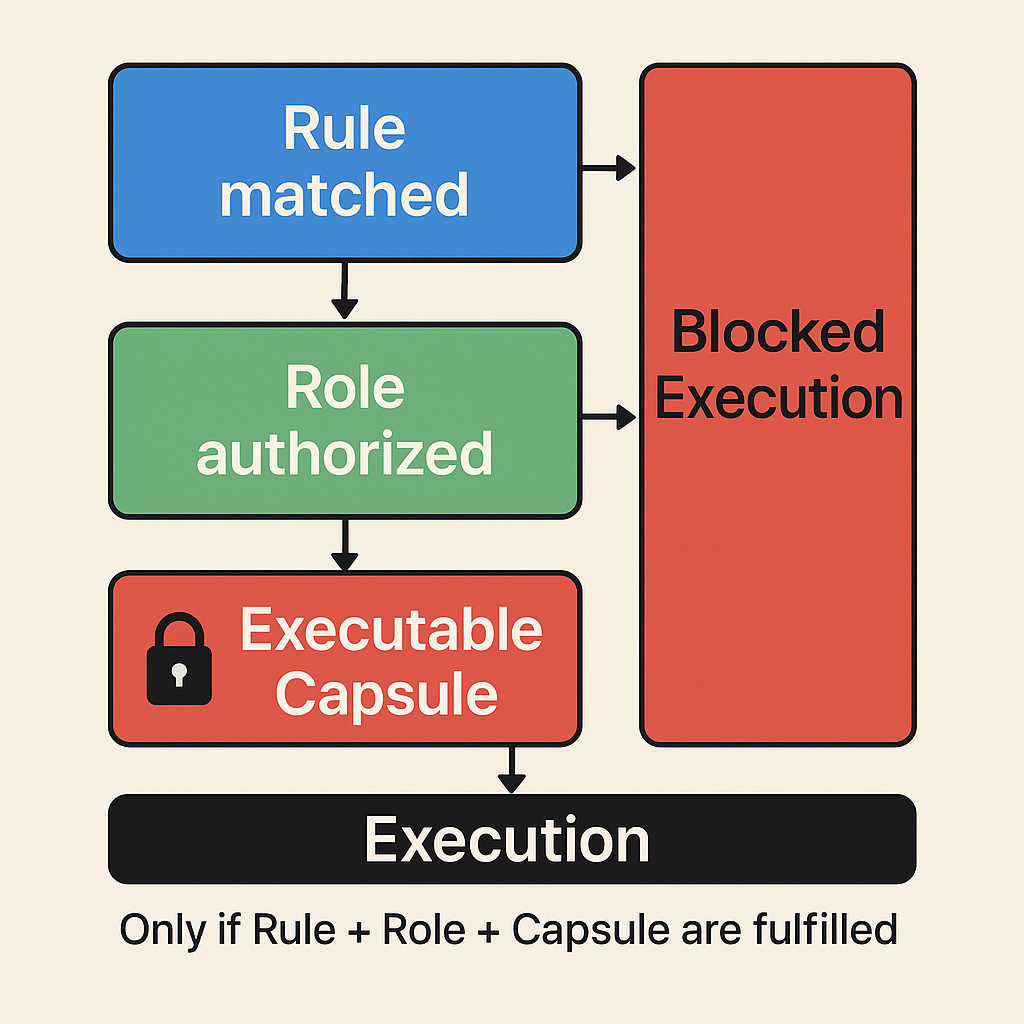
# Executive Summary

SACA is a structural execution architecture ensuring that digital behavior only occurs when three mandatory preconditions are simultaneously met: (1) a matched rule, (2) an explicitly authorized role, and (3) a signed, auditable capsule. If any of these are missing, execution is structurally blocked. This transforms digital control from interpretive governance to architectural enforcement. SACA applies to AI and non-AI systems alike. It is technology-agnostic, logic-bound, fallback-free, and audit-anchored.

# Reader’s Guide

|  |  |
| --- | --- |
| **Role / Expertise** | **Focus Sections** |
| System Architects | Part 1, Part 2, Part 3A |
| Legal / Compliance Professionals | Part 4, Part 5, Appendix A |
| Institutional and Political Leaders | Executive Summary, Part 5B, Part 6 |
| Auditors and Supervisory Authorities | Part 2, Part 3B, Appendix A, Part 6 |
| Open Source / Infrastructure Leaders | Part 3C, Part 6, Appendix C |

*MATCHED + ROLE AUTHORIZED + SIGNED CAPSULE PRESENT. Else: execution = BLOCKED]*

**

# Part 1: Principle of Structural Execution Binding

SACA defines execution as a binary permission model:  
- Rule: A deterministically matched control logic  
- Role: An assigned, cryptographically signed actor identity  
- Capsule: An isolated execution container with audit trace, expiry, and revocation  
  
Structural Blockade: If any precondition is missing → execution cannot occur.

# Part 2: System Requirements and Control Logic

A system is only SACA-compliant if it satisfies all of the following:  
- Pre-execution binding: rules and roles must be structurally locked before runtime  
- Signed capsule logic: execution units must be encapsulated, signed, versioned  
- Forkable traceability: every state must be independently auditable  
- Fallback-free behavior: no hot-patches, no overrides, no runtime exceptions

# Part 3A: System Architects – Technical Control Flow

Capsule Signing: Must use tamper-proof digital signature standards (e.g., Ed25519, ECDSA)  
Key Management: Role-based keys issued via governance authority, with rotation and revocation protocols  
Component View:  
- Policy Engine  
- Role Registry  
- Capsule Execution Core  
- Audit Layer (immutable + hash chained)  
- Interface Shields (e.g. API Gateway lockdown)  
Deployment Patterns: Compatible with containerized (K8s), microservice (sidecar), or event-driven architectures

## Execution Flow Example

1. Rule Match → via Policy Engine → POLICY-001  
2. Role Check → Registry validation of service-A.internal  
3. Capsule Creation → includes timestamp, rule-ID, role-ID  
4. Audit Trace → capsule log hash chained to ledger  
5. Execution → sanitize\_and\_export() in container  
6. Audit Completion → signed trace export

# Part 3B: Auditors – Trace Anchoring & Verifiability

Audit Artefacts: Timestamped rule reference, role-ID, execution hash, context capsule  
Log Verification: Structured correlation with IDW PS 330, ISO 19011 audit trails  
Review Conditions: Audit trace must be externally replicable, not self-reported  
Fork Validity: Forks must retain verifiable rule-role lineage and capsule continuity

## Sample Tools

- Hash Verifier: Capsule chain integrity checker  
- Role Decoder: Validates SPIFFE/SVID tokens  
- Policy Trace Map: Generates reasoning graph from audit capsule chain

## Tooling References (Real World)

- Sigstore Rekor: https://github.com/sigstore/rekor  
- Open Policy Agent (OPA): https://www.openpolicyagent.org  
- SPIFFE/SPIRE: https://spiffe.io

# Part 3C: OSS / Infrastructure Leaders

Forkability Enforcement: Right to audit and replicate must be structurally embedded  
Governance Anchors: Open source forks must retain audit integrity and role trace  
Runtime Integrity: DevSecOps compatible, capsule-integrity checks at CI/CD integration  
Toolchain Example: Sigstore, OPA, SPIFFE/SPIRE, Kyverno policy sets, GitOps trace anchoring

## Fork Scenario

- Replace key authority  
- Re-sign capsules  
- Publish capsule continuity hash diff

## Contributor Governance

- Contributor keys = GPG trust chain  
- All contributions = capsule-asserted  
- Maintainer sign-off = policy signature

## Practical Integration Note

While SACA itself is a system-independent architectural model, practical implementation questions—such as GitHub references or CI/CD templates—are addressed via reference systems.  
MaxOneOpen, the first full-scale SACA implementation, provides such an example. It includes:  
- Capsule-based execution logic  
- Fully auditable rule-role-event bindings  
- Public treaty interfaces  
- Policy-driven CI/CD integration

# Part 4: Legal Anchoring & Norm Alignment

## Norm Mapping

|  |  |
| --- | --- |
| **SACA Structural Element** | **Corresponding Norm** |
| Rule-Role Binding | NIS2 Art. 21(2c), ISO 27002 |
| Pre-execution Enforcement | ISO 27001 A.12.1.2 |
| Auditability by Design | IDW PS 982, AI Act Art. 12 |
| Forkable Control | AI Act Annex IX, EC OSS Guide |
| No-Fallback Architecture | COSO Principle 10 |

## Legal SLA Clause (Sample)

“All system-level behaviors under this contract are subject to structural pre-execution control under the SACA model. Execution is permitted exclusively upon verifiable rule-role binding and auditable capsule certification. Failure to meet these structural conditions constitutes a material breach of governance compliance.”

## Application Scenario

- Domain: Regulatory agency procurement  
- Requirement: Capsule-based event log with signed rule-role-reference

## Governance Policy Model

To structurally ensure SACA compliance within enterprise or public institutions, policies should define:  
- Binding of rules to enforceable roles  
- Designation of capsule issuance authorities  
- Provisions for third-party verifiability (audit interfaces)  
- Minimum structural guarantees (no override, no fallback, audit-or-fail logic)

# Part 5: Strategic Framing & Institutional Fit

## Meta-Comparison Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **SACA** | **ISO 27001** | **COSO** | **IDW PS 982** |
| Rule-bound Execution | ✅ Mandatory | ❌ Optional | ❌ Advisory | ⚠️ Partial |
| Role Locking | ✅ Signed | ⚠️ Role Docs | ❌ | ⚠️ Documented |
| Audit Anchoring | ✅ Cryptographic | ⚠️ Logging | ❌ Reports | ✅ Structural |
| Fallback Allowed | ❌ None | ✅ Yes | ✅ Yes | ✅ Case-by-case |
| Fork & Replicability | ✅ Required | ❌ | ❌ | ❌ |

## Policy & Institutional Scenarios

Government Use Case:  
- Digital Register Updates (e.g., Land/Company)  
 - SACA enforces: Only signed roles can trigger a change  
 - Rule source: National law + policy register

## Roadmap to Readiness

1. Identify critical execution zones  
   2. Implement open-access policy registry  
   3. Launch SACA-certified infrastructure pilot (capsule-first)

## Political Program Mapping

|  |  |
| --- | --- |
| **Initiative / Regulation** | **Relevance to SACA** |
| Digital Services Act (DSA) | Structural traceability, audit logging |
| Interoperable Europe Act | Modular control & verifiability across systems |
| DOME Initiative | Secure, enforceable digital infrastructures |
| Gaia-X | Policy-based execution, federation-level auditability |
| Data Governance Act | Enforcement over soft interpretation |

# Part 6: Licensing and Enforcement

- SACA is open, but conditional  
- Any system claiming SACA compliance must:  
 - Preserve rule-role integrity  
 - Maintain capsule audit trails  
 - Allow public fork replication  
- Systems allowing unauditable overrides or non-signed execution are not SACA-compliant

# Appendix A: Structural Enforcement vs Policy Governance

|  |  |  |
| --- | --- | --- |
| **Characteristic** | **Policy-based Governance** | **SACA** |
| Control Type | Advisory | Structural |
| Enforcement Moment | Post-hoc | Pre-execution |
| Responsibility Assignment | Interpretive | Signature-locked |
| System Trust Model | Human-mediated | Machine-verifiable |

# Appendix B: Structural Security Measures

SACA systems enforce security by structural impossibility:  
- No mutable execution paths  
- No runtime injection  
- No role spoofing  
- No unsigned behavior allowance

# Appendix C: Community and Infrastructure Relevance

- Forkable doesn’t mean vulnerable – forks are licensed, signed, and auditable  
- Community governance anchored by capsule-continuity and signature verification  
- Supports contributor integrity models in OSS environments (e.g. CNCF, OpenSSF)

# Glossary (Extract)

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Capsule | Isolated execution unit with structural preconditions |
| Forkability | Right to independently verify, copy, and re-anchor a system implementation |
| Execution Blockade | Prevention of execution when any condition is unmet |
| Role Binding | Cryptographically enforced identity-permission linkage |

# Final Note

SACA is not optional, not approximate, and not suggestive. It is binary: either structurally enforced, or non-compliant.  
  
It redefines digital responsibility not as an interpretive process – but as an execution condition.  
  
Nothing runs unless responsibility is embedded.