Digital Sovereignty Through Explicit Authorization

Cryptographic Enforcement of Individual Control in Open Systems

February 21, 2026

Mats Heming Julner

Abstract

Digital infrastructure routinely requires individuals to delegate operational power over assets, identity, or computation to external entities. In prevailing architectures, such delegation frequently becomes persistent, implicit, or difficult to revoke, creating structural authority asymmetries between users and intermediaries.

This paper formalizes explicit authorization as a cryptographic design discipline that preserves individual sovereignty while enabling scalable coordination. We identify the minimal properties required for users to retain ultimate control over execution, present a reference enforcement architecture, and analyze the security guarantees that emerge when revocation, bounded scope, domain separation, and replay protection are treated as mandatory invariants. Under this approach, reliance on institutional trust becomes optional rather than foundational. Systems may still incorporate operators, markets, and governance frameworks, but the capacity for action remains cryptographically contingent on demonstrable, current consent. Institutions may remain useful but are no longer structurally necessary for enforcement of authority. Authority becomes conditional rather than structurally embedded.

1. Introduction

The growth of digital finance, online identity, and automated coordination has intensified a long-standing tension: enabling third parties to act efficiently on behalf of users without permanently transferring authority to them. Historically, delegation has been implemented through custody, administrator privileges, or standing permissions. While operationally convenient, these methods tend to accumulate risk. Once granted, authority often persists beyond the user's awareness or practical ability to retract it.

Recent advances in cryptographic signing standards and verifiable computation allow an alternative approach: execution contingent on explicit, user-originated authorization that is narrow, time-bounded, and revocable. This paper examines how such mechanisms can serve as the foundation of digital sovereignty, defined here not as political independence, but as the continued capacity of an individual to determine whether and how actions affecting them may occur.

2. Defining Digital Sovereignty

For purposes of this work, an individual remains sovereign within a system if they retain continuous, system-enforceable ability to:

1. deny execution,
2. revoke previously granted permission,
3. limit quantitative exposure,

4. restrict contextual applicability,

5. audit historical usage,

6. change or remove executors.

Sovereignty in this sense is mechanical rather than political. It refers to properties enforced by the system's verification rules, not to social or legal autonomy. If these conditions are not met, authority migrates structurally toward operators or infrastructure providers, regardless of nominal ownership.

3. Failure Modes in Existing Architectures

Common mechanisms that erode sovereignty include:

• perpetual token allowances,

• omnibus custody arrangements,

• upgradeable contracts with administrator keys,

• governance overrides,

• implicit renewal of permissions,

• cross-domain replay,

• reliance on operator abstention rather than technical prevention.

Such systems may function effectively but embed latent dependence on institutional behavior. For example, a token approval that remains valid indefinitely until manually revoked creates persistent latent authority. If the user forgets or is unable to revoke it, the system continues to treat past consent as current permission, thereby violating revocation dominance.

4. Explicit Authorization as a Design Discipline

We define explicit authorization as a requirement that every act of execution must be accompanied by verifiable proof of valid consent originating from the affected principal. This transforms delegation from a durable transfer into a continuously evaluated predicate enforced at the point of execution. The burden of correctness shifts from organizational assurances to mechanical validation.

5. Cryptographic Building Blocks

A sovereign authorization framework minimally requires:

5.1 Identity

Stable cryptographic representation of principals.

5.2 Typed Intent

Unambiguous description of permitted actions.

5.3 Domain Separation

Prevention of unintended portability across contexts.

5.4 Nonce or Uniqueness Constraint

Defense against replay.

5.5 Validity Interval

Automatic expiration.

5.6 Capability Bounding

Quantitative and functional limits.

5.7 Revocation Facility

Grantor-controlled invalidation.

5.8 Deterministic Verification

Objective enforcement independent of operator discretion.

6. Authorization Objects

These elements can be composed into signed data structures, authorization objects, that travel with execution requests. Crucially, the signature is not evidence of past approval; it is a prerequisite for present action.

7. Revocation Dominance

A defining feature of sovereignty-preserving systems is that revocation must dominate prior consent. Explicit withdrawal of authorization MUST render any previously issued permission unusable, regardless of remaining validity window or executor cooperation. Revocation must not depend on temporal expiry, grantee acknowledgment, or institutional mediation. It must be enforceable directly through system verification rules. If historical permission can survive explicit withdrawal, sovereignty becomes conditional on institutional behavior rather than cryptographic enforcement. For this reason, revocation must be strictly stronger than prior consent. Implementations typically rely on registries, state transitions, or monotonic versioning that validators are required to consult before honoring a request.

8. Reference System Topology

A generic architecture may include:

• user agents producing authorizations,

• storage or indexing layers tracking status,

• executors submitting actions,

• verification environments enforcing rules,

• observers reconstructing history.

Trust in operators is replaced by a requirement for verifiable proof.

Core Invariant

No state transition affecting a principal is valid unless accompanied by a cryptographically verifiable authorization satisfying scope, domain, temporal, and revocation constraints.

9. Security Properties

When rigorously implemented, explicit authorization yields:

• non-escalation of privilege,

• replay resistance,

• bounded exposure,

• executor substitutability,

• survivability of institutional failure.

10. Institutional Participation Without Structural Dependence

Institutions may continue to provide valuable services: aggregation, liquidity, dispute resolution, compliance tooling. However, their operational power is derived from user consent, not embedded entitlement. A user can withdraw cooperation without requiring institutional consent.

11. Related Work and Relationship to Prior Paradigms

The ideas presented here build upon a substantial body of research in distributed security and capability systems. Capability-based security architectures have long explored fine-grained delegation via unforgeable references rather than identity-based access control. Public-key infrastructures introduced portable cryptographic assertions of authority. OAuth and related web authorization frameworks enabled delegated access with revocation and scope restriction in client-server environments. Smart contract ecosystems further extended delegation through token approvals, permit signatures, meta-transactions, and typed data standards.

This work does not replace these traditions. Instead, it elevates them from implementation techniques to mandatory execution invariants. The central claim is that sovereignty depends not merely on the availability of authorization mechanisms, but on the requirement that every act of execution be contingent upon them.

11.1 On-Chain Authorization Standards

Existing standards such as EIP-2612 (permit) and EIP-3009 (transferWithAuthorization) demonstrate practical deployments of signed, typed authorizations in production environments. These standards enable gasless approvals and signature-based transfers but typically treat authorization as tightly coupled to a specific execution path.

The framework proposed in this paper generalizes such mechanisms by elevating authorization from a transaction convenience to a mandatory execution invariant. In particular, revocation dominance, domain separation rigor, and explicit replay constraints are treated here not as optional features but as structural requirements for sovereignty preservation.

12. Implementation Considerations

Real-world deployments must balance security with usability. A particularly critical challenge is key custody and recovery. Mechanical sovereignty at the execution layer depends on control of private keys; loss or compromise of those keys undermines enforceable control regardless of authorization design. Recovery mechanisms such as social recovery, multisignature guardianship, or trusted hardware must therefore be designed carefully to avoid reintroducing structural authority

persistence. This paper does not prescribe a specific custody model but assumes that any recovery scheme must preserve revocation dominance and prevent unilateral expansion of delegated authority. Open challenges include:

• key custody and recovery,

• human-readable intent expression,

• privacy of authorization data,

• efficient revocation dissemination,

• cross-chain equivalence.

13. What Explicit Authorization Cannot Guarantee

No cryptographic system can prevent voluntary re-delegation, coercion, or external legal compulsion. Sovereignty here concerns enforceable system behavior, not broader socio-political dynamics. Mechanical sovereignty constrains what a system will execute under its rules; it does not eliminate coercion, legal compulsion, or social pressure external to the system. The practical value of explicit authorization therefore varies by threat model: it provides strongest protection against unintended automation, institutional drift, and latent permission persistence, but offers limited protection against direct coercive control over key holders. This distinction does not diminish the invariant; it clarifies its scope.

14. Research Directions

Areas for continued exploration include:

• formal verification of authorization invariants,

• zero-knowledge consent proofs,

• programmable revocation markets,

• interoperability standards.

15. Conclusion

By conditioning execution on verifiable, current permission, explicit authorization enables participation in complex digital ecosystems without irreversible authority transfer. Institutions remain possible. Dependence on them becomes optional. This work is both a conceptual clarification and a minimal normative specification. It does not mandate a particular implementation but defines invariants that conformant systems must satisfy. While this paper defines mandatory

invariants for systems claiming to preserve digital sovereignty, it does not prescribe a specific contract architecture, storage model, or custody scheme. Multiple implementations may satisfy the same invariants. The objective of this framework is not to eliminate institutions but to ensure that their authority derives from verifiable user consent rather than structural necessity. In doing so, it establishes a foundation for digital systems in which sovereignty is mechanically preserved rather than institutionally promised.

Clarification before Appendices

Status of Normative Material

Sections within the appendices contain prescriptive requirements expressed using normative language (MUST, SHOULD, MUST NOT). These define the minimum properties necessary for implementations that aim to preserve digital sovereignty as described in the body of the paper. The main text provides conceptual framing and rationale; the appendices specify enforceable construction.

APPENDIX A: Normative Requirements (RFC Style)

A conformant implementation:

MUST

• verify signatures against canonical encoding,

• enforce domain separation,

• check validity intervals,

• enforce capability bounds,

• consult revocation state,

• prevent replay,

• produce an auditable record.

MUST NOT

• execute based solely on past approval,

• assume executor honesty,

• infer permissions not explicitly present.

Failure to satisfy any MUST requirement voids the sovereignty guarantees described in this paper.

APPENDIX B: Canonical Authorization Object (Minimum Form)

```
Authorization {
    address grantor;
    address grantee;
    address token;
    uint256 maxPerUse;
    uint256 validAfter;
    uint256 validBefore;
    bytes32 nonce;
}
```

Semantics:

• grantor → authority origin

• grantee → permitted executor

• token/resource → scope

• maxPerUse → quantitative cap

• validity window → temporal bound

• nonce → uniqueness / replay defense

Nonce uniqueness MUST be enforced at minimum per (grantor, domain) pair unless a stronger constraint is specified. Implementations MAY enforce global uniqueness, but MUST ensure that a nonce cannot be reused in a manner that permits replay or unintended authorization reuse. This minimal form assumes that action-type constraints are enforced by the verifying environment (e.g., a contract that interprets the authorization within a single execution path). Implementations requiring explicit action-type binding SHOULD incorporate an additional typed intent field (e.g., a function selector or structured intent hash) into the signed payload to prevent cross-function ambiguity.

APPENDIX C: Cryptographic Construction

```
AUTH_TYPEHASH = keccak256(
  "Authorization(address grantor,address grantee,address token,uint256 maxPerUse,uint256 validAfter,uint256 validBefore,bytes32 nonce)"
)

structHash = keccak256(
  abi.encode(
    AUTH_TYPEHASH,
```

```
    grantor,
    grantee,
    token,
    maxPerUse,
    validAfter,
    validBefore,
    nonce
  )
)
```

```
digest = keccak256(0x1901 || domainSeparator || structHash)
signature = sign(grantor_private_key, digest)
```

Verification MUST recover grantor. The domainSeparator MUST include a unique identifier for the execution environment sufficient to prevent cross-domain replay. At minimum, this SHOULD bind the authorization to a specific chain identifier, verifying contract address, or application namespace. Implementations that omit environment-specific domain separation risk unintended portability and replay across execution contexts.

APPENDIX D: Minimal Verifier (Pseudocode)

The following pseudocode assumes a trusted source of current time within the execution environment.

```
function verify(auth, sig):
    require(currentTime >= auth.validAfter)
    require(currentTime <= auth.validBefore)
    require(!revoked(auth))
    require(!nonceUsed(auth.nonce))

    signer = recover(digest(auth), sig)
    require(signer == auth.grantor)

    markNonceUsed(auth.nonce)
```

This minimal verifier assumes single-use or per-execution bounded authorizations. Implementations supporting reusable or cumulative authorizations MUST maintain monotonic accounting sufficient to prevent exceeding declared quantitative limits. Failure to track cumulative usage reintroduces latent authority expansion and violates bounded capability guarantees.

APPENDIX E: Revocation Models

Acceptable approaches include:

• explicit hash cancellation

• epoch/version invalidation

• monotonic sequence supersession

Revocation MUST require computational and economic cost less than or equal to that of exercising the authorization it invalidates. In execution environments with dynamic fee markets or congestion, implementations MUST ensure that revocation cannot be economically or procedurally censored relative to execution of the authorization itself. If exercising an authorization is reliably cheaper or more accessible than revoking it, revocation dominance becomes illusory in practice. Systems aiming to preserve sovereignty should therefore evaluate revocation accessibility under adversarial network conditions.

APPENDIX F: Common Failure Modes

• incorrect domain separator

• mismatched encoding

• signature pre-hashing mistakes

• forgetting nonce consumption

• allowing executor substitution

• optional revocation checks

APPENDIX G: Test Vector Template

Implementations SHOULD publish:

• example Authorization

• structHash

• domainSeparator

• digest

• signature

• expected recovery result

Deviation from expected values indicates incompatibility with this specification.